Средства для создания надежных центров обработки данных

Обеспечение высокой доступности систем на основе MySQL

Чарльз Белл
Мэтс Киндал
Ларс Талманн
Предисловие Марка Каллагана

O'REILLY® РУССКАЯ РЕДАКЦИЯ bhv®

MySQL High Availability

Charles Bell,
Mats Kindahl,
and Lars Thalmann

O'REILLY®

Обеспечение высокой доступности систем на основе MySQL

Чарльз Белл
Мэтс Киндал
Ларс Талманн
Предисловие Марка Каллагана

РУССКАЯ РЕДАКЦИЯ bhv®

2012

УДК 681.3.068
ББК 32.973.26–018.1
Б43

Чарльз Белл, Мэтс Киндал и Ларс Талманн

Б43 Обеспечение высокой доступности систем на основе MySQL / Пер. с англ. — М. : Издательство «Русская редакция» ; СПб. : БХВ-Петербург, 2012. — 624 стр. : ил.

ISBN 978-5-7502-0409-0 («Русская редакция»)
ISBN 978-5-9775-0799-8 («БХВ-Петербург»)

Данная книга — подробное руководство по обеспечению высокой доступности ИТ-систем, построенных с использованием СУБД MySQL.

Здесь рассматриваются методические приемы и функции, раскрываются общие подходы и тонкости, связанные с репликацией и мониторингом серверов баз данных. Авторы — лидеры команды разработчиков MySQL и признанные эксперты в области теории и практики применения СУБД — приводят множество реальных примеров, сопровождая их подробным анализом. «Изюминкой» книги является рассказ о недокументированных и неочевидных функциях, позволяющих повысить отказоустойчивость MySQL в любой среде — в среде «обычных», виртуальных и кластерных серверов, а также в облачных вычислениях.

Издание состоит из пятнадцати глав и предметного указателя. Рекомендуется системным администраторам, администраторам БД и всем интересующимся практическими аспектами повышения надежности ИТ-систем.

УДК 681.3.068
ББК 32.973.26–018.1

Чарльз Белл, Мэтс Киндал, Ларс Талманн

Обеспечение высокой доступности систем на основе MySQL

Совместный проект издательства «Русская редакция» и издательства «БХВ-Петербург».

РУССКАЯ РЕДАКЦИЯ bhv®

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 21.09.11. Формат $70\times100^{1}/_{16}$.
Печать офсетная. Физ. печ. л. 39. Тираж 1 500 экз. Заказ №

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-0-596-80730-6 (англ.)
ISBN 978-5-7502-0409-0 («Русская редакция»)
ISBN 978-5-9775-0799-8 («БХВ-Петербург»)

Оглавление

Предисловие

Репликация — объект многочисленных исследований. На основе их результатов было создано много опытных вариантов, большинство из которых так и не перешло в разряд рабочих систем. Напротив, репликацию MySQL широко используют на практике, но она не имеет теоретической базы. В этой книге мы постараемся исправить такое положение вещей. Здесь вы найдете ответы на вопросы, на которые раньше могли ответить лишь те немногие, кто с головой ушел в исходники, проводя дни, а порой и ночи за отладкой.

Репликация позволяет организовать сервисы данных высокой доступности, устойчивые к неизбежным сбоям. Причин сбоя может быть великое множество, включая неисправность диска, сервера или центра обработки данных. Даже если оборудование идеальное и полностью дублировано, вспомните человеческий фактор. Ошибочно удаленные из БД таблицы, запись приложениями неверных данных — все это причины сбоев. Однако, при надлежащей подготовке, восстановление гарантировано после любого сбоя. Ключи к спасению — это избыточность данных и резервное копирование. В репликации MySQL поддерживаются обе эти возможности.

Репликация MySQL позволяет не только выполнить восстановление после сбоя. Ее часто используют для масштабирования операций чтения. В системе MySQL вы можете эффективно выполнять репликацию на большое количество серверов. Это экономически выгодная стратегия, обеспечивающая обработку большого количества запросов чтения, генерируемых приложениями, на обычном оборудовании.

У репликации MySQL есть и другие полезные свойства. Возьмем онлайн-DDL — компонент сложных коммерческих СУБД. В MySQL онлайн-DLL не поддерживается, однако средствами репликации у вас получится реализовать нечто подобное. Вообще, если подойти к репликации творчески, то можно многого добиться.

Репликация — один из компонентов, которому СУБД MySQL обязана своей популярностью. Среди прочего, она позволяет преобразовать систему любительского уровня на основе MySQL в успешное критически важное приложение для бизнеса. Как и многое, что связано с MySQL, репликация отличается простотой, поэтому возможны накладки при работе в производственной среде. В этой книге рассказано, как правильно пользоваться репликацией MySQL. В ней вы узнаете то, как реализована репликация, где ее узкие места, как предупредить возникновение сбоев и как с ними бороться, если они все-таки возникли.

Репликация MySQL постоянно совершенствуется. Это тернистый путь проб и ошибок, вследствие чего репликация становится все более эффективной, надежной и интересной. Так, в MySQL 5.1 предложен метод построчной репликации.

Несмотря на многообразие развертываний MySQL, я занимаюсь в основном сервисами данных для интернет-приложений, в этой области идея репликации СУБД MySQL на распределенные системы хранения, такие как HBase и Hadoop, имеет огромный потенциал. Так удается и совершенствовать применение MySQL в центрах обработки данных.

Мне довелось участвовать в группах поддержки развертываний MySQL для Facebook и Google, что дало возможность и время изучать многое из того, что описано в этой книге. Авторы книги — эксперты по репликации MySQL, — и готовы поделиться с вами опытом.

Марк Каллаган (Mark Callaghan)

Введение

На протяжении многих лет авторы этой книги создают СУБД на основе MySQL и работают с ними. Чарльз Белл (Charles Bell) — старший разработчик направления репликации и резервного копирования. В круг его интересов входят все аспекты MySQL, теория баз данных, проектирование ПО и гибкая методология разработки ПО. Доктор Мэтс Киндал (Mats Kindahl) — ведущий разработчик направления репликации в группе MySQL Backup and Replication (Репликация и резервное копирование MySQL). Он является главным архитектором и воплотителем идеи построчной репликации в MySQL, и, кроме того, создал платформу модульного тестирования для MySQL. Доктор Ларс Талманн (Lars Thalmann), руководитель отдела разработки и технический руководитель группы MySQL Replication and Backup, создал множество компонентов репликации и резервного копирования. Принимал участие в разработке технологий кластеризации, репликации и резервного копирования MySQL.

Много отличных изданий посвящено MySQL, и только некоторые из них освещают дополнительные возможности, такие как высокая доступность, надежность и удобство в обслуживании. Наша книга призвана восполнить пробелы в знаниях о MySQL. В ней рассмотрены все эти, а также некоторые другие темы.

Чтобы облегчить восприятие книги, мы включили в нее рассказ о специалисте по MySQL, решающем повседневные задачи, которые ставит перед ним руководство. Его зовут Джоэл Томас, он новый сотрудник в компании, делающей первые шаги в освоении MySQL. На ваших глазах Джоэл изучает MySQL, пытаясь решить сложнейшие проблемы, стоящие перед специалистами MySQL. Надеемся, наш рассказ не будет скучным.

Кому адресована книга

Книга предназначена для специалистов в области MySQL. Читатель должен знать основы языка SQL, администрирования СУБД MySQL и операционной системы, на которой он работает. Мы, в свою очередь, представили дополнительные сведения о репликации, аварийном восстановлении, системном мониторинге и других ключевых темах обеспечения высокой доступности. Список книг, которые можно почитать по теме, вы найдете в главе 1.

Как работать с книгой

Книга состоит из трех частей. Первая часть охватывает тему репликации MySQL, включая высокую доступность и масштабируемость. Во второй части рассмотрены вопросы мониторинга и производительности при построении надежных центров данных. Часть III посвящена дополнительным областям MySQL, включая облачные вычисления и кластеры MySQL.

Часть I Репликация

- Глава 1 *Введение*. Обзор тематики и аудитории данной книги.
- Глава 2 *Основы репликации MySQL*. Здесь рассмотрены процедуры по ручной и автоматической настройке простейшей репликации.
- Глава 3 *Двоичный журнал событий*. Рассказ о критическом файле, предназначенном для сборки репликации и позволяющем выполнять аварийное восстановление, поиск неисправностей и прочие административные задачи.
- Глава 4 *Роль репликации в обеспечении высокой доступности*. Описание способов восстановления после сбоя сервера, в том числе, и с применением автоматизированных сценариев.
- Глава 5 *Роль репликации MySQL в горизонтальном масштабировании*. Описание способов и топологий, позволяющих улучшить время отклика и обрабатывать большие наборы данных.
- Глава 6 *Дополнительные возможности репликации*. Обзор таких тем, как безопасная передача данных и построчная репликация.

Часть II Мониторинг и восстановление после сбоев

- Глава 7 *Основы мониторинга*. Обзор основных параметров ОС, которые необходимо отслеживать, и средств наблюдения за ними.
- Глава 8 *Мониторинг MySQL*. Знакомство со средствами мониторинга активности и производительности базы данных.
- Глава 9 *Мониторинг механизмов БД*. Более подробный рассказ о некоторых параметрах, за которыми необходимо вести наблюдение, на примерах, характерных для MyISAM или InnoDB.
- Глава 10 *Мониторинг репликации*. Подробная информация о том, как отслеживать происходящее на главных и подчиненных серверах.
- Глава 11 *Устранение неполадок репликации*. Ваши действия при сбоях и перезапусках, повреждении данных и других происшествиях.
- Глава 12 *Защита инвестиций*. Применение способов резервного копирования и аварийного восстановления.
- Глава 13 *MySQL Enterprise*. Обзор инструментальных средств, позволяющих упростить выполнение многих задач, представленных в предыдущих главах.

Часть III Среды с высокой доступностью

- Глава 14 *Облачные вычисления*. Знакомство с наиболее популярной службой, построенной на основе облачных вычислений, Amazon.com AWS, и обзор способов использования системы MySQL в подобной виртуальной среде.
- Глава 15 *MySQL Cluster*. Посвящается использованию данного средства для достижения высокой доступности.

- Приложение *Репликация: секреты* и *советы*. Набор процедур, незаменимых в определенных ситуациях.

Условные обозначения

В книге использованы следующие условные обозначения:

- Простой текст Используется для обозначения заголовков меню, параметров и кнопок.
- *Курсив* Так выделены новые термины, имена таблиц баз данных; адреса URL, электронной почты, имена файлов и утилит Unix.
- `Моноширинный шрифт` Параметры командной строки, переменные и другие элементы кода, содержимое файлов и вывод команд.
- **`Моноширинный шрифт полужирный`** Команды или другие виды текста, который пользователь должен ввести дословно.
- *`Моноширинный шрифт курсивный`* Текст, вместо которого пользователь должен подставить свои значения.


Под этим значком вы найдете подсказки, советы или общие примечания.


Таким значком обозначено предупреждение или предостережение.

Использование листингов

Эта книга призвана помогать вам в работе. Вообще, вы можете использовать приведенный в ней код в своих программах или документации. Разрешения на использование не требуется (если только вы не копируете значительную часть кода). Так, для написания программы, содержащей несколько фрагментов кода из данной книги, разрешения не требуется. Для продажи или распространения компакт-диска с примерами кода издательства O'Reilly *требуется* разрешение. Если книга и листинг цитируется в ответе на вопрос, разрешение не требуется. Для вставки объемного листинга из данной книги в документацию к вашей продукции разрешение *необходимо*.

Мы будем признательны вам за ссылки на авторов, хотя это и не обязательно. При этом, обычно, указывают название книги, автора, издательство и ISBN. Пример: «MySQL High Availability, by Charles Bell, Mats Kindahl, and Lars Thalmann. Copyright 2010 Charles Bell, Mats Kindahl, and Lars Thalmann, 9780596807306».

Если вы полагаете, что, используя примеры кода, вы превысили оговоренный выше лимит, не стесняйтесь — обращайтесь к нам по адресу: *permissions@oreilly.com*.

Обратная связь

Каждый листинг из этой книги был протестирован на различных платформах. Содержащаяся в книге информация также была проверена на всех этапах производственного процесса. Тем не менее, могут быть ошибки и «сырые» места, сведения о которых мы с благодарностью примем, наряду с любыми пожеланиями, сделанными вами для будущих изданий. С автором или редакторами можно связаться по адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (для США и Канады)
707-829-0515 (для остальных стран)
707-829-0104 (факс)

Веб-страница, посвященная книге, со списком опечаток, выдержками и дополнительными сведениями, находится по адресу:

http://www.oreilly.com/catalog/9780596807306.

Чтобы оставить комментарий или задать технические вопросы по книге, отправьте сообщение электронной почты с указанием номера ISBN (9780596807306) книги по адресу:

bookquestions@oreilly.com.

Дополнительные сведения о наших книгах, конференциях, информационных центрах и сети O'Reilly Network вы найдете на нашем сайте:

http://www.oreilly.com.

Библиотека Safari® Books Online

Служба Safari Books Online — это поисковая цифровая библиотека, позволяющая быстро находить ответы на интересующие вас вопросы среди 7500 справочных изданий и видеосюжетов технического и творческого направлений.

Оформив подписку, вы сможете прочитать любую страницу и просмотреть любое видео из нашей библиотеки в режиме онлайн. Также доступно чтение книг в мобильном телефоне и портативных устройствах. Возможен доступ к печатным изданиям до выхода тиража, исключительный доступ к рукописям и обратная связь с авторами и копирование листингов, создание избранного, загрузка глав, создание закладок, заметок, печать страниц и масса других функций, позволяющих экономить время.

Издательская компания O'Reilly Media выгрузила эту книгу на сервер службы Safari Books Online. Чтобы получить полный доступ к этой и другим книгам подобной тематики издательства O'Reilly и других издательств, пройдите бесплатную регистрацию по адресу: *http://mysafaribooksonline.com.*

Благодарности

Авторы выражают благодарность техническим редакторам Марку Каллагану (Mark Callaghan), Луису Суаресу (Luis Soares) и Моргану Токеру (Morgan Tocker) за внимание к мелочам и дальновидные деловые советы. Без вашей помощи хорошей книги не получилось бы.

Мы также хотим поблагодарить наших весьма одаренных коллег по группе разработчиков средств репликации MySQL Альфранио Корейа (Alfranio Correia), Андрей Елкин (Andrei Elkin), Чжень-Синь Хи (Zhen-Xing He), Сергей Козлов (Serge Kozlov), Свен Сандберг (Sven Sandberg), Луис Суарес (Luis Soares), Рафаль Сомла (Rafal Somla), Ли-Бинь Сонг (Li-Bing Song), Инго Струинг (Ingo Strüwing) и Дао-Гань Ку (Dao-Gang Qu) за их преданный труд, благодаря которому репликация MySQL стала надежным и мощным средством. Особую благодарность выражаем специалистам службы техподдержки MySQL — они создали мост между потребностями потребителей и нами с нашим собственным взглядом на продукт. Также, мы благодарны членам сообщества, которые не жалели своих сил и времени на то, чтобы улучшить MySQL для всех.

И наконец, самое главное: мы благодарны нашему редактору, Энди Орему (Andy Oram), за помощь в оформлении работы и смирение с нашим иногда переливающим через край энтузиазмом по поводу MySQL.

Чарльз благодарит любящую жену Аннетт за терпение и понимание того, что во время работы над книгой приходилось жертвовать семейными делами. Ты — любовь и вдохновение. Кроме того, Чарльз выражает благодарность коллегам по группе разработчиков MySQL из Oracle — всем, кто помогал день ото дня, не требуя наград. И наконец, Чарльз благодарен всем братьям и сестрам во Христе за каждодневное испытание и поддержку.

Мэт благодарен своей жене Лил и сыновьям Йону и Ханесу за любовь и понимание в трудные времена. Не представляю жизнь и любовь без вас. Мэт также благодарен коллегам по MySQL в компаниях Sun и Oracle и не только. Вы, без преувеличения, одни из самых острых умов этого бизнеса. Все было интересно и вдохновенно.

Ларс благодарен всем своим коллегам — бывшим и настоящим, — сделавшим MySQL столь интересным местом работы. На самом деле, это больше, чем работа. Меня поражает разносторонняя натура членов группы разработки MySQL и широта взглядов многих преданных разработчиков. Дух сообщества MySQL — вот что превращает работу в почетную миссию. То, что мы создали вместе, выше похвал. Просто удивительно, как поначалу столь небольшая группа людей умудрилась создать продукт, на котором сегодня работают многие компании из списка «Fortune 500».

ЧАСТЬ I

Репликация

Ваша задача номер один при обеспечении надежности среды MySQL — настроить репликацию. Полученные здесь навыки пригодятся вам позднее, когда вы будете настраивать другие аспекты среды, обеспечивающие высокую доступность, и управлять ими.

ГЛАВА 1

Введение

Джоэл просматривал объявления в поисках новой работы. Его нынешняя работа — не самая плохая, да и компания помогала ему во время учебы в колледже. Правда, диплом он получил несколько лет назад, а карьерный рост его пока еще оставлял желать лучшего.

— Неплохо, — сказал он, обводя объявление. Требовался специалист по вычислительной технике и навками работы с MySQL. У него был и опыт работы с MySQL и соответствующее образование. Просмотрев еще несколько объявлений, он решился-таки позвонить насчет работы с MySQL. После нескольких поверхностных вопросов менеджер по кадрам назначила ему собеседование через два дня.

В назначенный день он прошел три собеседования, прежде чем был представлен президенту и исполнительному директору компании, Роберту Саммерсону, для решающего экзамена на профпригодность. В перерывах между вопросами г-н Саммерсон заглядывал в свои записи. До сих пор вопросы по информационным технологиям носили общий характер, но Джоэл знал, что жесткий «допрос» по MySQL не за горами.

Наконец, экзаменатор сказал:

— Вы неплохо подготовились, господин Томас. Можно — просто Джоэл?

— Да, конечно, — ответил Джоэл и получил в награду еще один трудный период, в ходе которого экзаменатор в третий раз просмотрел свои записи.

— Что вы знаете о MySQL? — Г-н Саммерсон положил руки на стол, одарив его пронзительным взглядом.

Джоэл принялся объяснять, что он знает о MySQL. Собрал в кучу весь материал, прочитанный в ночь перед собеседованием. Примерно через 10 минут он исчерпал себя.

Г-н Саммерсон подождал пару минут, затем встал, протянул Джоэлу руку и сказал:

— Джоэл, вы сказали все, что я хотел услышать. Вы приняты.

— Спасибо.

Г-н Саммерсон жестом пригласил Джоэла следовать за ним.

— Сейчас мы зайдем в отдел кадров, и вас оформят на работу. Две недели — испытательный срок. Вы сможете начать с понедельника?

От радости Джоэл расплылся в улыбке: «Да, конечно».

— Отлично. Г-н Саммерсон еще раз пожал ему руку и сказал:

— Готовьтесь к оценке конфигурации моих MySQL серверов. Мне понадобится полный отчет о конфигурации и работоспособности.

Радость постепенно проходила, когда, выезжая со стоянки, Джоэл не поехал домой, а свернул к ближайшему книжному магазину. «Нужна серьезная книга по MySQL», — думал он.

Итак, вы решились на установку и эксплуатацию среды MySQL. Тогда впереди у нас немало интересного и полезного.

В отличие от небольшой площадки, для поддержки крупного предприятия необходимо планирование, предвидение, опыт и еще раз планирование. От вас, как от администратора или будущего администратора базы данных крупного предприятия, требуется следующее.

- Разработка планов восстановления критических бизнес-данных в случае катастрофы. Желательно выполнить эту процедуру для проверки хотя бы один раз.
- Разработка планов по управлению клиентской или пользовательской базой и мониторингу нагрузки на каждый из узлов сайта с целью оптимизации.
- Планирование оперативного масштабирования в условиях быстрого роста пользовательской базы.

Во всех подобных случаях крайне важно предвидеть возможные события и подготовиться оперативно на них реагировать.

Веб-сайты не единственные приложения, использующие большое количество серверов. Поэтому сервер для поддержки определенного приложения здесь может обозначаться термином *развертывание* (deployment) вместо *сайт* (site) или *веб-сайт* (website). На самом деле это может быть вебсайт, CRM-система (систем управления взаимоотношениями с клиентами, customer relationship management) или онлайновая игра. Книга посвящена уровню базы данных таких систем, но некоторые примеры касаются интеграции прикладного уровня и уровня БД.

Есть две вещи, необходимые для поддержки быстродействия и доступности сайта: резервное копирование данных и избыточность в системе. Резервные копии позволяют восстановить узел до состояния, предшествующего сбою, а избыточность позволит продолжать работу сайта даже после прекращения работы одного или нескольких узлов.

Существует много способов резервного копирования — выберите для себя наиболее подходящий. Нужно восстановить состояние данных на заданный момент времени — проверьте, что у вас есть все необходимое для процедуры восстановления состояния на определенный момент времени (point-in-time recovery — PITR). Если во время резервного копирования серверы должны оставаться в работе, воспользуйтесь одной из форм оперативной архивации.

Избыточность достигается дублированием оборудования, параллельной работой нескольких экземпляров программ и репликацией для создания идентичных копий данных на нескольких компьютерах. Это позволяет при выходе из строя одного компьютера подключиться к другому, на котором есть копия данных.

Кроме репликации важную роль в масштабировании системы, а при необходимости, и добавлении новых узлов, играет резервное копирование. Если все сделать правильно, то для добавления нового подчиненного сервера достаточно будет, образно говоря, просто нажать на кнопку.

Что такое репликация?

Если вы читаете эту книгу, то, вероятно, знаете ответ на этот вопрос. Тем не менее, рассмотрим идеи и принципы репликации.

Репликация — это создание копии или реплики всех изменений, внесенных в данные *главного (master) сервера*, на другом сервере, который называется *подчиненным (slave) сервером*. Обычно так создают полную копию главного сервера, но возможности репликации на этом не исчерпаны.

Чаще всего, репликацию используют для двух целей: создание резервной копии главного сервера — чтобы избежать потери данных в случае сбоя; и копирование главного сервера для генерации отчета и анализа работы без помех текущим операциям.

Это не только существенно упрощает жизнь малого бизнеса, но и открывает новые горизонты:

- *Поддержка филиалов.* Можно поддерживать серверы в удаленных филиалах, реплицируя изменения между ними. Это необходимо для защиты данных, а также для удовлетворения нормативных требований к доступности информации о бизнесе для проверки.
- *Обеспечение работы организации даже при выходе из строя одного из серверов.* В случае поломки одного сервера обработку трафика можно передать другому.
- *Продолжение работы организации в случае стихийного бедствия.* Репликация позволяет направлять сведения об изменениях в резервный центр обработки данных, удаленный от места катастрофы.
- *Защита от ошибок оператора.* Можно настроить репликацию на подчиненный сервер с задержкой. Для этого подчиненный сервер подключают к главному с отставанием на заданный промежуток времени, например, на один час. Если в команде на главном сервере обнаружится ошибка, ее можно будет устранить до передачи и исполнения на подчиненном сервере.

Одна из главных задач репликации в современных приложениях — *масштабирование*. Обычно современные приложения выполняют намного больше операций чтения, чем операций записи. Чтобы разгрузить главный сервер, настройте подчиненный сервер, единственным назначением которо-

го будет исполнение запросов чтения данных. Подключив подсистему балансировки нагрузки, можно будет направлять запросы чтения на соответствующий подчиненный сервер, а запросы записи — на главный.

Применяя репликацию для масштабирования, важно знать, что репликация MySQL является *асинхронной*. То есть, сначала транзакции фиксируются на главном сервере, а затем реплицируются на подчиненный сервер и там исполняются. Это означает, что при непрерывной репликации подчиненный сервер будет «отставать» от главного.

Асинхронная репликация превосходит синхронную по скорости и масштабируемости. Однако для своевременной репликации особо важных данных придется решать проблему асинхронной репликации.

Другая немаловажная сторона репликации — обеспечение высокой доступности путем создания избыточности данных. Наиболее распространенный вариант развертывания включает два главных сервера, то есть репликация обеспечивает постоянную доступность пары главных серверов, в которой каждый сервер является зеркальным отражением другого. При сбое одного из главных серверов второй сервер мгновенно берет на себя его функции.

Кроме развертывания пары главных серверов существуют и другие способы достижения высокой доступности без репликации, например, общие или реплицируемые диски. И хотя данные способы не имеют прямого отношения к MySQL, это важные средства обеспечения высокой доступности.

А можно ли обойтись без резервного копирования?

Стратегия резервного копирования — критическая составляющая доступности системы. Регулярное резервное копирование обеспечивает защиту от сбоя и аварии, что, до некоторой степени, можно достичь репликацией. Но даже если репликация проводится эффективно и правильно, некоторые вещи ей не под силу. Работоспособная стратегия резервного копирования пригодится для:

- *защиты от ошибок*. Репликация не поможет, если ошибка обнаружена слишком поздно. В этом случае, необходимо откатить систему до состояния, предшествующего возникновению ошибки, и устранить проблему. Для этого требуется эффективное расписание архивации.

  Репликация отчасти защищает от ошибок, если выполняется с задержкой. Но если ошибка будет обнаружена по истечении времени задержки, то ошибочные изменения вступят в силу и на подчиненном сервере. В общем, защитить систему от ошибок, надеясь только на репликацию, невозможно — нужны и резервные копии;
- *создания новых серверов*. Чтобы создать новые серверы — будь то подчиненные серверы для масштабирования или резервные главные — до-

статочно сделать резервную копию существующего сервера и восстановить архивный образ на новом сервере. При этом потребуется быстрый и эффективный способ архивации, чтобы сократить время простоя и поддерживать загруженность системы на приемлемом уровне;

- *обеспечения соответствия нормативным требованиям.* Кроме интересов бизнеса, сохранность данных, в том числе и при катастрофах, может быть продиктована требованиями закона. Несоответствие нормативным требованиям может повлечь серьезные проблемы для организации.

Словом, стратегия резервного копирования необходима для работы бизнеса, независимо от других мер, предпринимаемых для сохранности данных.

Для чего нужен мониторинг?

Даже при правильной настройке репликации следует следить за загруженностью системы и возникающими неполадками. Требования бизнеса меняются вслед за потребительским спросом, и приходится балансировать систему для наиболее эффективного использования ресурсов и снижения вероятности ее недоступности из-за внезапного роста числа пользовательских запросов.

Есть целый ряд параметров, мониторинг, оценка и планирование которых поможет вам пережить подобные скачки. Например:

- Добавление индексов к часто используемым таблицам.
- Оптимизация запросов и структуры БД, обеспечивающая минимальное время выполнения запросов.
- Слишком долгая блокировка указывает на то, что одну таблицу используют несколько подключений. Возможно, следует «развести» таблицы по разным серверам.
- Если после наращивания системы некоторые из подчиненных серверов обрабатывают слишком много или слишком мало запросов чтения, лучше перенастроить систему для обеспечения равномерной загруженности серверов.
- Чтобы преодолеть внезапный рост запросов ресурсов, необходимо определить нормальный уровень нагрузки для каждого сервера и знать, когда отклик системы замедляется из-за резкого повышения нагрузки.

Словом, без мониторинга вы не сможете найти проблемные запросы, перегруженные подчиненные серверы или таблицы, которые используются неправильно.

Дополнительные источники

Существует много литературы о применении MySQL для решения различных задач, а также о системах высокой доступности. Ниже приведен список книг, рекомендованных всем желающим использовать MySQL:

- *Paul DuBois. MySQL. (Addison-Wesley)* — справочник по MySQL аж на 1200 страницах! В нем вы найдете все, что вам нужно знать о MySQL (и даже больше).
- *Baron Schwartz et al. High Performance MySQL, 2nd Edition (O'Reilly, http://oreilly.com/catalog/9780596101718/)* — одна из лучших книг о работе с MySQL в корпоративной среде. В книге рассмотрена оптимизация запросов, а также обеспечение работоспособности и доступности системы.
- *Theo Schlossnagle. Scalable Internet Architectures (Sams Publishing)*. Книга написана одним из величайших теоретиков в отрасли, обязательна к прочтению для всех, кто работает с масштабируемыми системами.

В книге используется библиотека Python (под названием MySQL Python Replicant), разработанная авторами для решения многих административных задач. Загрузить библиотеку MySQL Python можно на Launchpad по адресу: https://launchpad.net/mysql-replicant-python.

Заключение

В следующей главе мы начнем с основ настройки репликации. Поэтому устраивайтесь поудобнее, запускайте свой компьютер и вперед...

Джоэл копался с регулировкой своего кресла, когда в дверь постучали.

— Устраиваешься, Джоэл? — спросил г-н Саммерсон.

Джоэл не знал, что сказать. В свой первый рабочий день он должен был настроить подчиненный сервер для репликации. Это заняло больше времени, чем он предполагал, поэтому он с трепетом ждал мнение начальства о проделанной работе. Джоэл сказал первое, что пришло в голову: «Да, сэр. Вот пытаюсь справиться с креслом».

— Джоэл, по работе с документацией — все хорошо. Хотелось бы, чтобы ты написал отчет и изложил свое мнение о мерах, которые нужно предпринять для совершенствования управления сервером базы данных.

Джоэл кивнул: «Будет сделано».

— Отлично. У тебя еще будет день на обустройство. Отчет нужен к среде, к концу рабочего дня.

Прежде чем Джоэл смог ответить, г-н Саммерсон удалился.

Джоэл уселся и повернул рычаг кресла. С характерным звуком спинка кресла откинулась, и он инстинктивно выбросил руки вперед: «Оп-па!» Кое-как вернул спинку кресла на место и посмотрел на дверь — хорошо, что никто не видел этой его гимнастики. «Так, этот рычаг лучше не трогать», — подумал он.

ГЛАВА 2

Основы репликации MySQL

Джоэл вздрогнул от резкого стука в дверь, возвещавшего о неожиданном приходе босса. Не успел Джоэл сказать «войдите», как начальство уже возникло в дверях.

— Джоэл, есть жалобы на то, что время отклика увеличилось. Проверь, что можно сделать, чтобы ускорить отклик. Администратор говорит, приложения выполняют слишком много операций чтения. Найди способ разгрузить сервер.

Джоэл не успел ничего ответить, как Саммерсон уже ушел по своим делам. «Может, он считает, что нам нужен сервер помощнее», — подумал Джоэл.

Словно прочитав его мысли, Саммерсон, стоя в дверях, обернулся:

— Да, кстати, мы покупали все оборудование в одном интернет-магазине. И у них там есть пара-тройка серверов — у нас еще таких нет. Посмотри, может они нам сгодятся? Хорошо?

И вышел из комнаты.

«Когда ж я привыкну?» — подумал Джоэл, доставая любимую книгу по MySQL с полки и глядя в оглавление. Он нашел главу о репликации, надеясь на лучшее.

Репликация MySQL, если ее грамотно применять, — инструмент очень полезный. В то же время, она может стать настоящей головной болью в случае сбоя или неправильной настройки. В этой главе мы рассмотрим основы репликации MySQL. Для начала возьмем простенькую систему, а затем перейдем к базовым методам, которые вы добавите в свой «арсенал».

В этой главе рассмотрены следующие сценарии использования:

- *Горячий резерв* Если «упадет» сервер, то остановится все: не будет возможности обрабатывать транзакции (может быть, критически важные), получать сведения о клиентах или извлекать необходимые данные. Подобного сценария нужно избежать (почти) любой ценой, так как это может серьезно подорвать бизнес. Самый простой способ: настроить дополнительный сервер, единственная цель которого — быть в горячем резерве, готовым взять на себя функции главного сервера, если тот выйдет из строя.
- *Создание отчетов* Создание отчетов на основе хранящихся на сервере данных снизит производительность сервера, в некоторых случаях, значительно. Если при создании отчетов выполняется много фоновых заданий,

то для этих целей лучше создать дополнительный сервер. В заданное время останавливается репликация, и на сервере отчетов создается снимок БД, после чего «громоздкие» запросы можно адресовать ему, а не главному серверу предприятия. К примеру, если вы остановите репликацию после завершения последней транзакции дня, то получите отчет за день, в то время как остальная деятельность предприятия будет идти в обычном ритме.

- *Отладка и аудит* Изучение выполненных на сервере запросов поможет вам разобраться в некоторых вопросах, например: не возникала ли нехватка производительности при выполнении отдельных запросов, имела ли место рассинхронизация сервера из-за ошибки в запросе.

Репликация — первые шаги

В этой главе представлены сложные приемы, позволяющие максимально повысить эффективность и выгоду от репликации, но начнем мы с простого примера (рис. 2-1) с одним экземпляром репликации от главного сервера к подчиненному. Для этого не требуются знания внутренней архитектуры или механизма выполнения репликации (об этом речь пойдет перед выполнением более сложных сценариев).

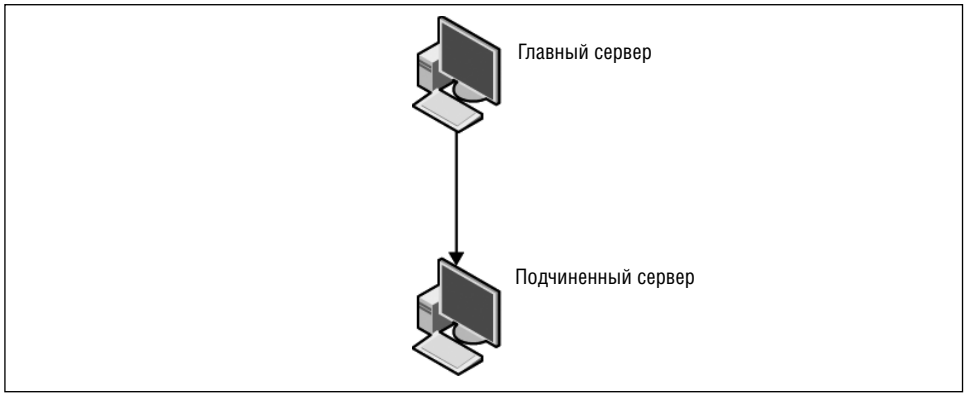


Рис. 2-1. Простая схема репликации

Настройка простой репликации выполняется в три шага:

1. Конфигурирование одного из серверов в качестве главного.
2. Конфигурирование одного из серверов в качестве подчиненного.
3. Подключение подчиненного сервера к главному.

Если репликация не планировалась с самого начала, и нужные параметры конфигурации не были включены в файлы *my.cnf*, то для того чтобы выполнить шаги 1 и 2, потребуется перезагрузить оба сервера.

Выполнять процедуру данного раздела проще всего с учетной записью shell на компьютере, имеющем достаточно полномочий для изменения файла *my.cnf*. Как правило, речь идет о полномочиях mysql или учетной записи на сервере, которая обладает всеми (ALL) полномочиями.

В производственной среде следует предоставлять минимум необходимых полномочий. Точные рекомендации см. в разделе «Полномочия, необходимые для настройки репликации».

Конфигурирование главного сервера

Перед настройкой сервера в качестве главного проверьте, что на сервере есть активный двоичный журнал (binary log) и уникальный идентификатор сервера. Подробнее двоичный журнал будет рассмотрен далее — сейчас необходимо усвоить, что в нем хранится запись обо всех изменениях, сделанных на главном сервере, позволяющая воспроизвести их на подчиненном сервере. Идентификатор сервера нужен, чтобы отличить один сервер от другого. Для того чтобы настроить двоичный журнал и идентификатор сервера, необходимо выключить сервер и добавить к файлу конфигурации my.cnf параметры log-bin, log-bin-index и server-id, как показано в листинге 2-1. Добавленные параметры выделены полужирным.

Лист. 2-1. Параметры, добавляемые к файлу my.cnf, для настройки главного сервера

```
[mysqld]
user              = mysql
pid-file          = /var/run/mysqld/mysqld.pid
socket            = /var/run/mysqld/mysqld.sock
port              = 3306
basedir           = /usr
datadir           = /var/lib/mysql
tmpdir            = /tmp
log-bin           = master-bin
log-bin-index     = master-bin.index
server-id         = 1
```

В параметре log-bin задано базовое имя для всех файлов, созданных в двоичном журнале (позднее вы увидите, что двоичный журнал состоит из нескольких файлов). Если в параметре log-bin имя файла задано с расширением, то расширение будет проигнорировано, а будет использовано только базовое имя файла (то есть, имя без расширения).

В параметре log-bin-index задано базовое имя индексного файла двоичного журнала, в котором содержится список всех файлов двоичного журнала.

По правде говоря, можно не указывать имя файла в параметре log-bin. Значение по умолчанию — hostname-bin. Значение hostname берется из параметра pid-file, по умолчанию — это имя узла (возвращаемое системным вызовом gethostname(2)). Если позднее администратор изменит имя узла, имена файлов двоичного журнала будут также изменены, а изменения за-

писаны в индексный файл. Нелишним будет создать имя, уникальное для сервера, но не привязанное к компьютеру, на котором он работает, так как при работе с несколькими файлами двоичного журнала, которые на полпути меняют свои имена, может возникнуть путаница.

Если значение параметра log-bin-index не определено, то значением по умолчанию будет базовое имя файлов двоичного журнала (hostname-bin, если не задано стандартное значение в параметре log-bin). Это значит, что если вы не задали значение параметра log-bin-index, то после изменения имени узла, имя индексного файла будет изменено обязательно. Получается, что если вы измените имя узла и запустите сервер, то сервер не найдет индексный файл и будет думать, что он отсутствует. Таким образом, у вас появится пустой журнал.

У каждого сервера есть уникальный идентификатор сервера. Поэтому если подчиненный сервер подключить к главному с идентичным значением параметра server-id, будет создана ошибка, указывающая на то, что у главного и подчиненного серверов совпадают идентификаторы сервера.

После того, как вы добавили параметры log-bin и server-id в файл конфигурации, снова запустите сервер и завершите настройку, добавив пользователя репликации.

Внеся изменения в файл конфигурации главного сервера, перезапустите главный сервер, чтобы изменения вступили в силу.

Подчиненный сервер инициирует обычное подключение клиента к главному серверу и посылает запрос на главный сервер на получение всех изменений. Для того чтобы подключить подчиненный сервер, на главном сервере должен быть пользователь с особыми полномочиями репликации. В лист. 2-2 представлен обычный сеанс клиента mysql на главном сервере и команды, позволяющие добавить нового пользователя и предоставить ему соответствующие полномочия.

Лист. 2-2. Создание пользователя репликации на главном сервере

```
master> CREATE USER repl_user;
Query OK, 0 rows affected (0.00 sec)
master> GRANT REPLICATION SLAVE ON *.*
    -> TO repl_user IDENTIFIED BY 'xyzzy';
Query OK, 0 rows affected (0.00 sec)
```

В полномочии REPLICATION SLAVE нет ничего необычного, за исключением того, что пользователь может получать дамп двоичного журнала от главного сервера. Нет ничего предосудительного в том, чтобы наделить полномочием REPLICATION SLAVE обычную учетную запись пользователя. Тем не менее, пользователя с подобным полномочием лучше хранить отдельно от других пользователей. Если в дальнейшем потребуется запретить подключение для некоторых подчиненных серверов, можно будет просто удалить учетную запись.

Конфигурирование подчиненного сервера

Настроив главный сервер, перейдем к настройке подчиненного. Как и главному, подчиненному серверу нужно присвоить уникальный идентификатор сервера. Кроме того, в файл my.cnf можно добавить имена файлов журнала ретрансляции и индекса журнала ретрансляции (подробнее о журнале ретрансляции см. в главе 5) при помощи параметров relay-log и relay-log-index. Рекомендуемые параметры конфигурации содержатся в листинге 2-3. Добавляемые параметры выделены.

Лист. 2-3. Параметры, добавляемые к файлу my.cnf для настройки подчиненного сервера

```
[mysqld]
user              = mysql
pid-file          = /var/run/mysqld/mysqld.pid
socket            = /var/run/mysqld/mysqld.sock
port              = 3306
basedir           = /usr
datadir           = /var/lib/mysql
tmpdir            = /tmp
server-id         = 2
relay-log-index   = slave-relay-bin.index
relay-log         = slave-relay-bin
```

По умолчанию значения параметров relay-log и relay-log-index, как и для параметров log-bin и log-bin-index, зависят от имени узла. Стандартное значение для relay-log — hostnamerelay-bin, а для relay-log-index — hostname-relay-bin.index. Использование умолчаний чревато проблемой, так как в случае изменения имени узла сервера индексный файл журнала ретрансляции не будет найден, и сервер будет полагать, что файлы журнала ретрансляции пустые.

После изменения файла my.cnf, перезапустите подчиненный сервер, чтобы изменения вступили в силу.

Подключение главного и подчиненного серверов

Вам осталось сделать заключительный шаг в настройке базовой репликации: настроить подчиненный сервер на главный, чтобы он знал, откуда выполнять репликацию. Для этого нужно ответить на четыре вопроса о главном сервере:

- Имя узла?
- Номер порта?
- Учетная запись на главном сервере, обладающая необходимыми полномочиями для репликации на подчиненный сервер?
- Пароль для этой учетной записи?

В ходе настройки главного сервера вы уже создали учетную запись с соответствующими полномочиями и паролем. Имя узла присвоено операционной

системой и не может быть изменено в файле my.cnf, но в этом файле можно задать номер порта (если номер порта не задан, будет использовано стандартное значение — 3306). Два заключительных шага для начала репликации: нацелить подчиненный сервер на главный при помощи команды CHANGE MASTER TO, а затем запустить репликацию командой START SLAVE.

```
slave> CHANGE MASTER TO
    ->  MASTER_HOST = 'master-1',
    ->  MASTER_PORT = 3306,
    ->  MASTER_USER = 'repl_user',
    ->  MASTER_PASSWORD = 'xyzzy';
Query OK, 0 rows affected (0.00 sec)

slave> START SLAVE;
Query OK, 0 rows affected (0.15 sec)
```

Поздравляем! Простейшая репликация между главным и подчиненным серверами настроена. Если изменить БД на главном сервере — например, добавить или заполнить таблицы — изменения будут реплицированы на подчиненный сервер. Попробуйте сами, создайте тестовую БД (если еще не создали), таблицы, добавьте в них данные, и посмотрите на работу репликации.

Обратите внимание, что в параметре MASTER_HOST можно указать либо имя узла, либо IP-адрес. Если задано имя узла, IP-адрес узла будет извлечен вызовом gethostname(3). В зависимости от конфигурации, это может означать выполнение разрешения имени узла путем просмотра DNS. Шаги по настройке самого просмотра выходят за рамки этой книги.

Полномочия, необходимые для настройки репликации

Чтобы подключить подчиненный сервер к главному для репликации, кроме учетной записи shell с правом доступа к критическим файлам, необходимо иметь учетную запись с определенными полномочиями. По соображениям безопасности, учетную запись, используемую для настройки главного и подчиненного серверов лучше наделить только самыми необходимыми полномочиями.

- Для создания и удаления пользователей учетная запись должна обладать полномочием CREATE USER.
- Чтобы предоставить учетной записи репликации полномочие REPLICATION SLAVE, необходимо иметь полномочие REPLICATION SLAVE с параметром GRANT OPTION.

Для выполнения других процедур, связанных с репликацией (см. далее), вам понадобятся дополнительные опции:

- Для выполнения команды FLUSH LOGS (как и любой команды FLUSH) требуется полномочие RELOAD.
- Для выполнения команд SHOW MASTER STATUS и SHOW SLAVE STATUS, потребуется полномочие SUPER или REPLICATION CLIENT.
- Чтобы выполнить команду CHANGE MASTER TO, необходимо полномочие SUPER.

К примеру, чтобы предоставить пользователю *mats* достаточные полномочия для выполнения всех процедур, описанных в этой главе, выполните следующее:

```
server >    GRANT REPLICATION SLAVE, RELOAD, CREATE USER, SUPER
       ->   ON *.*
       ->   TO mats@'192.168.2.%'
       ->   WITH GRANT OPTION;
```

Кое-что о двоичном журнале

Движущей силой репликации является *двоичный журнал* (*binary log* или просто *binlog*). Это запись всех изменений, внесенных в БД или сервер. Для того чтоб управлять репликацией или устранять возникающие неполадки, нужно понимать, как работает двоичный журнал. В данном разделе речь пойдет именно об этом.

На рис. 2-2 схематически показана архитектура репликации: главный сервер с двоичным журналом и подчиненный, получающий изменения от главного посредством двоичного журнала. Подробнее об архитектуре репликации читайте в главе 6. В конце выполнения инструкции в конец двоичного файла добавляется запись, а синтаксический анализатор получает уведомление о том, что инструкция выполнена. Как правило, в двоичный журнал записываются только инструкции, выполнение которых подходит к концу, однако в отдельных случаях, записывается другая информация: либо в дополнение, либо вместо инструкции. Скоро вы узнаете, почему так происходит, но до поры давайте считать, что в двоичный журнал записываются только выполняемые инструкции.

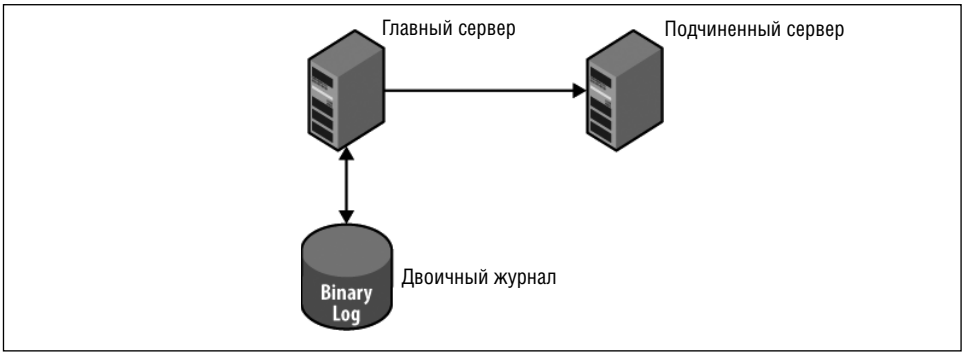


Рис. 2-2. Роль двоичного журнала в репликации

Что записано в двоичном журнале

Назначение двоичного журнала — запись изменений, внесенных в таблицы БД. Двоичный журнал применяется для репликации, а также для восстанов-

ления на определенный момент времени (см. главу 12) и в некоторых, очень редких случаях, для аудита.

Отметим, что в двоичном журнале содержатся только внесенные в БД изменения, так что записи инструкций, выполнение которых не привело к изменению данных в БД, в двоичный журнал не добавляются.

Традиционно в ходе репликации MySQL запись изменений выполняется посредством сохранения инструкции SQL, приведшей к изменению. Это называется логической репликацией или репликацией на основе команд (statement-based replication). Логическая репликация имеет ограничения, препятствующие корректному реплицированию всех инструкций. Поэтому, начиная с версии MySQL 5.1, была предложена построчная репликация (row-based replication). В отличие от логической репликации, в ходе построчной репликации каждое изменение в отдельности записывается в строку двоичного журнала. Кроме дополнительного удобства, построчная репликация в иных ситуациях дает выигрыш в скорости.

Чтобы почувствовать разницу, представьте себе сложное обновление с использованием множества соединений и предложений, начинающихся с WHERE. Вместо того чтобы выполнять всю логическую часть на подчиненном сервере повторно, как это следует при логической репликации, все, что вам нужно будет знать — это состояние строк после изменения. С другой стороны, если за одно обновление изменено 10000 строк, проще записать инструкцию, а не 10000 отдельных изменений, как это делается в построчной репликации.

Построчная репликация, ее применение и реализация рассмотрены в главе 6. Следующие примеры построены на основе логической репликации — так проще понять действия, выполняемые над БД.

Работа репликации

На примере репликации из предыдущего раздела рассмотрим события двоичного журнала, характерные для простых инструкций. Начнем с подключения клиента с командным интерфейсом к главному серверу и выполним пару команд, чтобы извлечь двоичный журнал:

```
master> CREATE TABLE tbl (text TEXT);
Query OK, 0 rows affected (0.04 sec)

master> INSERT INTO tbl VALUES ("Yeah! Replication!");
Query OK, 1 row affected (0.00 sec)

master> SELECT * FROM tbl;
+ --------------------  +
| text                  |
+ --------------------  +
| Yeah! Replication!    |
+ --------------------  +
1 row in set (0.00 sec)
```

```
master> FLUSH LOGS;
Query OK, 0 rows affected (0.28 sec)
```

Команда FLUSH LOGS принудительно выполняет ротацию двоичного журнала. Это позволит нам взглянуть на файл журнала «целиком», как он есть. Рассмотреть файл журнала поближе нам поможет команда SHOW BINLOG EVENTS (см. лист. 2-4).

Лист. 2-4. Просмотр событий, содержащихся в двоичном журнале

```
master> SHOW BINLOG EVENTS\G
*************************** 1. row ***************************
   Log_name: master-bin.000001
        Pos: 4
 Event_type: Format_desc
  Server_id: 1
End_log_pos: 106
       Info: Server ver: 5.1.33, Binlog ver: 4
*************************** 2. row ***************************
   Log_name: master-bin.000001
        Pos: 106
 Event_type: Query
  Server_id: 1
End_log_pos: 197
       Info: use `test`; CREATE TABLE tbl (text TEXT)
*************************** 3. row ***************************
   Log_name: master-bin.000001
        Pos: 197
 Event_type: Query
  Server_id: 1
End_log_pos: 305
       Info: use `test`; INSERT INTO tbl VALUES (“Yeah! Replication!”)
*************************** 4. row ***************************
   Log_name: master-bin.000001
        Pos: 305
 Event_type: Rotate
  Server_id: 1
End_log_pos: 349
       Info: master-bin.000002;pos=4 4 rows in set (0.02 sec)
```

В данном двоичном журнале мы видим четыре события: событие format description (описание формата), два события query (запрос) и событие rotate (ротация). Событие query характеризует выполнение инструкций над БД, обычно они записываются в двоичный журнал, тогда как события format description и rotate используются внутри сервера для управления журналом. Подробнее эти события мы обсудим в главе 6, но сейчас рассмотрим внимательнее столбцы, соответствующие каждому событию.

Event_type Это тип события. Здесь мы видим только три типа события, но их гораздо больше. Тип события — это основной способ передачи информации на подчиненный компьютер. На сегодняшний день, в версиях MySQL 5.1.18 — 5.1.39 насчитывается 27 типов событий (некоторые из них не используются и присутствуют для обеспечения обратной совместимости). Число это не окончательно, и, если потребуется, в следующих версиях будут добавлены новые события.

Server_id Идентификатор сервера, создавшего событие.

Log_name Имя файла, в котором хранится событие. Событие всегда хранится в одном файле и не делится.

Pos Позиция начала события в файле: место, где записан первый байт события.

End_log_pos Позиция в файле, где событие завершается и начинается следующее. Значение это на один байт выше, чем значение позиции последнего байта события, поэтому в диапазоне от Pos до End_log_pos – 1 находятся байты, в которых содержится событие. Длина события характеризуется разностью End_log_pos – Pos.

Info Это понятный для человека текст с информацией о событии. Для различных событий выводятся различные сведения, но для события запрос, по крайней мере, выводится содержащаяся в запросе инструкция.

Два первых столбца, Log_name и Pos, составляют позицию события в двоичном журнале (*binlog position*), и применяются для обозначения расположения или позиции события. Кроме сведений, рассмотренных нами, в каждом событии содержится много другой информации, в частности, временная метка, представляющая собой количество секунд с начала эпохи (классический пример временной метки Unix: 1970-01-01 00:00:00 UTC).

Структура и содержимое двоичного журнала

Как уже отмечалось, двоичный журнал это, на самом деле, не один файл, а набор файлов. Так проще управлять журналом (например, можно удалить старые журналы, не трогая последние). В состав двоичного журнала входит набор файлов журнала с реальным содержимым, а также индексный файл двоичного журнала, в котором есть сведения о существующих файлах журнала. Организация двоичного журнала показана на рис. 2-3.

Один из файлов двоичного журнала является *активным (active binlog file)*. Это файл, в который запись выполняется в данный момент (и откуда, как правило, выполняется чтение).

Каждый файл двоичного журнала начинается с события format description, а завершается событием rotate. В событии format description, кроме всего прочего, содержится версия сервера, на котором был создан файл и общие сведения о сервере и двоичном журнале.

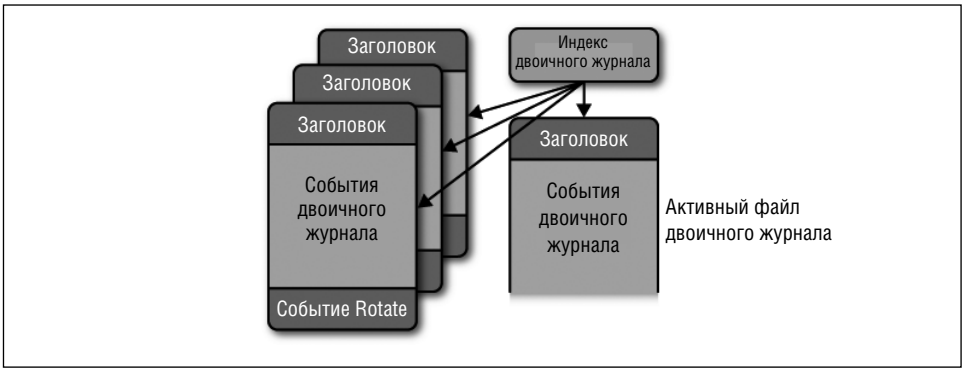


Рис. 2-3. Структура двоичного журнала

В событии rotate указано место продолжения двоичного журнала — имя следующего файла в последовательности.

Каждый файл сформирован событиями двоичного журнала, а каждое событие — это отдельная структурная частица журнала. В событии двоичного журнала format description есть флаг, указывающий на то, что файл полностью закрыт. Флаг установлен до тех пор, пока в файл журнала выполняется запись — флаг снимается во время закрытия файла. Таким образом можно отыскать поврежденные файлы журнала и восстановить репликацию в случае сбоя.

Попытайтесь выполнить на главном сервере еще одну инструкцию, и вы увидите, что изменения никак не отражены в двоичном журнале.

```
master> INSERT INTO tbl VALUES («What's up?»);
Query OK, 1 row affected (0.00 sec)

master> SELECT * FROM tbl;
+ --------------------  +
| text                  |
+ --------------------  +
| Yeah! Replication!    |
| What's up?            |
+ --------------------  +
row in set (0.00 sec)

master> SHOW BINLOG EVENTS\G
same as before
```

Куда пропало последнее событие? Итак, нам известно, что двоичный журнал состоит из нескольких файлов, а инструкция SHOW BINLOG EVENTS отображает содержимое только первого файла двоичного журнала. Для многих пользователей, намеревающихся просмотреть содержимое активного файла двоичного журнала, это будет неожиданностью. Если имя первого файла двоичного журнала (в котором содержатся ранее показанные события) *master-bin.000001*, просмотрите события в следующем файле журнала — в нашем случае, это файл *master-bin.000002* — выполнив следующее:

```
master> SHOW BINLOG EVENTS IN 'master-bin.000002'\G
*************************** 1. row ***************************
   Log_name: master-bin.000002
        Pos: 4
 Event_type: Format_desc
  Server_id: 1
End_log_pos: 106
       Info: Server ver: 5.1.30-log, Binlog ver: 4
*************************** 2. row ***************************
   Log_name: master-bin.000002
        Pos: 106
 Event_type: Query
  Server_id: 1
End_log_pos: 205
       Info: use `test`; INSERT INTO tbl VALUES("What's up?")
2  rows in set (0.00 sec)
```

Вы, конечно, обратили внимание, на то, что в лист. 2-4 двоичный журнал завершается событием rotate, а в поле Info содержится имя следующего файла журнала и позиция начала событий. Узнать, в какой файл двоичного журнала выполняется запись в данный момент, позволяет команда SHOW MASTER STATUS.

```
master> SHOW MASTER STATUS\G
*************************** 1. row ***************************
            File: master-bin.000002
        Position: 205
    Binlog_Do_DB:
Binlog_Ignore_DB:
1 row in set (0.00 sec)
```

Теперь, просмотрев двоичный журнал, остановите и выполните сброс подчиненного сервера, и удалите таблицу:

```
master> DROP TABLE tbl;
Query OK, 0 rows affected (0.00 sec)

slave> STOP SLAVE;
Query OK, 0 rows affected (0.08 sec)

slave> RESET SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

После этого таблицу можно удалить и произвести повторный запуск репликации главного сервера:

```
master> DROP TABLE tbl;
Query OK, 0 rows affected (0.00 sec)

master> RESET MASTER;
Query OK, 0 rows affected (0.04 sec)
```

Команда RESET MASTER удаляет все файлы двоичного журнала и выполняет очистку индексного файла двоичного журнала. Инструкция RESET SLAVE удаляет все файлы, используемые на подчиненном сервере для репликации, для того чтобы начать все сначала.

Если репликация активна, команды RESET MASTER и RESET SLAVE запускать нельзя. Поэтому:

- Перед выполнением команды RESET MASTER (на главном сервере) убедитесь в том, что к нему не присоединены подчиненные серверы.
- Перед выполнением команды RESET SLAVE (на подчиненном сервере) убедитесь в том, что на подчиненном сервере репликация остановлена, выполнив команду STOP SLAVE.

В данной главе рассмотрены самые основные события, а полный подробный список, вы найдете в документе MySQL Internals Manual.

Управление репликацией средствами Python

Возможность автоматизации административных процедур крайне важна для обслуживания крупных развертываний, и вы, вероятно, задумывались над тем, можно ли автоматизировать все эти процедуры. Ответ — да, можно. Сейчас, используя описания из предыдущих разделов, мы начнем разрабатывать простую библиотеку для управления репликацией. Функциональность нашей библиотеки мы расширим в следующих главах.

Проект доступен на Launchpad по адресу: *http://launchpad.net/mysql-replicant-python*. Здесь вы найдете сведения о проекте и сможете загрузить исходный код и документацию.

Сначала, необходимо создать модель подключения серверов посредством репликации. Существует множество способов подключения большого числа серверов, но подключать их нужно в определенной конфигурации, которая называется *топологией (topology)*. Топология рассмотрена в главе 5, но базовые примеры топологии весьма просты (рис. 2-4): древовидная топология и топология с двумя главными серверами (применяемая для обеспечения высокой доступности).

Идея состоит в том, чтобы создать модель подключения серверов на компьютере (любом компьютере, скажем, на вашем ноутбуке), как показано на рис. 2-4, и разработать библиотеку, которая позволит управлять подключениями путем изменения модели. К примеру, чтобы подключить подчиненный сервер к другому главному серверу, достаточно изменить подключение подчиненного сервера в модели, и библиотека пошлет соответствующие команды на выполнение задачи.

Для того чтобы библиотека могла работать на различных платформах и различных развертываниях, необходимо продумать следующее:

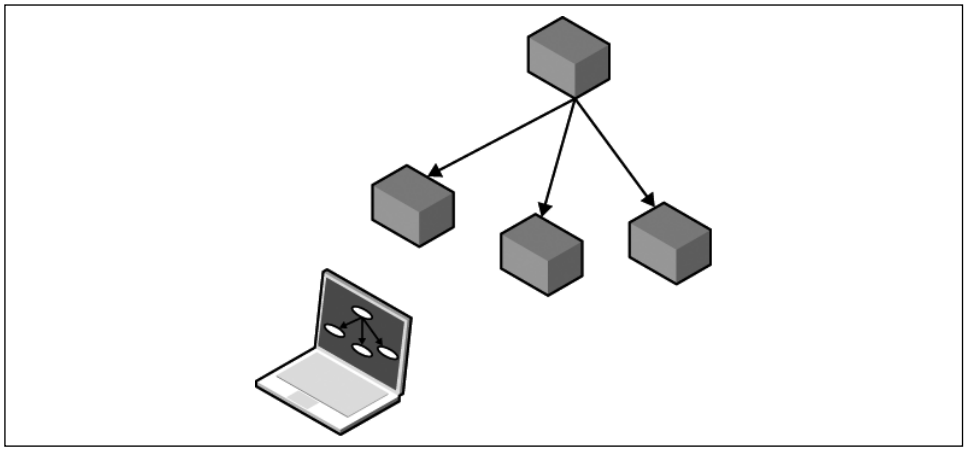


Рис. 2-4. Модель топологии репликации

- Серверы работают под управлением различных операционных систем: Windows, Linux и разновидности Unix, например, Solaris или Mac OS X. В зависимости от ОС, процедуры пуска и останова серверов, а также имена файлов конфигурации отличаются. Поэтому библиотека должна поддерживать различные ОС и быть расширяемой для других ОС, которых нет в библиотеке.
- Развертывание, может состоять из серверов под управлением различных версий MySQL. Например, в ходе обновления развертывания до новой версии сервера, в развертывании будут совместно работать новые и старые версии серверов. Библиотека должна уметь управлять подобным развертыванием.
- Развертывание состоит из серверов с множеством ролей, поэтому должна быть возможность назначать серверам различные роли. В дополнение, нужна возможность создавать новые роли, которых не было в начале.
- Необходима возможность выполнять SQL-запросы на каждом сервере. Это необходимо для конфигурирования, а также для извлечения информации, требующейся для управления развертыванием.
- Необходимо на каждом компьютере выполнять команды оболочки. Это нужно для решения некоторых административных задач, которые невозможно выполнить с помощью интерфейса SQL.
- Должна быть возможность добавлять и удалять параметры в файл конфигурации сервера.
- Библиотека должна поддерживать развертывание с несколькими серверами на компьютере. Для этого требуется распознавать различные файлы конфигурации и файлы БД, используемые различными MySQL-серверами на одном компьютере.
- Следует включить набор утилит, предназначенных для выполнения общих задач, таких как настройка репликации, и, в то же время, библиотека

должна быть расширяемая для новых служебных функций, которых не было в начале.

Все эти сложности, насколько возможно, скрыты за нехитрым интерфейсом, написанным на Python. Авторы предпочли Python из-за лаконичности этого языка, его легко читать, он есть во всех ОС, на которых работает MySQL, и популярность его для написания сценариев общего назначения растет. В лист. 2-5 содержится пример использования библиотеки для перенаправления всех подчиненных серверов на новый главный сервер.

Обратите внимание, что данный код — это всего лишь образец использования библиотеки. В этом примере, выполняется мгновенный останов репликации, что может повлечь потерю транзакций, выполняемых на активном сервере. О том, как правильно сменить главный сервер, читайте в главе 4.

Лист. 2-5. Перенаправление подчиненных серверов с использованием библиотеки

```
import MyDeployment

for slave in MyDeployment.slaves:
slave.stop()
change_master(slave, MyDeployment.master[1])
slave.start()
```

С кодом, благодаря которому такое возможно, вы познакомитесь в следующих разделах. Чтобы облегчить восприятие кода, мы опустили некоторые процедуры, связанные с проверкой ошибок и другие защитные меры, необходимые для стабильности и безопасности библиотеки. Полная версия кода библиотеки находится по адресу: *http://launchpad.net/mysql-replicant-python.*

Основные классы и функции

Первое, что нужно для работы с библиотекой — это базовые определения часто используемых параметров.

Первые классы — это исключения, используемые функциями в библиотеке.

Error Базовый класс для всех исключений в библиотеке.

EmptyRowError Исключение возникает при попытке выбрать поле в запросе, не вернувшем ни одной строки.

NoOptionError Исключение возникает, если компонент ConfigManager не находит параметр.

SlaveNotRunningError Исключение создается, если подчиненный сервер, который должен работать, выключен.

NotMasterError Исключение создается, если сервер не является главным, вследствие чего действие недопустимо.

NotSlaveError Исключение создается, если сервер не является подчиненным, вследствие чего действие недопустимо.

Position Класс позиционирования в двоичном журнале. Состоит из имени файла и величины байтового смещения внутри файла. Методом представления выводится поддающееся анализу представление о позициях в журнале (для хранения их во вторичном хранилище или просмотра).

Чтобы сравнить или запросить позицию, класс определяет оператор сравнения, позволяющий библиотеке привести позиции в двоичном журнале в порядок. Обратите внимание, что позиции на разных серверах могут отличаться друг от друга, поэтому нет смысла сравнивать позиции на разных серверах.

User Данный класс представляет пользователя с именем и паролем. Используется для многих типов учетных записей: учетная запись пользователя MySQL, учетная запись пользователя shell и учетная запись пользователя репликации (см. далее).

Операционная система

Для работы с разными ОС к вашим услугам набор классов, абстрагирующих различия между ними. Суть в том, чтобы снабдить каждый класс методами для решения необходимых задач, которые в каждой ОС реализованы посвоему. Все, что нам пока нужно, это методы останова и запуска сервера.

Machine Это — базовый класс для компьютера. В нем содержится вся информация, общая для данного типа компьютера. В экземпляре *machine* следует ожидать наличие, по меньшей мере, следующих членов:

`Machine.defaults_file`

Стандартное расположение файла *my.cnf* на компьютере.

`Machine.start_server(`*сервер*`)`

Метод запуска сервера.

`Machine.stop_server(`*сервер*`)`

Метод останова сервера.

Linux Класс для управления сервером, работающем на компьютере Linux. В данном классе для запуска и останова сервера используются сценарии init(8), хранящиеся в каталоге */etc/init.d*.

Solaris Класс для управления сервером, работающем на компьютере Solaris. Для запуска и останова сервера используется команда svadm(1M).

Класс сервера

В классе Server определены простейшие функции, которыми реализованы высокоуровневые функции, применяющиеся в интерфейсе.

Server.Server(имя, ...) Класс Server представляет сервер в системе. Во всей системе каждому работающему серверу соответствует один объект. Здесь описаны только наиболее значимые параметры, полный список которых вы найдете в проекте на Launchpad.

name Имя сервера, применяется для создания значений параметров pid-file, log-bin и log-bin-index. Если значение не задано, то оно будет взято из параметра pid-file, параметра log-bin и параметра log-bin-index, а в крайнем случае, буде использовано стандартное значение.

host, port и socket Узел, на котором расположен сервер; порт подключения к серверу в качестве MySQL-клиента; и socket подключения (для этого же узла).

ssh_user Комбинация пользователя и пароля для подключения к компьютеру, на котором работает сервер. Применяется для выполнения административных команд, таких как запуск и останов сервера или чтение и запись в файл конфигурации.

sql_user Комбинация пользователя и пароля для подключения к серверу с учетной записью MySQL при выполнении SQL-команд.

machine Объект, в котором содержатся примитивы, относящиеся к ОС. Мы выбираем имя, чтобы избежать конфликта имен, при помощи стандартного модуля os библиотеки. Данный параметр позволяет с помощью различных методов запускать и останавливать сервер, выполнять другие задания, а также изменять параметры, относящиеся к ОС. Параметры будут описаны позднее.

server_id Дополнительный параметр для хранения ID сервера, определенного в файле конфигурации каждого сервера. Если параметр опущен, идентификатор сервера считывается из файла конфигурации сервера. Если и в файле конфигурации нет ID сервера, то это — несвязанный сервер, не участвующий в репликации, ни как главный, ни как подчиненный сервер.

config_manager Дополнительный параметр для хранения ссылки на диспетчер конфигурации, на который отправляются запросы о конфигурации сервера.

Server.connect() *и Server.disconnect*() Методы connect и disconnect предназначены для создания подключения к серверу в сеансе перед началом выполнения команд и для отключения от сервера по завершении сеанса соответственно.

Важность методов в том, что иногда приходится оставлять подключение к серверу открытым даже после выполнения SQL-команд. В противном случае, во время выполнения, например, команды FLUSH TABLES WITH READ LOCK, блокировка автоматически снимается после разрыва подключения.

Server.ssh(*команда*) *и Server.sql*(*команда, аргументы*) Используется для выполнения на сервере команды оболочки или SQL-команды.

Оба метода — ssh и sql — возвращают итерируемые строки. Метод ssh возвращает список строк вывода выполненной команды, а sql возвращает список объектов внутреннего класса Row. Класс Row определяет методы __iter__ и next, и итерация выполняется над возвращаемыми строками, например:

```
for row in server.sql("SHOW DATABASES"):
print row["Database"]
```

Для обработки инструкций, возвращающих одну строку, в классе определен метод __getitem__, который выбирает поле из одной строки или создает исключение, если строка отсутствует. И если известно, что в возвращаемом значении есть только одна строка (что характерно для многих SQL-команд), то вы можете уйти от цикла, показанного в предыдущем примере и выполнить подобную команду:

```
print server.sql("SHOW MASTER STATUS")["Position"]
```

Server.fetch_config() и Server.replace_config()

Методы fetch_config и replace_config производят выборку файла конфигурации из удаленного сервера в память, чтобы пользователь мог добавить или удалить параметры или изменить значения некоторых параметров. Пример модуля, позволяющего добавить значения в параметры log-bin и log-bin-index:

```
config = master.fetch_config()
config.set('log-bin', 'capulet-bin')
config.set('log-bin-index', 'capulet-bin.index')
master.replace_config(config)
```

Server.start() *и Server.stop*() Для выполнения своей миссии, которая зависит от ОС сервера, методы start и stop пересылают информацию на объект machine. Методы выполняют либо запуск, либо останов сервера соответственно.

Роли сервера

В зависимости от своих ролей, серверы работают по-разному. В частности, главным серверам при подключении подчиненных серверов необходим пользователь репликации, однако подчиненные серверы не нуждаются в этой учетной записи, если они не являются главными серверами, к которым подключены подчиненные. Для выполнения гибкого захвата конфигурации серверов в различных ролях существуют классы.

При использовании на сервере метода imbue, на сервер посылаются соответствующие данной роли команды конфигурирования. На протяжении жизненного цикла развертывания роли сервера могут меняться, поэтому мы приводим роли, служащие лишь для начального развертывания. Тем не менее, в развертывании сервер всегда выполняет определенную для него функцию, что связано с определенной ролью.

Во время изменения роли сервера, из сервера следует удалить некоторые сведения о конфигурации, поэтому, при изменении роли сервера, используется метод unimbue.

В данном примере определены только три роли. Далее в книге вы познакомитесь и с другими ролями.

Role Это базовый для всех ролей класс. Для каждого производного класса должны быть определены методы imbue и (необязательно) unimbue для установки роли на один сервер. Для выполнения производным классом некоторых общих задач, в классе Role определены некоторые вспомогательные функции.

Role.imbue(*сервер*) Данный метод предназначен для установки на сервер новой роли путем выполнения соответствующего кода.

Role.unimbue(сервер) Данный метод позволяет произвести очистку перед установкой другой роли.

Role._set_server_id(*сервер, файл конфигурации*) Если в конфигурации отсутствует идентификатор сервера, данный метод устанавливает значение server.server_id. Если в конфигурации есть ИД сервера, то его значение передается в функцию server.server_id.

Role._create_repl_user(*сервер, пользователь*) Данный метод создает пользователя репликации на сервере, наделяя его полномочиями, необходимыми подчиненному серверу репликации.

Role._enable_binlog(*сервер, файл конфигурации*) Метод включает на сервере двоичный журнал, присваивая параметрам log-bin и log-bin-index соответствующие значения. Если на сервере параметру log-bin уже присвоено значение, ничего не будет выполнено.

Role._disable_binlog(*сервер, файл конфигурации*) Метод отключает двоичный журнал путем удаления параметров log-bin и log-binindex из двоичного журнала.

Vagabond (*Несвязанный сервер*) Стандартная роль, назначаемая любому серверу, не принимающему участие в развертывании репликации. По существу, такой сервер свободен и не отвечает ни за что.

Master (*Главный сервер*) Роль сервера, выступающего в качестве главного. С данной ролью будет установлен ID сервера, включен двоичный журнал и создан пользователь репликации для подчиненных серверов. Имя и пароль для пользователя репликации будет сохранено на сервере таким образом, что при подключении подчиненных серверов, класс сможет отыскать имя пользователя репликации.

Final (*Тупиковый сервер*) Роль последнего подчиненного сервера, то есть сервера, не имеющего двоичного журнала. С установкой роли на сервер ему будет присвоен ID сервера, будет отключен двоичный журнал, и будет выполнена команда CHANGE MASTER для подключения подчиненного сервера к главному.

Обратите внимание, что для записи на сервер файла конфигурации его нужно остановить, а после записи файла сервер нужно перезапустить. Во время запуска сервера выполняется только чтение файла конфигурации, после выполнения чтения файл закрывается, но мы не будем рисковать и остановим сервер перед изменением файла.

Одно из критических проектных решений здесь — это не сохранять информацию о состоянии серверов, к которым применяется роль. Может показаться привлекательным сохранить список главных серверов, добавив их к объектам роли. Однако роли у серверов могут изменяться в течение жизненного цикла развертывания, поэтому роли применяются только для настройки системы. По причине того, что мы можем хранить в роли параметры, роли можно использовать для настройки нескольких серверов единообразно.

```
slave_role = Final(master=MyDeployment.master)
for slave in MyDeployment.slaves:
    slave_role.imbue(slave)
```

Создание новых подчиненных серверов

Теперь, когда мы уже имеем представление о двоичном журнале, все готово для решения одной из основных проблем способом, которым мы ранее создали подчиненный сервер. Во время настройки подчиненного сервера мы не задали место, откуда следует начинать репликацию, — и подчиненный сервер начнет чтение двоичных журналов на главном от начала. А это не годится, если главный сервер уже какое-то время работает: помимо того, что сервер будет перегружен воспроизведением множества событий, не факт, что вы сможете получить необходимые журналы. Дело в том, что они могут быть удалены с главного сервера и храниться для безопасности в другом месте (подробнее об этом, а также о резервном копировании и восстановлении на определенный момент времени см. главу 12). Поэтому нам нужен другой способ создания новых подчиненных серверов под названием *начальная загрузка подчиненного сервера (bootstrapping a slave)*, при котором нет необходимости начинать репликацию от начала.

Команда CHANGE MASTER TO имеет два полезных параметра: MASTER_LOG_FILE и MASTER_LOG_POS. С их помощью можно указать позицию в двоичном журнале, с которой главный сервер должен будет начать отправлять события, чтобы не отправлять события с самого начала.

Данные параметры команды CHANGE MASTER TO помогут нам выполнить начальную загрузку подчиненного сервера следующим образом:

1. Настроить новый подчиненный сервер.
2. Создать резервную копию главного сервера (или подчиненного сервера, реплицирующего главный). Распространенные способы резервного копирования представлены в главе 12.
3. Записать позицию двоичного журнала, которая соответствует данной резервной копии (другими словами, позиция, следующая за последним событием, в сторону текущего состояния главного сервера).
4. Восстановить резервную копию на новом подчиненном сервере. Распространенные способы восстановления представлены в главе 12.
5. Настроить подчиненный сервер на запуск репликации с этой позиции.

В зависимости от того, какой сервер вы будете архивировать в шаге 2 — главный или подчиненный, — в процедуре будут небольшие различия. Мы начнем с описания начальной загрузки нового подчиненного сервера, когда работает всего один главный сервер. Это называется *копирование (cloning)* главного сервера.

Копирование главного сервера — это создание снимка состояния сервера, что, как правило, выполняется при архивации. Существуют различные способы архивации сервера, но в этой главе мы воспользуемся одним из самых простых: создадим логическую резервную копию, выполнив команду mysql dump. Среди прочих способов — создание физической резервной копии путем копирования файлов БД, оперативная архивация с помощью таких средств как InnoDB Hot Backup или даже создание снимка состояния тома при помощи Linux LVM (Logical Volume Manager). Различные способы и их достоинства всесторонне рассмотрены в главе 12.

Копирование главного сервера

Утилита mysqldump позволяет выполнить все приведенные в этом разделе шаги за одно действие. Но чтобы пояснить действия, здесь мы выполним все шаги по отдельности, а более компактную версию вы найдете далее в этом разделе.

Создание копии главного сервера (рис. 2-5) начнем с его архивации. Предположим, что главный сервер работает, и в его кэше немало таблиц. Стало быть, чтобы найти позицию в двоичном журнале необходимо сбросить все таблицы и заблокировать БД, чтобы предотвратить внесение изменений. Для этого выполните команду FLUSH TABLES WITH READ LOCK:

```
master> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.02 sec)
```

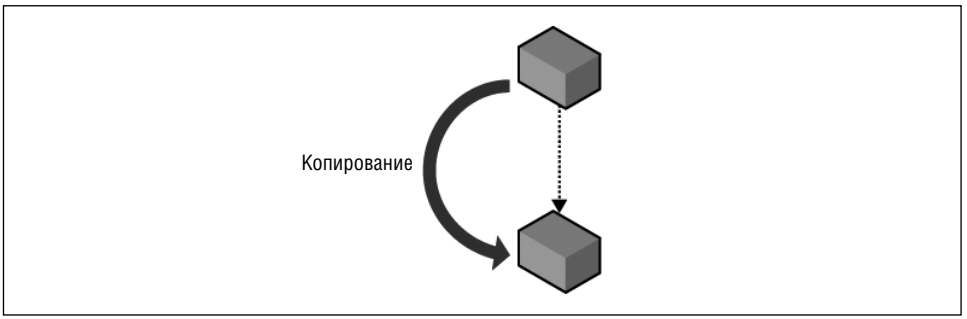


Рис. 2-5. Копирование главного сервера для создания нового подчиненного

После блокировки БД все готово для создания резервной копии и просмотра позиции в двоичном журнале. По причине того, что на главном сервере не происходит изменений, вы сможете корректно открыть текущий файл и позицию в двоичном журнале с помощью команды SHOW MASTER STATUS.

Подробнее о командах SHOW MASTER STATUS и SHOW MASTER LOGS рассказано в главе 6.

```
master> SHOW MASTER STATUS\G
*************************** 1. row ***************************
             File: master-bin.000042
         Position: 456552
      Binlog_Do_DB:
  Binlog_Ignore_DB:
1 row in set (0.00 sec)
```

Позиция следующего записываемого события, с которого должна начинаться репликация, — master-bin.000042, 456552. Все, что находится до указанной точки, будет в резервной копии. Записав позицию в двоичном журнале, переходите к созданию резервной копии. Резервную копию БД проще всего создать с помощью утилиты mysqldump:

```
$ mysqldump --all-databases --host=master-1 >backup.sql
```

Создав точную копию главного сервера, можно снять блокировку таблиц БД на главном сервере, и дать ему возможность продолжать обрабатывать запросы.

```
master> UNLOCK TABLES;
Query OK, 0 rows affected (0.23 sec)
```

Затем, восстановите резервную копию на подчиненном сервере с помощью утилиты mysql:

```
$ mysql --host=slave-1 <backup.sql
```

Восстановив резервную копию главного сервера на подчиненном, запустите подчиненный сервер. Здесь вам пригодится записанная ранее позиция в двоичном журнале. Настройте подчиненный сервер с помощью команды CHANGE MASTER TO и запустите подчиненный сервер:

```
slave> CHANGE MASTER TO
    ->  MASTER_HOST = 'master-1',
    ->  MASTER_PORT = 3306,
    ->  MASTER_USER = 'slave-1',
    ->  MASTER_PASSWORD = 'xyzzy',
    ->  MASTER_LOG_FILE = 'master-bin.000042',
    ->  MASTER_LOG_POS = 456552;
Query OK, 0 rows affected (0.00 sec)

slave> START SLAVE;
Query OK, 0 rows affected (0.25 sec)
```

Поздравляем! Вы выполнили копирование главного сервера, создав новый подчиненный сервер. В зависимости от загруженности главного сервера, подчиненному серверу, возможно, придется «догонять» главный с позиции, записанной вами, но это не так накладно, чем начинать с самого начала.

Многие из описанных выше шагов можно сделать автоматически при помощи утилиты mysqldump. Следующая команда позволяет создать логическую резервную копию всех БД на сервере с именем master:

```
$ mysqldump --host=master -all-databases \
> --master-data=1 >backup-источник.sql
```

Параметр --master-data=1 указывает на то, что утилита mysqldump должна записать инструкцию CHANGE MASTER TO с файлом и позицией в двоичном журнале, указанными на выводе команды SHOW MASTER STATUS.

Затем, восстановите резервную копию на подчиненном сервере:

```
$ mysql --host=slave-1 <backup-source.sql
```

Помните, что параметр --master-data=1 позволяет выполнить инструкцию CHANGE MASTER TO для главного сервера. Чтобы потом копировать подчиненный сервер, необходимо выполнить все шаги, приведенные в следующем разделе.

В зависимости от времени, потраченного на архивацию, объем не вошедших в архив данных на главном сервере может оказаться внушительным. Поэтому, прежде чем переводить подчиненный сервер в оперативный режим, прочитайте раздел «Обеспечение согласованности данных» ниже.

Копирование подчиненного сервера

Подключив подчиненный сервер к главному, можно пользоваться им, а не главным сервером, для создания новых подчиненных серверов. Таким образом можно создать новый подчиненный сервер, не переводя главный в автономный режим. Если ваша БД обширна или потребляет много трафика, время простоя может оказаться значительным, включая время, необходимое для создания резервной копии, и время, в течение которого подчиненные догонять «догонять» главный сервер.

Процесс копирования подчиненного сервера показан на рис. 2-6 и, в основе своей, напоминает копирование главного. Отличие заключается в поиске позиции в двоичном журнале. Необходимо также учесть, что копируемый вами подчиненный сервер выполняет репликацию главного сервера.

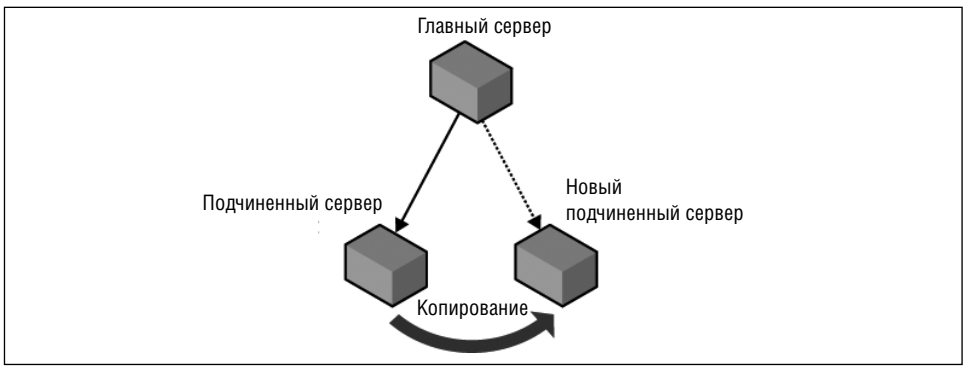


Рис. 2-6. Копирование подчиненного сервера для создания нового главного

Первое, что нужно сделать перед началом архивации, — остановить подчиненный сервер, чтобы прекратить его изменение. Если во время архивации выполняется репликация, архивный образ будет несогласованный, так как в момент резервного копирования БД в нее будут вноситься изменения. Исключение составляют некоторые методы оперативного резервного копирования, такие как InnoDB Hot Backup, где нет нужды останавливать подчиненный сервер перед созданием резервной копии.

```
original-slave> STOP SLAVE;
Query OK, 0 rows affected (0.20 sec)
```

После остановки подчиненного сервера, можно, как и в предыдущем примере, сбросить таблицы и создать архив. Создав резервную копию подчиненного (не главного) сервера, для определения места начала репликации не используйте команду SHOW SLAVE STATUS вместо SHOW MASTER STATUS. Вывод команды довольно внушительный, и о нем мы поговорим подробнее в главе 6, но чтобы получить позицию в двоичном журнале главного сервера, которая будет выполнена на подчиненном, запишите значения полей Relay_Master_Log_File и Exec_Master_Log_Pos.

```
original-slave> SHOW SLAVE STATUS\G
    ...
Relay_Master_Log_File: master-bin.000042
    ...
   Exec_Master_Log_Pos: 546632
```

Создав резервную копию и восстановив ее на новом подчиненном сервере, настройте запуск репликации с этой позиции и запустите подчиненный сервер:

```
new-slave> CHANGE MASTER TO
        ->  MASTER_HOST = 'master-1',
        ->  MASTER_PORT = 3306,
        ->  MASTER_USER = 'slave-1',
        ->  MASTER_PASSWORD = 'xyzzy',
        ->  MASTER_LOG_FILE = 'master-bin.000042',
        ->  MASTER_LOG_POS = 546632;
Query OK, 0 rows affected (0.19 sec)

new-slave> START SLAVE;
Query OK, 0 rows affected (0.24 sec)
```

Копирование главного и подчиненного серверов отличается лишь незначительно. В библиотеке Python можно объединить обе процедуры в одну, которая позволит создавать новые подчиненные серверы путем создания резервной копии на исходном сервере и подключения нового подчиненного сервера к главному.

Распространенный способ создания резервной копии — это: вызвать процедуру FLUSH TABLES WITH READ LOCK, затем создать архив файлов БД. Как правило, это намного быстрее, но использовать процедуру FLUSH TABLES WITH READ LOCK с InnoDB не безопасно!

Процедура FLUSH TABLES WITH READ LOCK блокирует таблицы, и новые транзакции будет невозможно начать. Однако остается несколько фоновых процессов, выполнение которых FLUSH TABLES WITH READ LOCK не блокирует.

Способы безопасного создания резервной копии таблиц InnoDB:

- Выключите сервер и копируйте файлы. Способ хорош для крупных БД, так как восстановление данных при помощи утилиты mysqldump занимает много времени.
- Запустите команду mysqldump после выполнения процедуры FLUSH TABLES WITH READ LOCK (как мы делали ранее).
- Выполнив процедуру FLUSH TABLES WITH READ LOCK, создайте снимки состояния тома при помощи одного из решений, например, LVM для Linux или ZFS (Zettabyte File System) для Solaris.

Сценарии операции копирования

В библиотеке Python копирование главного сервера осуществляется простым копированием находящейся на нем БД при помощи объекта Server, который представляет главный сервер. Для этого используется функция clone (лист. 2-7).

Копирование подчиненного сервера проводится аналогично, только резервная копия создается на одном сервере, в то время как новый подчиненный подключается к другому серверу для выполнения репликации. Поддерживать репликацию как главного, так и подчиненного серверов несложно при помощи двух параметров: параметра source, указывающего источник архивации, и параметра use_master, определяющего, куда должен подключиться подчиненный сервер после восстановления из резервной копии. Пример вызова метода clone:

```
clone(slave = slave[1], source = slave[0], use_master = master)
```

Следующий шаг — написать служебные функции для реализации копирования, которые пригодятся и для других целей. В лист. 2-6 представлены следующие функции:

fetch_master_pos Возвращает позицию в двоичном журнале главного сервера (то есть, позицию следующего события, которое будет записано в журнал).

fetch_slave_pos Возвращает позицию в двоичном журнале подчиненного сервера (то есть, позицию следующего события, которое будет прочитано из журнала главного сервера).

replicate_from Аргументы: подчиненный сервер, главный сервер и позиция в двоичном журнале. Нацеливает подчиненный сервер на репликацию главного с заданной позиции.

Функция replicate_from считывает поле repl_user на главном сервере, в котором находятся имя пользователя и пароль пользователя репликации. Заглянув в определение класса Server, этого поля вы не найдете. Оно добавляется во время установки на сервер роли Master.

Лист. 2-6. Служебные функции, возвращающие позиции главного и подчиненного сервера

```
_CHANGE_MASTER_TO = """CHANGE MASTER TO
   MASTER_HOST=%s, MASTER_PORT=%s,
   MASTER_USER=%s, MASTER_PASSWORD=%s,
   MASTER_LOG_FILE=%s, MASTER_LOG_POS=%s"""

def replicate_from(slave, master, position):
      slave.sql(_CHANGE_MASTER_TO, (master.host, master.port,
                                    master.repl_user.name,
                                    master.repl_user.passwd,
                                    position.file, position.pos))

def fetch_master_pos(server):
   result = server.sql("SHOW MASTER STATUS")
   return mysqlrep.Position(server.server_id, result["File"],
                            result["Position"])
def fetch_slave_pos(server):
   result = server.sql("SHOW SLAVE STATUS")
   return mysqlrep.Position(server.server_id,
                            result["Relay_Master_Log_File"], result["Exec_
                            Master_Log_Pos"])
```

Это все функции, необходимые для создания функции clone. При копировании подчиненного сервера, вызывающее приложение передает отдельный аргумент use_master, нацеливая новый подчиненный сервер на заданный главный для репликации. При копировании главного сервера вызывающее приложение опускает отдельный аргумент use_master, сообщая функции использовать в качестве главного сервера «исходный» сервер.

Способов создать резервную копию сервера много, а в лист. 2-7 приведен всего один — создание логической резервной копии сервера при помощи утилиты mysqldump. Далее мы покажем, как расширить процедуру архивации таким образом, чтобы с помощью этого же базового кода можно было организовать начальную загрузку новых подчиненных серверов с использованием произвольно взятых методов резервного копирования.

Лист. 2-7. Функция копирования главного или подчиненного сервера

```
def clone(slave, source, use_master = None):
   from subprocess import call
   backup_file = open(server.host + "-backup.sql", "w+")
   if master is not None:
      stop_slave(source)
   lock_database(source)
```

```
if master is None:
   position = fetch_master_position(source)
else:
   position = fetch_slave_position(source)
call(["mysqldump", "--all-databases", "--host='%s'" % source.host],
   stdout=backup_file)
if master is not None:
   start_slave(source)
backup_file.seek()         # К началу
call(["mysql", "--host='%s'" % slave.host], stdin=backup_file)
if master is None:
   replicate_from(slave, source, position)
else:
   replicate_from(slave, master, position)
start_slave(slave)
```

Распространенные задачи репликации и их решение

У всех известных стратегий масштабирования — горячий резерв и тому подобное — есть свои тонкости и «подводные камни». Мы покажем, как выполнять некоторые из этих задач, и как улучшить библиотеку Python для их поддержки.

В примерах к этому разделу мы опустили пароли. При настройке учетных записей для управления серверами вы можете открыть доступ только с некоторых узлов, управляющих развертыванием (создав учетную запись типа mats@'192.168.2.136'), либо вставить пароли в команды.

Создание отчетов

На большей части предприятий отчеты создаются регулярно: еженедельные отчеты о продажах, ежемесячные отчеты о затратах и расходах, а также разнообразные виды анализа данных с целью выявить тенденции или собрать фокус-группу для отдела маркетинга.

Выполнять такие запросы на главном сервере накладно. Запросы интеллектуального анализа данных требуют много вычислительных ресурсов и могут серьезно замедлить выполнение обычных операций, только лишь, чтобы установить, что, скажем, фокус-группу по маникюрным ножницам собирать не стоит. Более того, создание таких отчетов, как правило, дело не первой важности (в сравнении с обработкой обычных транзакций), и скорость здесь не требуется. Другими словами, эти отчеты не критичны ко времени, поэтому не важно, сколько времени уйдет на их создание — час или два.

Часто отчеты создают за определенный временной интервал, например: сводка продаж за день. Поэтому в нужный момент необходимо остановить

репликацию, чтобы не допустить попадание в отчет продаж из следующего дня. Вследствие того, что остановить подчиненный сервер на событии с указанной датой или временем невозможно, нужно найти другой способ создания отчетов.

Помните о запасном сервере (или двух, если отчетов действительно много)? Ведь он может реплицировать главный сервер. Для создания отчета нужно остановить репликацию, запустить приложения создания отчетов, затем повторно запустить репликацию — но все это без участия главного сервера.

Представьте, что один раз в сутки требуется создавать отчеты, в которые включены все транзакции от 0.00 до 23.59 часов. Необходимо остановить подчиненный сервер, предназначенный для создания отчетов. Таким образом, на нем будут выполнены события, произошедшие до полуночи, а события, произошедшие после полуночи, выполнены не будут. Нам лень делать это вручную, поэтому давайте подумаем, как автоматизировать процедуру. Достичь результата нам помогут эти шаги:

1. Перед наступлением полуночи, скажем, в 23.55, остановите подчиненный сервер отчетов, чтобы он не получал события от главного.
2. После полуночи найдите в двоичном журнале на главном сервере последнее событие, записанное до полуночи. Безусловно, если это сделать до наступления полуночи, можно пропустить некоторые события этого дня.
3. Запишите позицию в двоичном журнале данного события и запустите репликацию на подчиненном сервере до этой позиции.
4. Подождите, пока подчиненный сервер дойдет до этой позиции и остановится.

Вопрос номер один: как правильно запланировать выполнение заданий? В зависимости от ОС, есть различные способы это сделать. Мы опустим подробности, а о том, как запланировать выполнение заданий на Unix-подобных ОС, например, на Linux, см. в разделе «Планирование заданий в Unix».

Остановите подчиненный сервер, выполнив команду STOP SLAVE. После останова сервера найдите позицию в двоичном журнале.

```
slave> STOP SLAVE;
Query OK, 0 rows affected (0.25 sec)

slave> SHOW SLAVE STATUS\G
     ...
Relay_Master_Log_File: capulet-bin.000004
     ...
   Exec_Master_Log_Pos: 2456
1 row in set (0.00 sec)
```

Оставшиеся три шага выполняются до начала создания отчетов, и, как правило, являются частью сценария создания отчетов. Перед тем как создавать сценарий, давайте обдумаем каждый шаг.

Для чтения содержимого двоичного журнала нам подойдет утилита под названием mysqlbinlog. Данная утилита нам понадобится в во втором шаге, но подробнее о ней речь пойдет далее. В утилите mysqlbinlog есть два полезных параметра, --start-datetime и --stop-datetime, позволяющие прочесть только часть двоичного журнала. Итак, следующая команда предназначена для просмотра всех событий с момента останова подчиненного сервера до полуночи:

```
$ mysqlbinlog --force --read-from-remote-server --host=reporting.bigcorp.com \
> --start-datetime='2009-09-25 23:55:00' --stop-datetime='2009-09-25 23:59:59' \
> binlog files
```

Временная метка, хранящаяся в каждом событии, соответствует времени начала выполнения инструкции, а не времени записи инструкции в двоичный журнал.

Так как параметр --stop-datetime предписывает остановить вывод событий, начиная с первой временной метки после заданного времени и даты, то есть вероятность, что останется событие, выполнение которого началось до заданного времени, но запись в журнал о котором сделана после заданного времени — и это событие выпадет из установленного диапазона.

По причине того, что на главном сервере запись в двоичный журнал будет продолжаться, необходимо указать параметр --force. В противном случае, утилита mysqlbinlog не сможет прочитать открытый журнал. Для выполнения команды необходимо указать файлы двоичного журнала, которые нужно прочитать. Имена файлов зависят от параметров конфигурации, поэтому данные имена нужно извлечь из сервера. После этого, необходимо вычислить диапазон файлов двоичного журнала, которые нужно передать в команду mysqlbinlog. Получить список имен файлов двоичного журнала несложно при помощи команды SHOW BINARY LOGS:

```
master> SHOW BINARY LOGS;
+--------------------+----------+
| Log_name           |File_size |
+--------------------+----------+
| capulet-bin.000001 |    24316 |
| capulet-bin.000002 |     1565 |
| capulet-bin.000003 |      125 |
| capulet-bin.000004 |     2749 |
+--------------------+----------+
4 rows in set (0.00 sec)
```

В нашем примере только четыре файла, но их может быть намного больше. Просматривать большой список файлов, записанных до останова подчиненного сервера — пустая трата времени. Поэтому попытаемся уменьшить число файлов, которые нужно прочитать в поисках нужной позиции останова. Записав позицию в двоичном журнале на первом шаге после останова подчиненного сервера, несложно будет найти имя файла, на котором был остановлен подчиненный сервер, а затем вставить это имя и все следующие

за ним имена на ввод утилиты `mysqlbinlog`. Как правило, это один файл (или два, если в интервале между остановом подчиненного сервера и началом создания отчетов произойдет ротация двоичного журнала).

После выполнения команды mysqlbinlog, когда количество файлов двоичного журнала невелико, вы получите текстовый вывод каждого файла, содержащий некоторые сведения о событии.

```
$ mysqlbinlog --force --read-from-remote-server --host=reporting.bigcorp.com \
> --start-datetime='2009-09-25 23:55:00' --stop-datetime='2009-09-25 23:59:59' \
> capulet-bin.000004
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#090909 22:16:25 server id 1 end_log_pos 106 Start: binlog v 4, server v...
ROLLBACK/*!*/;
   .
   .
   .
# at 2495
#090929 23:58:36 server id 1 end_log_pos 2650 Query thread_id=27 exe...
SET TIMESTAMP=1254213690/*!*/;
SET /*!*/;
INSERT INTO message_board(user, message)
   VALUES ('mats@sun.com', 'Midnight, and I'm bored')
/*!*/;
```

Здесь интерес представляет поле end_log_pos последнего события в последовательности. В нашем примере, его значение равно 2650 — позиция, на которую будет записано следующее событие после полуночи.

Если вы внимательно рассмотрели вывод предыдущей команды, вы заметили, что в нем нет сведений о том, в каком файле двоичного журнала находится данная позиция. А чтобы найти событие необходимо знать файл. Если в команду mysqlbinlog передан один файл, то его имя очевидно, но если файлов два, то необходимо выяснить, в каком из них записано последнее события дня — в первом или во втором.

Обратите внимание на строку, с полем end_log_pos. В ней указан тип события. Каждый файл двоичного журнала начинается с события format description, строку которого можно видеть в предыдущем выводе. По format description можно узнать расположение искомого события. Если в выводе присутствуют два события format description, то искомое событие находится во втором файле, а если только одно — событие в первом файле.

Заключительный шаг перед началом создания отчетов — запустить репликацию и остановить ее точно на позиции, в которую должно быть записано событие, произошедшее после полуночи (уже записано, если быть до конца точным). Для этого подойдет малоизвестная команда START SLAVE

UNTIL. В эту команду нужно передать файл журнала главного сервера и позицию в журнале, на которой подчиненный сервер должен остановиться, после чего, она выполняет запуск подчиненного сервера. По достижении подчиненным сервером заданной позиции, он автоматически останавливается.

```
report> START SLAVE UNTIL
    ->  MASTER_LOG_POS='capulet-bin.000004',
    ->  MASTER_LOG_POS=2650;
Query OK, 0 rows affected (0.18 sec)
```

Подобно команде STOP SLAVE (без UNTIL), вывод команды START SLAVE UNTIL будет незамедлительный, а подчиненный сервер, вопреки ожиданиям, не успеет дойти до позиции, на которой ему предписано остановиться. Поэтому выполнение команд, запущенных после запуска STOP SLAVE UNTIL будет продолжаться, пока работает подчиненный сервер. Чтобы подождать, пока не подчиненный сервер достигнет позиции, на которой вы желаете остановить его, предназначена функция MASTER_POS_WAIT. Она блокирует выполнение команд до тех пор, пока подчиненный сервер не достигнет заданной позиции.

```
report> SELECT MASTER_POS_WAIT('capulet-bin.000004', 2650);
Query OK, 0 rows affected (231.15 sec)
```

Сейчас подчиненный сервер остановился на последнем событии дня, и все готово для анализа данных и формирования отчетов.

Создание отчетов при помощи библиотеки Python

Автоматизировать этот процесс на Python достаточно легко.

В лист. 2-8 содержится пример кода, позволяющего остановить создание отчетов в заданное время.

Функция fetch_remote_binlog считывает двоичный журнал с удаленного сервера с помощью команды mysqlbinlog. Содержимое файла(ов) будет возвращено в качестве итератора, организующего доступ к файлу, как к последовательности строк. Для оптимизации считывания можно добавить список файлов для проверки. Кроме того, чтобы уменьшить количество результатов, можно задать время и дату начала, и время и дату окончания. Все эти параметры будут переданы в программу mysqlbinlog.

Функция find_datetime_position выполняет поиск последнего поля end_log_pos в двоичном журнале, а также следит за количеством выполненных начальных событий. Она также запрашивает сервер отчетов о месте, на котором тот завершил чтение файла двоичного журнала, а затем запрашивает файлы журнала на главном сервере и находит файл, с которого нужно начать проверку.

Лист. 2-8. Программа на Python для запуска репликации по дате и времени

```
def fetch_remote_binlog(server, binlog_files=None,
                        start_datetime=None, stop_datetime=None):
    from subprocess import Popen, PIPE
    if not binlog_files:
         binlog_files = [
            row["Log_name"] for row in server.sql("SHOW BINARY LOGS")]
    command = ["mysqlbinlog",
               "--read-from-remote-server",
               "--force",
               "--host=%s" % (server.host),
               "--user=%s" % (server.sql_user.name)]
   if server.sql_user.passwd:
      command.append("--password=%s" % (server.sql_user.passwd))
   if start_datetime:
      command.append("--start-datetime=%s" % (start_datetime))
   if stop_datetime:
      command.append("--stop-datetime=%s" % (stop_datetime))
   return iter(Popen(command + binlog_files, stdout=PIPE).stdout)

def find_datetime_position(master, report, start_datetime, stop_datetime):
   from itertools import dropwhile
   from mysqlrep import Position
   import re

   all_files = [row["Log_name"] for row in master.sql("SHOW BINARY LOGS")]
   stop_file = report.sql("SHOW SLAVE STATUS")["Relay_Master_Log_File"]
   files = list(dropwhile(lambda file: file != stop_file, all_files))
   lines = fetch_remote_binlog(server, binlog_files=files,
                               start_datetime=start_datetime,
                               stop_datetime=stop_datetime)
   binlog_files = 0
   last_epos = None
   for line in lines:
      m = re.match(r"#\d{6}\s+\d?\d:\d\d:\d\d\s+"
                   r"server id\s+(?P<sid>\d+)\s+"
                   r"end_log_pos\s+(?P<epos>\d+)\s+"
                   r"(?P<type>\w+)", line)
      if m:
         if m.group("type") == «Start»:
            binlog_files += 1
         if m.group("type") == "Query":
            last_epos = m.group("epos")
   return Position(files[binlog_files-1], last_epos)
```

Следующие функции предназначены для синхронизации сервера отчетов перед созданием отчетов:

```
master.connect()
report.connect()
pos = find_datetime_position(master, report,
                             start_datetime="2009-09-14 23:55:00",
                             stop_datetime="2009-09-14 23:59:59")
report.sql("START SLAVE UNTIL MASTER_LOG_FILE=%s, MASTER_LOG_POS=%s",
          (pos.file, pos.pos))
report.sql("DO MASTER_POS_WAIT(%s,%s)", (pos.file, pos.pos))

      .
      .
код создания отчетов
      .
      .
```

Судите сами, с репликацией не так-то и сложно иметь дело. В данном листинге содержится ряд критических идей, которые пригодятся нам далее при изучении масштабирования: как запускать и останавливать подчиненный сервер в нужное время, как получить позиции в двоичном журнале или найти их с помощью стандартных средств и как все это интегрировать в ваше конкретное автоматизированное решение.

Планирование заданий в Unix

Проще всего добиться от подчиненного сервера, чтобы тот останавливался перед полуночью и приступал к созданию отчетов сразу после полуночи — это настроить задачу в программе cron(8), которая будет посылать команду на останов сервера и запуск сценария создания отчетов.

Например, следующие элементы crontab(5) управляют остановом подчиненного сервера перед полуночью и выполнением сценария отчетов, выполняющего откат подчиненного сервера вперед, примерно, в пять минут первого. Здесь сценарий stop_slave предназначен для останова подчиненного сервера, а сценарий daily_report — для создания ежедневного отчета (начинающегося с синхронизации, описанной выше).

```
stop reporting slave five minutes before midnight, every day
55 23 * * * $HOME/mysql_control/stop_slave

Run reporting script five minutes after midnight, every day
5 0 * * * $HOME/mysql_control/daily_report
```

Сохраним это в файле crontab, назовем его reporttab, и установим этот файл с помощью следующей команды:

```
$ crontab reporttab
```

Планирование заданий в Windows Vista

В Windows Vista гораздо проще планировать задания, чем в предыдущих версиях Windows. Программа Планировщик заданий (Task Scheduler) положительно подверглась пересмотру. Теперь Планировщик заданий (Task Scheduler) — это оснастка консоли MMC, интегрированная с программой Просмотр событий (Event Viewer), что позволяет использовать события в качестве триггеров для запуска заданий. Есть и другие параметры планирования и запуска.

Для запуска программы Планировщик заданий (Task Scheduler) в Windows Vista в меню Пуск (Start) откройте Панель управления (Control Panel), в папке Администрирование (Administrative Tools) откройте Планировщик заданий (Task Scheduler); или запустите компонент Выполнить (Run), нажав на сочетание клавиш Windows + R, и введите команду taskschd.msc. Для продолжения пройдите контроль учетных записей пользователей (User Account Control — UAC).

Чтобы создать триггер задачи, на панели действий (Action) щелкните **Создать простую задачу (Create Basic Task)**. На экране появится Мастер создания простой задачи (Create Basic Task Wizard).

На первой странице мастера присвойте заданию имя, если нужно, введите описание. Щелкните **Далее (Next)**.

На второй странице установите, как часто нужно выполнять задание. Здесь есть из чего выбрать: однократно, ежедневно, еженедельно и даже при входе в Windows или при наступлении определенного события. Выбрав нужный вариант, щелкните **Далее (Next)**.

В зависимости от заданной вами частоты, на третьей странице вам будет предложено указать параметры (например, дату и время) выполнения задания. Задав временные условия выполнения, щелкните **Далее (Next)**.

На четвертой странице выберите задачу или действие, которое следует выполнять при наступлении заданных условий. Это может быть запуск программы, отправка сообщения электронной почты или отображение сообщения для пользователя. Выберите действие и щелкните **Далее (Next)**.

В зависимости от действия, указанного вами на предыдущей странице, далее вы можете указать, что же должно произойти при выполнении задачи. Например, если вы выбрали запуск приложения, введите имя приложения или сценария, аргументы и укажите папку, в которой должно выполняться задание.

Предоставив все сведения, щелкните **Далее (Next)**. На последней странице проверьте задачу. Если все задано правильно, создайте задачу, щелкнув **Готово (Finish)**. Чтобы вернуться на предыдущие страницы и внести изменения, щелкните **Назад (Back)**. И наконец, здесь есть возможность, открыть страницу свойств после щелчка кнопки **Готово**. Это позволяет внести в задачу дополнительные изменения.

Заключение

В этой главе мы познакомились с репликацией MySQL, узнали зачем нужна репликация, и как ее настраивать. Мы также пробежались по двоичному журналу, в следующей главе рассмотрим его более подробно.

Джоэл завершил свой доклад г-ну Саммерсону о распределении нагрузки на четырех новых подчиненных серверах, и о его планах насчет расширения топологии в будущем.

— Молодец, Джоэл. А теперь еще раз — что такое подчиненный сервер?

Сдержав невольный вздох, Джоэл повторил:

— Подчиненный сервер — это копия данных, находящихся на сервере базы данных, он получает изменения с исходного, то есть главного, сервера базы данных...

ГЛАВА 3

Двоичный журнал событий

— Джоэл?

Джоэл вздрогнул, чуть не ударившись головой, и выполз из-под стола.

— Я тут переключал кое-какие кабели... — сказал он, будто оправдываясь.

Кивнув, г-н Саммерсон произнес непререкаемым тоном:

— Ты должен разобраться с проблемами из-за нового сервера в отделе маркетинга. Им нужно откатить данные к определенному моменту.

— Хорошо, посмотрим..., — начал было Джоэл, вспоминая, сохранились ли у него снимки предыдущих состояний системы.

— Я им сказал, что ты уже идешь.

После этого мистер Саммерсон развернулся и вышел. Следом заглянула девушка из разработчиков (ее Джоэл нашел очень привлекательной) и сказала:

— Он всегда такой, не принимай это на свой счет. Мы называем это «постановка задачи мимоходом».

Она рассмеялась и представилась:

— Меня зовут Эми.

Джоэл вышел из-за стола и подошел к ней:

— А я Джоэл...

После неловкой паузы Джоэл произнес:

— Я... ну... В общем, мне пора уже заняться этим делом.

Эми улыбнулась:

— Увидимся.

«Как бы с этим справиться...», — размышлял Джоэл, возвращаясь за стол, чтобы найти книгу по MySQL, которую он купил неделю назад.

В предыдущей главе было дано краткое введение в тему двоичных журналов событий. В этой главе мы разберем ее более подробно, приведем полное описание структуры двоичного журнала, формата события репликации и покажем, как использовать инструмент mysqlbinlog для анализа содержимого двоичных журналов и работы с ним.

В двоичном журнале протоколируются изменения, вносимые в базу данных, для последующего воспроизведения на любых подчиненных серверах.

Как правило, в двоичном журнале записываются все изменения, поэтому его также можно использовать для аудита модификации БД и восстановления состояния БД на заданный момент времени посредством повтора записанных в двоичном журнале действий.

В двоичном журнале содержатся только операторы, которые могут изменить БД. Заметьте: что регистрируются не только операторы, изменяющие БД, но и те, которые *потенциально способны* внести в нее изменения. Наибольшего внимания заслуживают те операторы, которые вносят изменения при соблюдении некоторого условия, такие как DROP TABLE IF EXISTS или CREATE TABLE IF NOT EXISTS, наряду с такими операторами, как DELETE и UPDATE с WHERE-условиями, не совпадающие ни с какими строками. Операторы SELECT обычно не протоколируются, поскольку они не вносят никаких изменений, но есть исключения.

Обычно транзакции на сервере выполняются не строго последовательно, а параллельно. Чтобы транзакции не конфликтовали и не генерировали несогласованный результат, сервер обеспечивает *упорядоченность* выполнения транзакции. Это означает, что транзакции выполняются так, что их выполнение дает тот же результат, что и строго *последовательное* выполнение этих транзакций.

В двоичном журнале транзакции регистрируется в порядке завершения этих транзакций на главном сервере. Несмотря на то, что на главном сервере транзакции могут выполняться параллельно, в двоичном журнале все они записываются последовательно, в порядке завершения.

Структура двоичного журнала

Концептуально двоичный журнал представляет собой последовательность *событий двоичного журнала (binlog events)* или просто *событий*. Как показано в главе 2, физически двоичный журнал состоит из нескольких файлов (рис. 3-1), в совокупности образующих двоичный журнал.

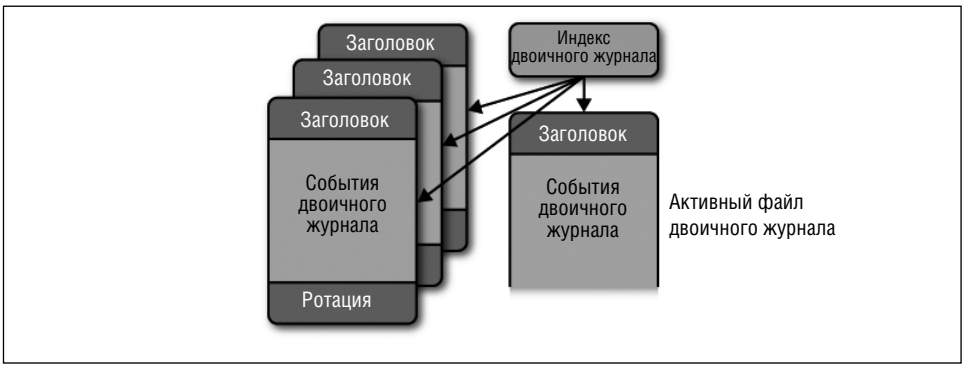


Рис. 3-1. Структура двоичного журнала событий

Фактические события хранятся в серии файлов, называемых *файлы двоичного журнала (binlog files)* с именами вида *host-bin.000001*, сопровождаемых *индексным файлом двоичного журнала* (*binlog index file)*, который обычно называется *host-bin.index* и отслеживает существующие файлы двоичного журнала. Файл двоичного журнала, в который сервер записывает данные в текущий момент, называется *активным* файлом двоичного журнала. Подчиненные серверы читают именно этот файл главного сервера. Именами файлов двоичного журнала и индексного файла двоичного журнала можно управлять, используя параметры log-bin и log-bin-index, о которых пойдет речь ниже.

Индексный файл отслеживает все файлы двоичного журнала, используемые сервером, обеспечивая корректное создание новых файлов двоичного журнала даже после перезапуска сервера. Каждая строка индексного файла содержит полное имя файла части двоичного журнала. Команды, влияющие на файлы двоичного журнала, такие как PURGE BINARY LOGS, RESET MASTER SERVER и FLUSH LOGS, действуют и на индексный файл, добавляя или удаляя строки согласно файлам, добавленным или удаленным с помощью этой команды.

Как показано на рис. 3-2, каждый файл двоичного журнала состоит из событий. В начале файла всегда находится событие Format_description, а в конце — событие Rotate. Обратите внимание, что при отключении или сбое сервера файл двоичного журнала может и не оканчиваться событием Rotate.

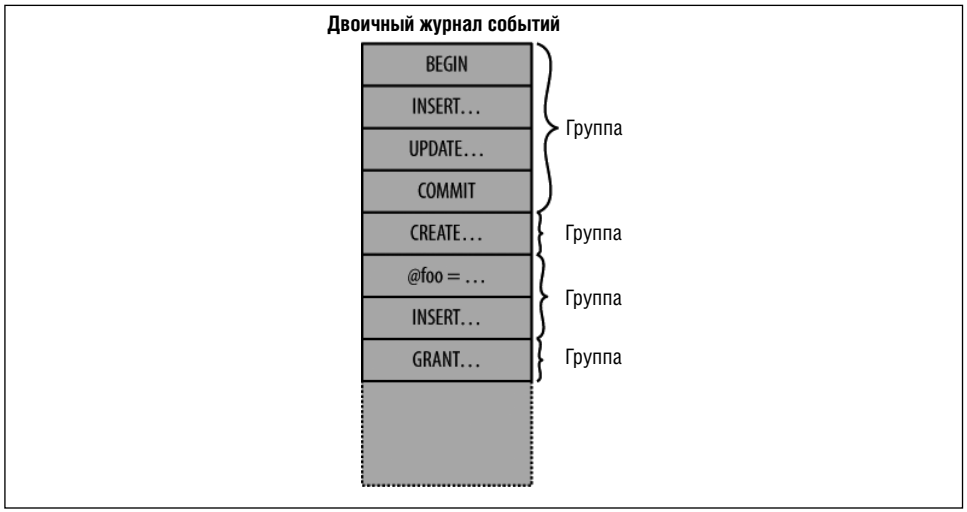


Рис. 3-2. Файл двоичного журнала с группами событий

Событие Format_description содержит информацию о сервере, который записал файл двоичного журнала, а также некоторую важную информацию о состоянии файла. В случае остановки и перезапуска сервера создается новый файл двоичного журнала и в него записывается новое событие Format_

description. Это необходимо, поскольку в период бездействия остановленного сервера вероятно внесение изменений в БД. Например, после обновления сервера должно быть записано новое событие Format _description.

По завершении записи в файл двоичного журнала добавляется событие Rotate, «закрывающее» файл. Это событие содержит ссылку на следующий файл двоичного журнала, включая имя файла и позицию, с которой следует начинать чтение. События Format_description и Rotate будут подробно описаны в следующем разделе.

За исключением событий Format_description и Rotate, события файла двоичного журнала сгруппированы в *группы*. В транзакционных механизмах БД каждая группа примерно соответствует транзакции. В случае механизмов БД, не поддерживающих транзакции, или операторов, которые не могут входить в транзакции (например, CREATE или ALTER), каждый оператор сам по себе является группой. Короче, каждая группа событий в файле двоичного журнала содержит либо один оператор, не принадлежащий транзакции, либо транзакцию, состоящую из нескольких операторов.

Как правило, каждая группа либо выполняется полностью, либо не выполняется совсем. Если по какой-то причине, подчиненный сервер останавливается посреди группы, репликация возобновится с начала группы, а не с последнего выполненного оператора (подробнее см. в главе 6).

Структура события двоичного журнала

В версии MySQL 5.0 был введен новый формат двоичного журнала — binlog 4. Предыдущие форматы сложно было расширять, вводя дополнительные поля, поэтому в binlog 4 была добавлена поддержка расширяемости. Этот формат по-прежнему совместим с форматом событий, использующимся во всех версиях, начиная с 5.0, но в каждой новой версии в формат двоичного журнала добавлялись новые события, а в отельные события — новые поля. Именно Binlog 4 и будет описан далее в этой главе.

Каждое событие двоичного журнала состоит из трех частей:

- *Общий заголовок* Общий заголовок, как ясно из его названия, является общим для всех событий в файле двоичного журнала. Общий заголовок содержит базовую информацию о событии. Наиболее важными полями являются тип события и размер события.
- *Постзаголовок* Постзаголовок индивидуален для каждого типа события; другими словами, для каждого типа событий в этом поле хранится разная информация. Размер этого заголовка, как и размер общего заголовка, остается неизменным во всех файлах двоичного журнала и задается событием Format_ description.
- *Тело события* Последним элементом любого события является тело переменного размера. Размер указывается в общем заголовке события. В теле хранятся основные данные о событии, которые различаются для

разных типов событий. Для события Query, например, в теле хранится запрос, а для события User_var — имя и значение переменной пользователя, только что заданное оператором.

Перечисление полного перечня форматов событий выходит за рамки данной книги, но поскольку события Format_description и Rotate важны для интерпретации других событий, мы кратко расскажем о них.

Как сказано выше, событие Format_description открывает любой файл двоичного журнала и содержит общую информацию о событиях в файле. Событие Format_description может отличаться в разных файлах, обычно по причине обновления или перезапуска сервера.

- *Версия формата файла двоичного журнала* Это версия файла двоичного журнала, которую не следует путать с версией сервера. В MySQL версий 3.23, 4.0 и 4.1 используется версия 3 двоичного журнала, тогда как MySQL 5.0 и выше — версия 4.

  Версия формата файла двоичного журнала изменяется при внесении разработчиками значительных изменений общую структуру файла или событий. В версии 5.0 начальное событие для файла двоичного журнала было изменено для поддержки другого формата. Общие заголовки событий также изменились, что и повлекло за собой изменение версии формата файла двоичного журнала.
- *Версия сервера* Это строка с номером версии сервера БД, создавшего файл. Она включает в себя версию сервера и дополнительную информацию, если используется специальная сборка. Обычный формат таков: номер версии из трех частей, после которого идет тире и дополнительные параметры сборки. Например, «5.1.40-debug-log» означает отладочную версию 5.1.40.
- *Длина общего заголовка* В этом поле хранится длина общего заголовка. Поскольку оно находится здесь, в Format_description, его значение может различаться в разных файлах двоичного журнала. Это верно для всех событий, за исключением Format_description и Rotate, которые изменяться не могут. Длина Format_description фиксирована, потому что любой сервер должен быть способен прочитать событие, сгенерированное любой версией сервера. У события Rotate общий заголовок фиксирован, потому что это событие используется при подключении подчиненного сервера к главному, т. е. оно вступает в дело до чтения любых других событий. Поэтому для этих двух событий размер общего заголовка фиксирован и не изменится в будущих версиях сервера.
- *Длина постзаголовка* Длина постзаголовка события не изменяется в конкретном файле двоичного журнала, в нем хранится массив значений длины постзаголовка для каждого типа событий. Поскольку количество событий зависит от сервера, то число событий, которое может сгенерировать сервер, сохраняется перед этим полем.

Поскольку и размер общего заголовка, и размер постзаголовка для каждого типа событий прописаны в событии Format_description, добавление новых событий и даже увеличение размера постзаголовков посредством добавления новых полей не повлияет на формат высокого уровня файла двоичного журнала.

При расширении формата необходимо тщательно следить, чтобы расширение не нарушило интерпретацию событий прежних версий. Например, в общий заголовок можно добавить поле с указанием на сжатие события типа компрессии, но учтите, что в отсутствие этого поля (например, в ситуациях, когда подчиненный сервер считывает события со старого главного сервера) серверу необходима возможность использовать прежний режим обработки событий.

Регистрация операторов в журнале

В MySQL традиционно применялась логическая репликация и лишь недавно была внедрена построчная репликация (см. главу 6). При логической репликации фактически выполняемый оператор пишется в двоичный журнал с дополнительной информацией, после чего этот оператор исполняется на подчиненном сервере. Не все операторы могут быть зарегистрированы таким образом, существуют исключения, о которых нужно знать. В этом разделе будет описан процесс регистрации операторов и даны важные предостережения.

Поскольку двоичный журнал — это общий ресурс, т. е. в нем регистрируют операторы все потоки сервера, важно предотвратить одновременное обновление двоичного журнала разными потоками. Для этого потоки устанавливают блокировку двоичного журнала (мьютекс LOCK_log) перед записью события в двоичный журнал и снимают ее сразу после записи. При этом нередки ситуации, что сразу несколько серверных потоков ожидают снятия блокировки.

Регистрация операторов языка манипулирования данными

Типичными операторами языка манипулирования данными (DML) являются операторы DELETE, INSERT и UPDATE. Для безопасности ведения журнала MySQL устанавливает на журнал блокировку во время записи и снимает ее после записи.

Чтобы гарантировать согласованность записей двоичного журнала и модификаций таблиц, изменяющий таблицу оператор регистрируется в двоичном журнале сразу после его завершения, но перед снятием блокировки с таблицы. В противном случае какой-нибудь оператор мог бы «втиснуться» между модификацией БД и регистрацией оператора в двоичном журнале. В результате порядок регистрации операторов в журнале отличался бы

от порядка их исполнения, что, очевидно, привело бы к рассинхронизации главного и подчиненных серверов. Например, оператор UPDATE с условие WHERE в такой ситуации может обновлять разные строки на главном и подчиненном серверах, поскольку «втиснувшийся» оператор мог изменить значения этих строк.

Регистрация операторов языка описания данных

Операторы языка описания данных (DDL) действуют на схему БД, примеры — CREATE TABLE и ALTER TABLE. Они создают или изменяют объекты файловой системы, например, описания таблиц в .frm–файлах и каталоги файловой системы, представляющие БД. Потому сервер хранит информацию о доступных объектах в виде структур данных в своей внутренней памяти. Для безопасного обновления структуры данных, например определения таблицы, необходимо установить на нее блокировку.

Поскольку для защиты этих структур данных используется единая блокировка, создание, изменение и разрушение объектов БД может привести к многочисленным проблемам с производительностью. Кстати, создание и разрушение временных таблиц, часто применяемых для хранения промежуточных результатов вычисления, также требует блокировки. Если вы создаете или разрушаете множество временных таблиц, то, уменьшив число этих операций, сможете повысить производительность.

Регистрация запросов

При логической репликации наиболее распространенным событием двоичного журнала является событие Query, представляющее операторы, выполняемые на главном сервере. Помимо оператора, это событие содержит дополнительную информацию, необходимую для его выполнения.

Вспомните, что двоичный журнал используется для разных целей, и порядок операторов в нем может отличаться от такового на главном сервере. В одних случаях часть двоичного журнала может быть воспроизведена на сервере для восстановления состояния на заданный момент времени (PITR), в других случаях репликация может начаться с середины журнала, поскольку до репликации на подчиненном сервере была восстановлена резервная копия. Наконец, администратор БД может в ручном режиме откорректировать двоичный журнал для устранения сбоев.

Во всех этих случаях события выполняются в различных *контекстах*. То есть, имеется информация, не заданная явно в момент выполнения оператора, но необходимая для корректного выполнения. Вот некоторые примеры:

- *Текущая база данных* Если оператор обращается к таблице, функции или процедуре без указания БД, для оператора неявно задана текущая БД.
- *Значение переменной, определяемой пользователем* Если оператор использует переменную, определяемую пользователем, значение этой переменной также относится к неявно заданному контексту для оператора.

- *Начальное число для функции* RAND Функция RAND генерирует псевдослучайные числа. Это означает, что сгенерированные числа могут повторяться, но являются случайными в том смысле, что подчиняются нормальному распределению. Для генерации используется начальное число, по которому с помощью псевдослучайной функции генерируются детерминированные последовательности. Это означает, что при одном и том же начальном числе функция RAND всегда будет возвращать идентичные последовательности. Тем не менее, начальное число с точки зрения оператора также задано неявно.
- *Текущее время* Очевидно, что время начала выполнения оператора относится к неявно заданному контексту. Время очень важно для вызова функций, зависимых от текущего времени, таких как NOW и UNIX_TIMESTAMP. Если время вызова оператора на главном и подчиненном серверах отличается, эти функции будут возвращать разные результаты.
- *Значение, используемое при вставке в поле с атрибутом* AUTO_INCREMENT Если оператор вставляет строку в таблицу с таким полем, то ее значение также явно не определено, поскольку зависит от строк, вставленных ранее.
- *Значение, возвращаемое вызовом* LAST_INSERT_ID Результат функции LAST_INSERT_ID зависит от значения, вставленного предыдущим оператором, что относит его к неявно заданному контексту оператора.
- *Идентификатор потока* Для некоторых операторов к контекст включается идентификатор потока, примером могут быть операторы, использующие временную таблицу или функцию CURRENT_ID.

Поскольку контекст для выполнения операторов не может быть известен в момент их повторного запуска при воспроизведении журнала на подчиненном сервере (либо на главном сервере после сбоя или перезапуска), необходимо сделать неявную информацию явной, добавив ее в двоичный журнал. Для этого существует несколько способов. Помимо вышеперечисленного, к неявному контексту относятся триггеры и хранимые процедуры, но об этом мы поговорим отдельно. Рассмотрим каждый вид неявно заданной информации отдельно.

Текущая база данных

В журнале фиксируется текущую БД и запросы к ней (в специальных полях Query). Эти поля также хранят события с операторами LOAD DATA INFILE (см. ниже), поэтому следующее описание действительно и для него.

Значения пяти функций вычисляются с использованием текущего времени и даты: NOW, CURDATE, CURTIME, UNIX_TIMESTAMP и SYSDATE. Первые четыре возвращают значение, основанное на времени начала исполнения оператора. В отличие от них, SYSDATE выдает значение time(2). Это различие лучше всего иллюстрирует сравнение вызова NOW и SYSDATE с промежуточной задержкой:

```
mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) |           SYSDATE() |
+---------------------+----------+---------------------+
| 2010-03-27 22:27:36 |        0 | 2010-03-27 22:27:38 |
+---------------------+----------+---------------------+
1 row in set (2.00 sec)

mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | OW()                |
+---------------------+----------+---------------------+
| 2010-03-27 22:27:49 |        0 | 2010-03-27 22:27:49 |
+---------------------+----------+---------------------+
1 row in set (2.00 sec)
```

Обе функций вычисляются сразу, но NOW возвращает время начала работы оператора, а SYSDATE выдает время с time(2).

Для корректной обработки этих функций в событии журнала сохраняется временная отметка *начала* выполнения. При воспроизведении журнала это значение копируется из поля события в поток выполнения подчиненного сервера и используется как время запуска оператора при вычислении функций, зависимых от времени.

Поскольку SYSDATE напрямую вызывает time(2), это небезопасно для репликации, т. к. результаты вызова на главном и подчиненном серверах будут различаться. Поэтому лучше воздерживаться от этой функции, зависимой от текущего времени.

События контекста

Неявно заданная информация необходима операторам, соответствующим таким условиям:

- если оператор содержит ссылку на переменную, определяемую пользователем (как в примере 3-1), необходимо добавить значение этой переменной в двоичный журнал;
- если оператор обращается к функции RAND, необходимо добавить начальное число функции в двоичный журнал;
- если оператор вызывает функцию LAST_INSERT_ID, необходимо добавить последний вставленный идентификатор в двоичный журнал;
- если оператор выполняет вставку в поле с атрибутом AUTO_INCREMENT, то необходимо добавить исходное значение этого поля в двоичный журнал.

Лист. 3-1. Операторы с переменными, определяемыми пользователем

```
SET @value = 45;
INSERT INTO t1 VALUES (@value);
```

В каждом из этих случаев в двоичный журнал добавляется одно или несколько *контекстных событий*, перед записью события, содержащего запрос. Поскольку таких событий может быть несколько, в двоичном журнале может несколько переменных, определяемых пользователем, вызовов функции RAND, или (почти) любые комбинации ранее перечисленных событий. Двоичный журнал хранит необходимую контекстную информацию в следующих событиях:

- *User_var* Каждое такое событие регистрирует имя и значение одной переменной, определяемой пользователем.
- *Rand* Фиксирует начальное значение случайного числа, используемое функцией RAND. Начальное число извлекается из состояния сеанса.
- *Intvar* Если оператор вставляется в поле с автоматическим приращением, это событие регистрирует значение внутреннего счетчика таблицы перед запуском оператора.

  Если оператор содержит вызов LAST_INSERT_ID, это событие регистрирует значение, которое вернула функция.

В примере 3-2 показаны некоторые операторы, которые генерируют все контекстные события, и то, как выглядят события при отображении с использованием SHOW BINLOG EVENTS. Обратите внимание, что перед каждым оператором может быть несколько контекстных событий.

Лист. 3-2. События запроса, использующего контекст

```
master> CREATE TABLE t1 (a INT AUTO_INCREMENT PRIMARY KEY, b INT, c CHAR(64));
Query OK, 0 rows affected (0.00 sec)

master> SET @foo = 12;
Query OK, 0 rows affected (0.00 sec)

master> SET @bar = 'Smoothnoodlemaps';
Query OK, 0 rows affected (0.00 sec)

master> INSERT INTO t1(b,c) VALUES (@foo,@bar), (RAND(), 'random');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

master> INSERT INTO t1(b) VALUES (LAST_INSERT_ID());
Query OK, 1 rows affected (0.00 sec)

master> SHOW BINLOG EVENTS FROM 238\G
*************************** 1. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 238
 Event_type: Query
  Server_id: 1
End_log_pos: 306
       Info: BEGIN
```

```
*************************** 2. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 306
 Event_type: Intvar
  Server_id: 1
End_log_pos: 334
       Info: INSERT_ID=1
*************************** 3. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 334
 Event_type: RAND
  Server_id: 1
End_log_pos: 369
       Info: rand_seed1=952494611,rand_seed2=949641547
*************************** 4. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 369
 Event_type: User var
  Server_id: 1
End_log_pos: 413
Info: @`foo`=12
*************************** 5. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 413
 Event_type: User var
  Server_id: 1
End_log_pos: 465
       Info: @`bar`=_latin1 0x536D6F6F74686E6F6F6... COLLATE latin1_swedish_ci
*************************** 6. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 465
 Event_type: Query
  Server_id: 1
End_log_pos: 586
       Info: use `test`; INSERT INTO t1(b,c) VALUES (@foo,@bar), (RAND(), ...
*************************** 7. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 586
 Event_type: Xid
  Server_id: 1
End_log_pos: 613
       Info: COMMIT /* xid=44 */
*************************** 8. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 613
 Event_type: Query
  Server_id: 1
End_log_pos: 681
       Info: BEGIN
```

```
*************************** 9. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 681
 Event_type: Intvar
  Server_id: 1
End_log_pos: 709
       Info: LAST_INSERT_ID=1
*************************** 10. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 709
 Event_type: Intvar
  Server_id: 1
End_log_pos: 737
       Info: INSERT_ID=3
*************************** 11. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 737
 Event_type: Query
  Server_id: 1
End_log_pos: 843
       Info: use `test`; INSERT INTO t1(b) VALUES (LAST_INSERT_ID())
*************************** 12. row ***************************
   Log_name: mysqld1-bin.000001
        Pos: 843 Event_type: Xid
  Server_id: 1
End_log_pos: 870
       Info: COMMIT /* xid=45 */ 12 rows in set (0.00 sec)
```

Идентификатор потока

Последний неявно заданный элемент, который иногда требуется для двоичного журнала, — это идентификатор потока для сеанса MySQL, исполняющего операторы. Идентификатор потока необходим зависимым от него функциям, использующим CONNECTION_ID,но еще важнее он для обработки временных таблиц.

Временные таблицы индивидуальны для каждого потока. Это означает, что две временные таблицы с одним и тем же именем могут сосуществовать при условии, что они определены в разных сеансах (потоках). Временные таблицы могут являться эффективным средством повышения производительности определенных операций, но требуют специальной обработки в связи с регистрацией в двоичном журнале.

Сервер работает с временными таблицами, создавая скрытые имена для хранения определений таблиц. Эти имена основаны на идентификаторе процесса сервера, идентификаторе потока, который создает таблицу, и специфичном для потока счетчике — это нужно, чтобы различать различные версии таблиц, созданные в одном потоке. Такая схема именования позволяет различать таблицы из разных потоков, но оператор сможет найти нужную ему таблицу, только если соответствующий идентификатор потока хранится в двоичном журнале.

Аналогично занесению в двоичный журнал текущей БД, идентификатор потока записывается в отдельное поле событий Query и поэтому может быть использован для вычисления специфичных для потока данных и корректной обработки временных таблиц. При регистрации события Query идентификатор потока считывается из серверной переменной pseudo_thread_id. Это означает, что он может быть задан перед выполнением оператора, но только если у вас есть привилегии SUPER. Эта серверная переменная позволяет mysqlbinlog корректно исполнять операторы, ее не следует использовать для других целей.

Событие Query для оператора, вызывающего функцию CONNECTION_ID или обращающегося к временной таблице, помечается в двоичном журнале как специфичное для потока. Поскольку идентификатор потока всегда присутствует в событии Query, этот флаг не обязателен и используется, главным образом, чтобы избежать излишней привязки к pseudo_thread_id.

Операторы LOAD DATA INFILE

Оператор LOAD DATA INFILE упрощает заполнение таблиц из файлов. К сожалению, он зависит от контекста, не охваченного контекстными событиями, которые мы обсуждали, а именно от файлов с данными.

Для обработки LOAD DATA INFILE сервер MySQL использует специальный набор событий, регистрирующих файл в двоичном журнале. Этот способ также очень удобен для передачи больших объемов данных от главного сервера к подчиненному — вскоре вы это увидите. Для корректной передачи и обработки оператора LOAD DATA INFILE вводятся несколько событий:

- *Begin_load_query* Это событие запускает передачу данных через файл.
- *Append_block* Серия из одного или нескольких таких событий позволяет сохранить все данные из файла, если размер файла превышает максимально допустимый для соединения размер пакета.
- *Execute_load_query* Это специализированный вариант события Query, которое содержит оператор LOAD DATA INFILE, выполняемый на главном сервере.

  Хотя оператор, содержащийся в этом событии, ссылается на имя файла на главном сервере, вместо поиска этого файла подчиненный сервер загрузит данные из событий Begin_load_query и Append_block.

Для каждого оператора LOAD DATA INFILE, выполненного на главном сервере, файл, подлежащий чтению, сопоставляется внутреннему буферу, связанному с файлом. Это буфер используется в последующей обработке. Кроме того, исполняемому оператору присваивается уникальный идентификатор, обозначающий файла, читаемый этим оператором.

В процессе выполнения оператора содержимое файла записывается в двоичный журнал как последовательность событий, начинающихся с со-

бытия Begin_load_query, обозначающего начало нового файла, за которым следует ноль или больше событий Append_block. Каждое событие, записанное в двоичный журнал, не превышает по размеру максимально допустимый объем пакета, заданный параметром max-allowed-packet.

После завершения чтения и загрузки файла выполнение оператора оканчивается внесением события Execute_load_query в двоичный журнал. Это событие содержит исполненный оператор вместе с идентификатором, назначенным требуемому оператором файлу. Обратите внимание, что в журнал заносится не исходный исполненный, а переработанный оператор.

В двоичных журналах старых версий можно обнаружить записи Load_log_event, Execute_log_event и Create_file_log_event. Такие события использовались для репликации оператора LOAD DATA INFILE до MySQL версии 5.0.3.

На примере листинга 3-3 показаны события, записанные в двоичный журнал в ходе успешного выполнения оператора LOAD DATA INFILE. В поле Info видеть присвоенный файлу идентификатор (в данном случае 1), также видно, что он фигурирует во всех событиях, сгенерированных при выполнении этого оператора. Файл foo.dat, который читает оператор, превышает максимально допустимый размер (16384 байт), и потому он разделен на три порции.

Лист. 3-3. Успешное выполнение LOAD DATA INFILE

```
master> SHOW BINLOG EVENTS IN 'master server-bin.000042' FROM 269\G
*************************** 1. row ***************************
   Log_name: master server-bin.000042
        Pos: 269
 Event_type: Begin_load_query
  Server_id: 1
End_log_pos: 16676
       Info: ;file_id=1;block_len=16384
*************************** 2. row ***************************
   Log_name: master server-bin.000042
        Pos: 16676
 Event_type: Append_block
  Server_id: 1
End_log_pos: 33083
       Info: ;file_id=1;block_len=16384
*************************** 3. row ***************************
   Log_name: master server-bin.000042
        Pos: 33083
 Event_type: Append_block
  Server_id: 1
End_log_pos: 33633
       Info: ;file_id=1;block_len=527
*************************** 4. row ***************************
   Log_name: master server-bin.000042
```

```
        Pos: 33633
 Event_type: Execute_load_query
  Server_id: 1
End_log_pos: 33756
       Info: use `test`; LOAD DATA LOCAL INFILE 'foo.dat' INTO ... ;file_id=1
4 rows in set (0.00 sec)
```

Фильтры двоичного журнала

Существует возможность фильтрации операторов из двоичного журнала с использованием параметров binlog-*-db: binlog-do-db и binlog-ignore-db. Параметр binlog-do-db используется, когда требуется воспроизвести только операторы, принадлежащие к определенной БД, игнорировав все остальные, а binlog-ignore-db позволяет, наоборот, игнорировать операторы из определенной БД, выполнив все остальные.

Эти параметры разрешается определять несколько раз. Например, чтобы исключить из репликации БД one_db и two_db, следует добавить в файл *my.cnf* следующие параметры:

```
[mysqld]
binlog-ignore-db=one_db
binlog-ignore-db=two_db
```

Во избежание путаницы мы объясним, как работает фильтрация, и дадим рекомендации о том, как избежать основных проблем. На рис. 3-3 показана схема фильтрации. Она производится на уровне операторов — оператор либо игнорируется целиком, либо целиком записывается в двоичный журнал. Параметры binlog-*-db рассматривают *текущую* БД (а не БД, с которой работает оператор). Поясним это на примере листинга 3-4, содержащего операторы, изменяющие таблицы в различных БД:

- строка 1 изменяет таблицу в текущей БД test, поскольку имя БД не указано в ссылке на таблицы;
- строка 2 изменяет таблицу в другой БД, а не в текущей;
- строка 3 изменяет таблицы в двух БД, ни одна из которых не является текущей.

Лист. 3-4. Операторы, использующие различные базы данных

```
USE bad; INSERT INTO t1 VALUES (1),(2);
USE bad; INSERT INTO good.t2 VALUES (1),(2);
USE bad; UPDATE good.t1, ugly.t2 SET a = b;
```

Рассмотрим пример с фильтрацией БД с помощью фильтра binlog-ignore-db=bad. Ни один из трех операторов лист. 3-4 не будет записан в двоичный журнал, хотя второй и третий операторы изменяют таблицы в обеих БД (*bad* и *ugly*). На первый взгляд это может показаться странным: почему бы не отфильтровать операторы по БД, таблицы которой они изменяют? Но предста-

вим, что бы произошло с третьим оператором, если бы вместо *bad* настроить фильтр на *ugly*. В этом случае третий оператор обновил бы одну БД, и сервер попал бы в «ловушку», чреватую рассогласованием БД. Проблема решается просто: фильтр действует только на текущую БД, и это правило используется для всех операторов (за некоторыми исключениями).

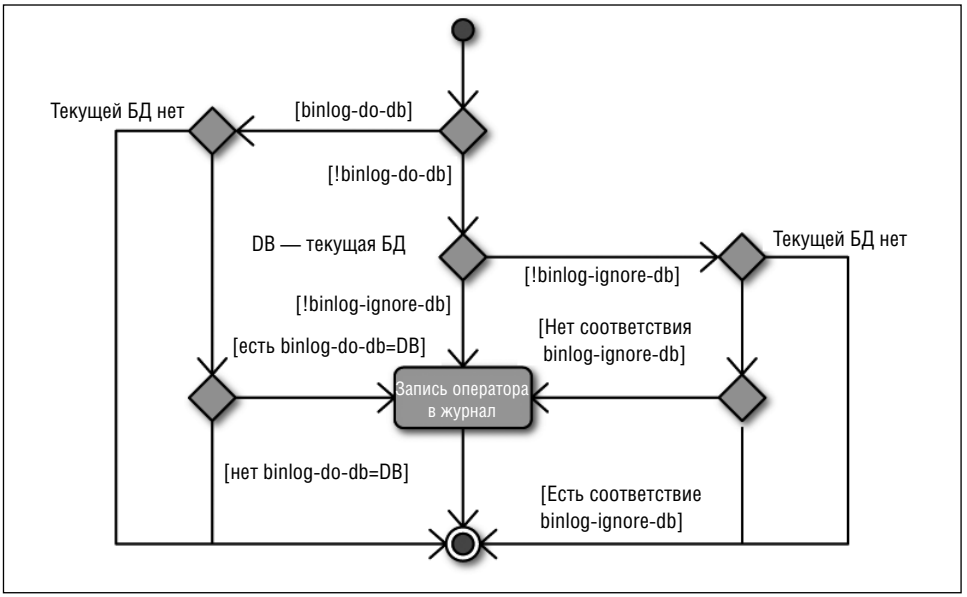


Рис. 3-3. Логическая схема действия фильтров binlog-*-db

Чтобы избежать ошибок при выполнении операторов, которые потенциально должны быть отфильтрованы, возьмите за правило не вставлять им БД в операторы, работающие с таблицами, функциями и процедурами. Вместо этого каждый раз, когда требуется обратиться к таблице из другой БД, делайте нужную БД текущей оператором USE. Другими словами, вместо

```
INSERT INTO other.book VALUES ('MySQL', 'Paul DuBois');
```

пишите так:

```
USE other; INSERT INTO book VALUES ('MySQL', 'Paul DuBois');
```

Этот режим не применяется при построчной репликации (о ней — в главе 6).Но, поскольку построчная репликация работает с модификациями строк, можно фильтровать операторы по имени таблицы, а не текущей БД.

А что происходит, когда и binlog-do-db и binlog-ignore-db используются одновременно? Рассмотрим такой конфигурационный файл:

```
[mysqld]
binlog-do-db=good
binlog-ignore-db=bad
```

Будет ли игнорироваться следующий оператор или нет?

```
USE ugly; INSERT INTO t1 VALUES (1);
```

Следуя логике рис. 3-3, находим, что если есть хоть одно правило binlog-do-db, все правила binlog-ignore-db полностью игнорируются, и поскольку явно активирована только БД *good*, указанный выше оператор будет фильтроваться.

Логика фильтров binlog-*-db делает бессмысленным одновременное объявление фильтров binlog-do-db и binlog-ignore-db. На самом деле лучше вовсе отказаться от binlog-*-db, поскольку двоичный журнал может быть использован как для восстановления, так и для репликации. Если же фильтр пропускает не все операторы двоичного журнала, то восстановить БД по двоичному журналу в случае сбоя не удастся.

Триггеры, события и хранимые программы

К особым структурам с точки зрения регистрации в журнале относятся хранимый в БД код — триггеры, события и хранимые стандартные программы (под последним понимают хранимые процедуры и функции). Их обработка в двоичном журнале имеет много общего, поэтому в этом разделе они рассматриваются вместе. Однако операторы, объявляющие и разрушающие хранимые программы, и операторы, вызывающие их, обрабатываются по-разному.

Операторы, объявляющие и разрушающие хранимые программы

В качестве примера ниже берутся триггеры, но те же принципы действуют для событий и хранимых программ. Чтобы понять, почему регистрация этих сущностей в журнале выполняется особым образом, рассмотрим листинг 3-5.

В этом примере таблица employee содержит информацию о сотрудниках воображаемой компании, а таблица log — журнал доступа к информации. Обратите внимание, что в log есть поле timestamp, куда заносится время модификации, а поле name таблицы employee является ее первичным ключом. Также предусмотрено поле status, содержащее сведение о результате вставки.

Для мониторинга изменений кадровых данных (например, для отчетности) создается три триггера, добавляющих запись в таблицу *log* при вводе, удалении и модификации записей сотрудников.

Обратите внимание, что триггеры идут после триггеров, т. е. записи в журнал добавляются только при успешном выполнении оператора, неудачные попытки исполнения не регистрируются (эту функцию мы добавим позже).

Лист. 3-5. Объявление таблиц и триггеров

```
CREATE TABLE employee (
   name CHAR(64) NOT NULL,
   email CHAR(64),
   password CHAR(64),
   PRIMARY KEY (name)
);
```

```
CREATE TABLE log (
   id INT AUTO_INCREMENT,
   email CHAR(64),
   message TEXT,
   ts TIMESTAMP,
   PRIMARY KEY (id)
);

CREATE TRIGGER tr_employee_insert_after AFTER INSERT ON employee FOR EACH ROW
   INSERT INTO log(email, status, message)
      VALUES (NEW.email, 'OK', CONCAT('Adding employee ', NEW.name));

CREATE TRIGGER tr_employee_delete_after AFTER DELETE ON employee FOR EACH ROW
   INSERT INTO log(email, status, message)
      VALUES (OLD.email, 'OK', 'Removing employee');

delimiter $$
CREATE TRIGGER tr_employee_update_after AFTER UPDATE ON employee FOR EACH ROW
BEGIN
   IF OLD.name != NEW.name THEN
      INSERT INTO log(email, status, message)
         VALUES (OLD.email, 'OK',
            CONCAT('Name change from ', OLD.name, ' to ', NEW.name));
   END IF;
   IF OLD.password != NEW.password THEN
      INSERT INTO log(email, status, message)
         VALUES (OLD.email, 'OK', 'Password change');
   END IF;
   IF OLD.email != NEW.email THEN
      INSERT INTO log(email, status, message)
         VALUES (OLD.email, 'OK', CONCAT('E-mail change to ', NEW.email));
   END IF;
END $$
delimiter ;
```

Теперь можно добавлять и удалять записи сотрудников (лист. 3-6), каждая такая операция регистрируется в таблице *log*. Операции добавления, удаления и изменения записи сотрудников могут быть выполнены пользователем, имеющим доступ к таблице *employee*, но как насчет доступа к таблице *log*? У пользователя, который может работать с таблицей *employee*, не должно быть возможности изменять таблицу *log*. Для этого существует множество причин, но все они сводятся к обеспечению достоверности содержимого таблицы *log*, необходимой для соответствия нормативам и аудита. Поэтому администратор БД может предоставить доступ к таблице *employee* кому угодно, но доступ к таблице *log* — лишь считанным пользователям.

Чтобы триггеры смогли изменять защищенную таблицу, они выполняются от имени пользователя, объявившего триггер, а не пользовате-

ля, модифицирующего таблицу *employee*. Поэтому операторы CREATE TRIGGER в лист. 3-5 выполняются администратором базы данных, у которого есть полномочия на вставку в таблицу log, а операторы, изменяющие информацию о сотрудниках (лист. 3-6) работают под учетными данными пользователей, которым разрешено изменять только таблицу employee.

После исполнения операторов лист. 3-6 модифицировать таблицу *employee* можно будет под записью сотрудника отдела кадров, а редактировать таблицу *log* — только под записью администратора БД. Учетная запись сотрудника отдела кадров не позволит добавлять или удалять записи из таблицы *log*.

Как видно из лист. 3-6, приходится записывать пароли в пользовательские переменные, прежде чем использовать их в операторе. Это нужно, чтобы не отправлять пароли открытым текстом на другой сервер (подробнее об этом см. во врезке ниже).

Лист. 3-6. Добавление, удаление и изменение пользователей

```
master> SET @pass = PASSWORD('xyzzy');
Query OK, 0 rows affected (0.00 sec)

master> INSERT INTO employee VALUES ('mats', 'mats@example.com', @pass);
Query OK, 1 row affected (0.00 sec)

master> UPDATE employee SET name = 'matz' WHERE email = 'mats@example.com';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

master> SET @pass = PASSWORD('foobar');
Query OK, 0 rows affected (0.00 sec)

master> UPDATE employee SET password = @pass WHERE email = 'mats@example.com';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

master> DELETE FROM employee WHERE email = 'mats@example.com';
Query OK, 1 row affected (0.00 sec)

master> SELECT * FROM log;
+----+-----------------+-----------------------------+---------------------+
| id | email           | message                     | ts                  |
+----+-----------------+-----------------------------+---------------------+
| 1 | mats@example.com | Adding employee mats        | 2010-01-13 18:56:08 |
| 2 | mats@example.com  | Name change from mats to matz | 2010-01-13 18:56:11|
| 3 | mats@example.com | Removing employee           | 2010-01-13 18:57:11|
+---+-----------------+-----------------------------+---------------------+
3 rows in set (0.00 sec)
```

Безопасность и двоичный журнал

Пользователь с привилегиями REPLICATION SLAVE может читать запись всего, что происходит на главном сервере, поэтому его учетная запись должна быть надлежащим образом защищена. Подробное описание мер безопасности находится за рамками данной книги, но некоторые предосторожности мы упомянем:

1. Запретите вход под этой учетной записью из-за брандмауэра.
2. Ведите журнал попыток входа под этой учетной записью на отдельном защищенном сервере.
3. Защищайте шифрованием соединения между главным и подчиненным серверами, например с помощью встроенной поддержки SSL.

Несмотря на защиту учетной записи, существует информация, которой не должно быть в двоичном журнале. Одним из типичных примеров являются пароли. События, содержащие пароли, для доступа к таблицам, могут быть записаны в двоичный журнал при выполнении операторов, модифицирующих таблицы на сервере:

```
UPDATE employee SET pass = PASSWORD('foobar')
   WHERE email = 'mats@example.com';
```

При репликации лучше переработать этот оператор, исключив из него пароль. Вместо пароля следует записать его хэш-значение в пользовательскую переменную, которую и следует использовать:

```
SET @password = PASSWORD('foobar');
UPDATE employee SET pass = @password WHERE email = 'mats@example.com';
```

Поскольку оператор SET не реплицируется, пароль не записывается в двоичный журнал и хранится только в памяти сервера во время выполнения оператора.

Это сработает, если в таблице сохранен хэш, а не сам пароль открытым текстом. В противном случае избежать попадания пароля в двоичный журнал не удастся. Так или иначе, рекомендуется хранить пароли в форме хэш-значения — так удастся защитить пароли даже от тех, кто умудрится украсть двоичные журналы.

Шифрование соединения между главным и подчиненным серверами обеспечивает некоторую защиту, но если скомпрометирован сам двоичный журнал, шифрование не поможет.

Возвращаясь к обсуждению контекста, можно сказать, что записи пользователя, вызывающего программу и пользователя, объявляющего триггер, относятся к неявно заданному контексту. Как будет показано в главе 6, при исполнении триггера на подчиненном сервере не важно, под какой записью был объявлен или сработал триггер. Напротив, пользовательские учетные записи очень важны при PITR-восстановлении.

Чтобы воспроизвести содержимое двоичного журнала на сервере без проблем из-за полномочий на доступ к различным таблицам, следует выполнять

все операторы под учетной записью SUPER. Но не факт, что триггеры были объявлены под записью SUPER, поэтому можно заново объявить триггеры от имени подходящей учетной записи. Объявление триггера от имени учетной записи SUPER вместо пользователя, исходно объявившего триггер, может окончиться неоправданным повышением привилегий.

Администратор БД может указывать запись пользователя, под которой будет выполняться триггер, используя необязательную секцию DEFINER в конструкции CREATE TRIGGER. Если такая секция не задана в синтаксисе оператора (см. лист. 3-7), соответствующий оператор будет переписан при регистрации в двоичном журнале: в него будет добавлена секция DEFINER, а в качестве объявившего будет указан текущий пользователь. В итоге определение триггера в двоичном журнале примет вид, показанный в лист. 3-7. В нем указана учетная запись пользователя, создавшего триггер (root@localhost), что и требуется в данном случае.

Лист. 3-7. Оператор CREATE TRIGGER в двоичном журнале

```
master> SHOW BINLOG EVENTS FROM 92236 LIMIT 1\G
*************************** 1. row ***************************
   Log_name: master-bin.000038
        Pos: 92236
 Event_type: Query
  Server_id: 1
End_log_pos: 92491
       Info: use `test`; CREATE DEFINER=`root`@`localhost` TRIGGER ...
1 row in set (0.00 sec)
```

Вызовы триггеров и хранимых программ

Разобравшись с объявлениями, перейдем к вызовам. Сразу скажем, что в действительности вызовы вовсе не обрабатываются в журнале. Вернее, оператор, вызвавший срабатывание триггера, регистрируется в двоичном журнале, но сам он не привязан к конкретному триггеру. То есть, исполнение оператора на подчиненном сервере вызывает срабатывание всех триггеров, связанных с таблицами, на которые ссылается исполняемый оператор. Это означает, что на главном и подчиненном серверах могут быть объявлены разные триггеры. При этом триггеры, объявленные на главном сервере, сработают на главном сервере, а триггеры, объявленные на подчиненном сервере, сработают на подчиненном сервере. Например, если триггер, регистрирующий модификации в журнале, не требуется на подчиненном сервере, то можно повысить производительность, удалив этот триггер на подчиненном сервере.

При этом любой контекст, необходимый для корректной репликации, будет записан в двоичный журнал до оператора, инициирующего срабатывание триггера, даже если этот контекст необходим просто самому триггеру. Так, в листинге 3-8 показан двоичный журнал после выполнения оператора INSERT из листинга 3-5. Обратите внимание, что первым в журнал запи-

сывается событие INSERT ID для первичного ключа таблицы log. Данный факт отражает модификацию таблицы log триггером, но подчиненному серверу это не нужно, так как на нем этот триггер работать не будет.

Заметьте, однако, что использование разных триггеров либо полный отказ от триггеров на главном или подчиненном серверах является исключительной ситуацией, и регистрация INSERT ID необходима для корректной репликации оператора INSERT в случае, когда триггер присутствует и на главном и на подчиненном сервере.

Лист. 3-8. Содержание двоичного журнала после выполнения INSERT

```
master> SHOW BINLOG EVENTS FROM 93340\G
*************************** 1. row ***************************
   Log_name: master-bin.000038
        Pos: 93340
 Event_type: Intvar
  Server_id: 1
End_log_pos: 93368
       Info: INSERT_ID=1
*************************** 2. row ***************************
   Log_name: master-bin.000038
        Pos: 93368
 Event_type: User var
  Server_id: 1
End_log_pos: 93396h
       Info: @`pass`=_latin1
0x2A394235303033343342433532453239313131373245423532 41... COLLATE
latin1_swedish_ci
*************************** 3. row ***************************
   Log_name: master-bin.000038
        Pos: 93396
 Event_type: Query
  Server_id: 1
End_log_pos: 93537
       Info: use `test`; INSERT INTO employee VALUES ...
3 rows in set (0.00 sec)
```

Хранимые процедуры

Хранимые функции, хранимые процедуры и события известны под общим названием *хранимые программы*. Но, поскольку в обработке хранимых процедур и хранимых функций много различий, эти вопросы рассматриваются в отдельных разделах (хранимые процедуры — в этом, а функции — в следующем).

В обработке хранимых процедур и триггеров есть несколько сходных аспектов, но различий все же больше. Как и триггеры, хранимые процедуры поддерживают секцию DEFINER, которая явно записывается в двоичный

журнал независимо от того, была ли она в исходном операторе. Но вызов хранимых стандартных процедур обрабатывается по-другому, нежели вызов триггеров.

Для начала добавим в код, показанный на лист. 3-6, вспомогательные процедуры для управления данными сотрудников. Все это можно сделать с помощью стандартных операторов INSERT, DELETE и UPDATE, но мы выбрали хранимые процедуры для демонстрации особенностей их регистрации в двоичном журнале. Допишем сначала функции для добавления и удаления записей сотрудников (лист. 3-9).

Лист. 3-9. Хранимые процедуры для управления записями сотрудников

```
delimiter $$
CREATE PROCEDURE employee_add(p_name CHAR(64), p_email CHAR(64),
                              p_password CHAR(64))
   MODIFIES SQL DATA
BEGIN
   DECLARE pass CHAR(64);
   SET pass = PASSWORD(p_pass)
   INSERT INTO employee(name, email, password)
      VALUES (p_name, p_email, pass);
END $$

CREATE PROCEDURE employee_passwd(p_email CHAR(64), p_password CHAR(64))
   MODIFIES SQL DATA
BEGIN
   DECLARE pass CHAR(64);
   SET pass = PASSWORD(p_password)
   UPDATE employee SET password = pass WHERE email = p_email;
END $$

CREATE PROCEDURE employee_del(p_name CHAR(64))
   MODIFIES SQL DATA
BEGIN
   DELETE FROM employee WHERE name = p_name;
END $$
delimiter ;
```

Для процедур employee_add и employee_passwd зашифрованный пароль записывается в отдельную переменную по упомянутым выше причинам, а процедура employee_del содержит только оператор DELETE, поскольку больше от нее ничего не требуется. Вот как записан в двоичный журнал вызов одной функции:

```
master> SHOW BINLOG EVENTS FROM 97911 LIMIT 1\G
*************************** 1. row ***************************
   Log_name: master-bin.000038
        Pos: 97911
 Event_type: Query
```

```
  Server_id: 1
End_log_pos: 98275
       Info: use `test`; CREATE DEFINER=`root`@`localhost`PROCEDURE ...
1 row in set (0.00 sec)
```

Как и ожидалось, в запись вызова в журнале добавлена секция DEFINER, в остальном код остался прежним. Обратите внимание, что оператор CREATE PROCEDURE реплицируется в событии Query, как и все остальные DDL-операторы.

Объявления хранимых процедур заносится в журнал так же, как объявления триггеров, но обработка их вызовов существенно отличается. На примере лист. 3-10 демонстрируется вызов процедуры, добавляющей запись работника, и сгенерированный в результате фрагмент журнала.

Лист. 3-10. Вызов хранимой процедуры

```
master> CALL employee_add('chuck', 'chuck@example.com', 'abrakadabra');
Query OK, 1 row affected (0.00 sec)

master> SHOW BINLOG EVENTS FROM 104033\G
*************************** 1. row ***************************
   Log_name: master-bin.000038
        Pos: 104033
 Event_type: Intvar
  Server_id: 1
End_log_pos: 104061
Info: INSERT_ID=1
*************************** 2. row ***************************
   Log_name: master-bin.000038
        Pos: 104061
 Event_type: Query
  Server_id: 1
End_log_pos: 104416
       Info: use `test`; INSERT INTO employee(name, email, password)
             VALUES ( NAME_CONST('p_name',_latin1'chuck' COLLATE 'latin1_
             swedish_ci'), NAME_CONST('p_email',_latin1'chuck@example.com'
             COLLATE 'latin1_swedish_ci'), NAME_CONST('pass',_latin1'
             *FEB349C4FDAA307A...' COLLATE 'latin1_swedish_ci'))
2 rows in set (0.00 sec)
```

В этом примере следует обратить внимание на следующее:

- Сам оператор CALL не записан в двоичный журнал, вместо него регистрируются операторы вызванной процедуры, или тело хранимой процедуры.
- Оператор переписан: в нем все ссылки на параметры хранимой процедуры (p_name, p_email и p_password) заменены вызовами NAME_CONST.
- Локальная переменная pass также заменена NAME_CONST, зашифрованный пароль содержится во втором параметре.

- Как и при регистрации в журнале вызова триггера, перед вызовом хранимой процедуры записывается событие Intvar с идентификатором записи журнала (строки, добавленной в таблицу *log*).

Поскольку ни имена параметров, ни локальные переменные не доступны вне хранимой процедуры, эти элементы с помощью NAME_CONST сопоставляются с константами, необходимыми для выполнения функции. Так удается использовать эти значения как параметры и локальные переменные. Впрочем, это незначительное изменение, которое не дает никаких преимуществ по сравнению с непосредственным использованием параметров.

Хранимые функции

У хранимых функций много общего с хранимыми процедурами и есть кое-какие сходства с триггерами. Аналогично тем и другим, хранимые функции имеют секцию DEFINER, которая часто (но не всегда) добавляется при записи оператора CREATE FUNCTION в двоичный журнал.

В отличие от хранимых процедур, хранимые функции возвращают значения и могут внедряться в SQL-код. Рассмотрим хранимую функцию для извлечения адреса сотрудника с заданным именем (лист. 3-11). Этот пример немного надуман, т. к. гораздо проще напрямую извлечь нужные данные, но она очень наглядна.

Лист. 3-11. Хранимая функция

```
delimiter $$
CREATE FUNCTION employee_email(p_name CHAR(64))
   RETURNS CHAR(64)
   DETERMINISTIC
BEGIN
   DECLARE l_email CHAR(64);
   SELECT email INTO l_email FROM employee WHERE name = p_name;
   RETURN l_email;
END $$
delimiter ;
```

Эту хранимую функцию удобно использовать в других операторах (см. лист. 3-12). В отличие от хранимых процедур, вызовы хранимых функций следует помечать как DETERMINISTIC, NO SQL или READS SQL DATA, если они должны быть записаны в двоичный журнал.

Лист. 3-12. Примеры использования хранимой функции

```
master> INSERT INTO collected(name, email) ('mats', employee_email('mats'));
Query OK, 1 row affected (0.01 sec)

master> SELECT employee_email('mats');
+------------------------+
| employee_email('mats') |
+------------------------+
```

```
| mats@example.com       |
+------------------------+
1 row in set (0.00 sec)
```

Вызовы хранимых функций реплицируются так же, как вызовы триггеров: как часть оператора, взывающего функцию. Так, репликация оператора INSERT, показанного на лист. 3-12, не требует предварять этот оператор в двоичном журнале какими-либо событиями. Все необходимые контекстные сведения будут добавлены в журнал вместе с записями, необходимыми для репликации хранимой функции, вызываемой из INSERT.

А что с оператором SELECT? Как правило, операторы SELECT не записываются в двоичный журнал, поскольку они не изменяют никакие данные. Но SELECT, содержащий хранимую функцию, является исключением. При выполнении хранимой функции сервер замечает вставку строки в таблицу *log* и помечает соответствующий оператор SELECT как модифицирующий данные и подлежащий записи в двоичный журнал.

Хранимые функции и привилегии

Привилегия CREATE ROUTINE необходима для объявления хранимых процедур и функций. Строго говоря, другие привилегии для этого не нужны, но, поскольку функции и процедуры обычно выполняются с привилегиями создателя, нет смысла объявлять их, не имея прав на доступ к таблицам и пр. ресурсам, с которыми эта процедура или функция работает.

Однако потоки репликации на подчиненном сервере работают без проверки привилегий, что открывает серьезную дыру в системе безопасности, позволяющую любому пользователю с привилегией CREATE ROUTINE повышать свои привилегии и выполнять любые команды на подчиненном сервере.

В версиях MySQL ниже 5.0 с этим не было проблем, поскольку все потоки, исполняющие операторы, подвергаются анализу при выполнении на главном сервере. Оператор, который входит за рамки привилегий, не записывается в двоичный журнал на главном сервере, и пользователь не сможет получить доступ к запретным для него объектам на подчиненном сервере. Однако с появлением хранимых процедур появилась поддержка условного ветвления, и сервер более не в состоянии анализировать все потоки при выполнении хранимой процедуры.

Поскольку в журнал записываются составляющие хранимые процедуры операторы, исполняемые на главном сервере, они будут выполнены и на подчиненном сервере. Однако оператор заносится в журнал только после успешного выполнения на главном сервере, поэтому получить через него доступ к запрещенным объектам невозможно, но с хранимыми функциями это не так.

Если хранимая функция объявлена с меткой SQL SECURITY INVOKER, злоумышленник может создать функцию, которая будет выполняться поразному на главном и на подчиненном серверах. Дыру в безопасности можно спрятать и в одной из ветвей условной конструкции, исполняемой на подчиненном сервере. Вот пример:

```
CREATE FUNCTION magic()
   RETURNS CHAR(64)
   SQL SECURITY INVOKER
BEGIN
   DECLARE result CHAR(64);
   IF @@server_id <> 1 THEN
      SELECT what INTO result FROM secret.agents LIMIT 1;
      RETURN result;
   ELSE
      RETURN 'I am magic!';
   END IF;
END $$
```

Одна часть кода выполняется только на главном сервере (ветвь ELSE), а другая часть (ветвь IF) — только на подчиненном сервере, где проверка привилегий отключена. Таким образом, обладатель привилегии CREATE ROUTINE получает возможности, равные привилегии SUPER.

Обратите внимание, что эта проблема не возникает, если функция объявляется с меткой SQL SECURITY DEFINER, поскольку такая функция выполняется с привилегиями пользователя и будет блокирована на подчиненном сервере.

Чтобы предотвратить повышение привилегий на подчиненном сервере, для объявления хранимых функций по умолчанию требуется привилегия SUPER. Но, поскольку хранимые функции очень удобны, а некоторые администраторы БД доверяют своим пользователям создание функций, эта проверка может быть деактивирована с помощью параметра log-bin-trust-function-creators.

События

Поддержка событий — это расширение MySQL, она не входит в стандартный язык SQL. Такие события не следует путать с событиями двоичного журнала, они обрабатываются хранимой программой, которая регулярно запускается специальным планировщиком событий.

Аналогично иным хранимым программам, определения событий также содержат секцию DEFINER. Поскольку события вызывают планировщик событий, их код выполняется только от имени объявившего событие, что исключает риск для безопасности, существующий в случае хранимых функций. При обработке событий соответствующие операторы записываются в двоичный журнал напрямую. Поскольку события вызываются на главном сервере, они автоматически отключаются и не вызываются на подчиненном сервере, иначе они были бы обработаны дважды.

На подчиненном сервере события отключены, но при потере связи с главным сервером события необходимо включить.

Например, важно включить обработку событий при повышении подчиненного сервера (см. главу 4). Легче всего это сделать так:

```
UPDATE mysql.events
      SET Status = ENABLED
    WHERE Status = SLAVESIDE_DISABLED;
```

Цель проверки — активировать только те события, которые были деактивированы при репликации с главного сервера (события могут отключаться и по другим причинам).

Специальные структуры

Обычно логическая репликация выполняется достаточно просто, но некоторые специальные структуры требуют осторожного обращения. Вспомните, что для того, чтобы оператор корректно выполнялся на подчиненном сервере, контекст необходимо откорректировать для оператора. Несмотря на то, что контекстные события, описанные выше, обрабатывают часть контекста, некоторые конструкции имеют дополнительный контекст, который не передается как часть процесса репликации.

Функция LOAD_FILE

Функция LOAD_FILE позволяет программно загружать файлы. Это довольно удобно, но файл должен существовать на подчиненном сервере для нормального выполнения репликации, поскольку сам файл во время репликации не передается, в отличие от файлов, загруженных с помощью LOAD DATA INFILE. Проявив немного изобретательности, можно переписать оператор, заменив LOAD_FILE на LOAD DATA INFILE, либо поместив содержимое файла в пользовательскую переменную. Рассмотрим следующий оператор INSERT:

```
master> INSERT INTO document(author, body)
    ->    VALUES ('Mats Kindahl', LOAD_FILE('go_intro.xml'));
```

Вы можете переписать этот оператор, задействовав LOAD DATA INFILE. Только будьте осторожны и выбирайте строку-разделитель полей и строк, которая не встретится в документе, поскольку вы собираетесь читать все содержимое файла в одно поле.

```
master> LOAD DATA INFILE 'go_intro.xml' INTO TABLE document
    ->    FIELDS TERMINATED BY '@*@' LINES TERMINATED BY '&%&'
    ->    (author, body) SET author = 'Mats Kindahl';
```

Альтернатива — сохранить содержимое файла в переменной, определяемой пользователем, а затем использовать ее в коде.

```
master> SET @document = LOAD_FILE('go_intro.xml');
master> INSERT INTO document(author, body) VALUES('Mats Kindahl, @document);
```

Модификации без транзакций и обработка ошибок

До этого момента мы рассматривали только модификации, которые осуществляются в составе транзакций, и совершенно не касались обработки ошибок. При использовании транзакций обработка ошибок довольно несложная: если при модификации таблицы, защищенной транзакциями, возникнет ошибка, никаких изменений внесено не будет. В этом вся суть транзакций — изменения, которые вносит сбойный оператор, можно смело игнорировать. То же самое верно для транзакций, которые откатываются: они не оказывают никакого влияния на таблицы и поэтому могут быть просто отброшены без риска рассогласования главного и подчиненного серверов.

«Изюминка» MySQL — поддержка нетранзакционных механизмов БД. Они дают некоторый выигрыш в скорости за счет отказа от журнала транзакций, необходимого транзакционным механизмам БД, и дополнительной оптимизации доступа к диску. Но с точки зрения репликации нетранзакционные механизмы БД требуют особого подхода.

Самое важное, на что следует обратить внимание, — это то, что поддержка репликации для любых нетранзакционных механизмов БД невозможна без некоторых допущений. В версиях, выше 5.1, некоторые ограничения снимаются при использовании построчной репликации (подробнее об этом см. в главе 6) — но даже она не обеспечивает поддержку любых механизмов БД.

Дополнительно усложняет репликацию возможность использовать механизмы БД, поддерживающие транзакции, и не поддерживающие их, в одной транзакции и даже в одном операторе. Вернемся к примеру на лист. 3-5: назначим для таблицы *log* нетранзакционный механизм БД, а для таблицы *employee* — транзакционный. В первом случае выбран не поддерживающий транзакции механизм MyISAM для повышения скорости. Дополним пример поддержкой мониторинга неудачного добавления записей сотрудников, создав пару триггеров, срабатывающих до и после вставки. Наличие в журнале поля *status* с значением FAIL означает, что триггер в начале вставки сработал, а триггер в конце вставки — нет, следовательно попытка добавить запись сотрудника была неудачной.

Лист. 3-13. Назначение механизмов БД для таблиц

```
CREATE TABLE employee (
   name CHAR(64) NOT NULL,
   email CHAR(64),
   password CHAR(64),
   PRIMARY KEY (email)
) ENGINE = InnoDB;

CREATE TABLE log (
   id INT AUTO_INCREMENT,
   email CHAR(64),
```

```
   message TEXT,
   status ENUM('FAIL', 'OK') DEFAULT 'FAIL',
   ts TIMESTAMP,
   PRIMARY KEY (id)
) ENGINE = MyISAM;

delimiter $$
CREATE TRIGGER tr_employee_insert_before BEFORE INSERT ON employee FOR EACH ROW
BEGIN
   INSERT INTO log(email, message)
      VALUES (NEW.email, CONCAT('Adding employee ', NEW.name));
   SET @LAST_INSERT_ID = LAST_INSERT_ID();
END $$
delimiter ;

CREATE TRIGGER tr_employee_insert_after AFTER INSERT ON employee FOR EACH ROW
   UPDATE log SET status = 'OK' WHERE id = @LAST_INSERT_ID;
```

Какое же влияние оказывает это изменение на двоичный журнал?

Для начала рассмотрим оператор INSERT из лист. 3-6. Допустим, оператор не защищен транзакцией, но параметр AUTOCOMMIT установлен в 1, оператор сам по себе будет являться транзакцией. Если оператор выполнен без ошибок, исполнение продолжится, а оператор будет записан в двоичный журнал в составе события Query.

Теперь рассмотрим, что происходит, если снова вызвать INSERT с теми же данными сотрудника. Поскольку поле *email* является первичным ключом, возникнет ошибка из-за дублирования ключа при попытке вставки. Но что будет с оператором, запишется ли он в двоичный журнал или нет?

Давайте посмотрим...

```
master> SET @pass = PASSWORD('xyzzy');
Query OK, 0 rows affected (0.00 sec)

master> INSERT INTO employee(name,email,pass)
    ->   VALUES ('mats','mats@example.com',@pass);
ERROR 1062 (23000): Duplicate entry 'mats@example.com' for key 'PRIMARY'
master> SELECT * FROM employee;
+------+-----------------+-------------------------------------------+
| name | email           | password                                  |
+------+-----------------+-------------------------------------------+
| mats | mats@example.com | *151AF6B8C3A6AA09CFCCBD34601F2D309ED54888 |
+------+-----------------+-------------------------------------------+
1  row in set (0.00 sec)

master> SHOW BINLOG EVENTS FROM 38493\G
*************************** 1. row ***************************
   Log_name: master-bin.000038
        Pos: 38493
```

```
 Event_type: User var
  Server_id: 1
End_log_pos: 38571
       Info: @`pass`=_latin1 0x2A313531414636423843334136414130394346434...
             COLLATE latin1_swedish_ci
*************************** 2. row ***************************
   Log_name: master-bin.000038
        Pos: 38571
 Event_type: Query
  Server_id: 1
End_log_pos: 38689
       Info: use `test`; INSERT INTO employee(name,email,pass) VALUES ...
2  rows in set (0.00 sec)
```

Видно, что оператор заносится в двоичный журнал, хотя таблица *employee* защищена транзакциями, а оператор не был исполнен. Анализ таблицы показывает, что в ней зарегистрирован только один сотрудник с такими данными. Это подтверждает откат оператора. Но почему оператор все-таки записан в двоичный журнал?

Анализ таблицы *log* поможет обнаружить причину.

```
master> SELECT * FROM log;
+---+-----------------+---------------------+------+---------------------+
| id | email           | message             | status| ts                  |
+---+-----------------+---------------------+------+---------------------+
| 1 | mats@example.com | Adding employee mats | OK   | 2010-01-13 15:50:45 |
| 2 | mats@example.com | Name change from ... | OK   | 2010-01-13 15:50:48 |
| 3 | mats@example.com | Password change      | OK   | 2010-01-13 15:50:50 |
| 4 | mats@example.com | Removing employee    | OK   | 2010-01-13 15:50:52 |
| 5 | mats@example.com | Adding employee mats | OK   | 2010-01-13 16:11:45 |
| 6 | mats@example.com | Adding employee mats  | FAIL | 2010-01-13 16:12:00 |
+---+-----------------+---------------------+------+---------------------+
6 rows in set (0.00 sec)
```

Посмотрите на последнюю строчку с состоянием FAIL. Это строка была добавлена в таблицу триггером, срабатывающим в начале вставки: tr_employee_insert_before. Чтобы в двоичном журнале правильно отражались модификации БД, сделанные на главном сервере, необходимо записать оператор в двоичный журнал, он вызывает другие операторы или триггеры, не охваченные транзакциями. Поскольку вызов оператора был неудачным, триггер завершения tr_employee_insert_after не был выполнен, и осталось состояние FAIL, установленное первым триггером.

Информация о неудачном вызове оператора на главном сервере должна быть записана в двоичный журнал. Для этого MySQL заносит в соответствующее поле события Query код ошибки, вызвавшей сбой оператора, и записывает это поле в двоичный журнал вместе с событием. Код ошибки

не отображается командой SHOW BINLOG EVENTS, но выводится утилитой mysqlbinlog, о которой рассказывается ниже.

Регистрация транзакций

Мы познакомились с записью в двоичный журнал отдельных операторов и контекстной информации. Транзакции требуют дополнительной обработки при регистрации в журнале.

Транзакция может начаться в одном из следующих случаев:

- когда пользователь вызывает START TRANSACTION или BEGIN;
- когда установлен параметр AUTOCOMMIT=1 и вызван оператор, обращающийся к защищенной транзакциями таблице. Обратите внимание, что операторы, модифицирующие лишь только таблицы, не защищенные транзакциями (например, MyISAM), не запускает транзакцию;
- когда установлен параметр AUTOCOMMIT=0 и предыдущая транзакция была зафиксирована или отменена неявно (вызовом оператора, выполняющего неявную фиксацию) или явно, оператором COMMIT или ROLLBACK.

Не каждый оператор, вызванный после запуска транзакции, является частью этой транзакции. Исключения требуют особого подхода с точки зрения двоичного журнала.

Нетранзакционные операторы по определению не входят в транзакции. Они действуют сразу после вызова, не ожидая завершения транзакции. Это также означает, что их невозможно откатить. Они также влияют на незавершенные транзакции: любой транзакционный оператор, вызванный после нетранзакционного, все равно добавляется к текущей незавершенной транзакции.

Кроме того, некоторые операторы вызывают неявную фиксацию транзакций. Их можно разделить на три группы по причине фиксации.

- *Операторы, записывающие файлы.* Большинство операторов DDL (CREATE, ALTER, и т. д.) неявно фиксируют незавершенные транзакции перед началом выполнения и после завершения. Эти операторы изменяют файлы файловой системы, и потому сами не являются транзакционными.
- *Операторы, модифицирующие таблицы БД mysql.* Все операторы, которые создают, удаляют или изменяют учетные записи или привилегии пользователей, вызывают неявную фиксацию и не могут входить в транзакции. Эти операторы модифицируют внутренние таблицы БД *mysql*, ни одна из которых не защищена транзакциями.

  В версиях MySQL ниже 5.1.3 эти операторы не вызывают неявной фиксации, но при обращении к нетранзакционным таблицам они обрабатываются как нетранзакционные операторы. Поддержка неявной фиксации добавлена для этих операторов в последующих версиях, что вызвало некоторую путаницу.

- *Операторы, требующие неявной фиксации для своей работы.* Операторы, блокирующие таблицы, административные операторы и LOAD DATA INFILE вызывают неявную фиксацию, поскольку это необходимо для их корректной работы.

Очевидно, что такие операторы не могут входить в транзакции, поскольку все незавершенные транзакции фиксируется до начала их выполнения. Полный список операторов, вызывающих неявную фиксацию, см. в электронной справке MySQL.

Кэш транзакций

Порядок операторов в двоичном журнале может отличаться от порядка их исполнения, поскольку в журнале приходится объединять операторы из разных транзакций. Разные сеансы могут одновременные начинать транзакции, и транзакционные механизмы БД ведут журналы транзакций, обеспечивающие корректное выполнение транзакций. Эти журналы скрыты от пользователя. Напротив, в двоичном журнале видны все транзакции из всех сеансов, записанные в порядке их фиксации, как если бы они выполнялась строго последовательно.

Для записи каждой транзакции в двоичный журнал как единого целого сервер должен разделять операторы, выполняемые в разных потоках. При фиксации транзакции сервер заносит все операторы из состава транзакции в двоичный журнал как единую операцию. С этой целью для каждого потока сервер поддерживает *кэш транзакций* (рис. 3-4). Каждый оператор, выполняемый в транзакции, помещается в кэш, содержимое которого затем копируется в двоичный журнал и опустошается после фиксации транзакции.

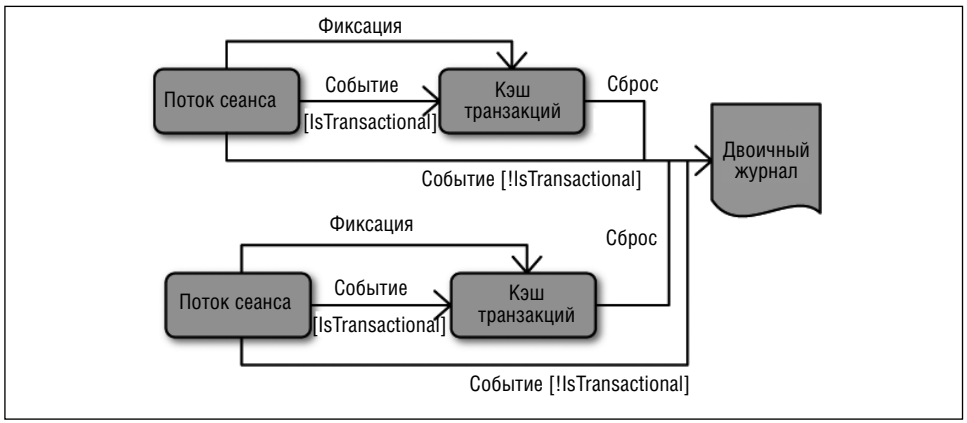


Рис. 3-4. Потоки с кэшем транзакций и двоичный журнал

Операторы, модифицирующие ресурсы вне транзакций, требуют особого внимания. Вспомним, что такие операторы не вызывают завершения текущей транзакции и потому эти модификации требуется где-то регистрировать, не фиксируя текущую транзакцию. С операторами, модифицирую-

щими одновременно транзакционные и нетранзакционные таблицы, все еще сложнее. Эти операторы считаются транзакционными, но выполняют и модификации, которые не являются частью транзакции. Логическая репликация не обеспечивает корректной обработки таких ситуаций. Решение этой проблемы будет описано ниже.

Регистрация нетранзакционных операторов

Если не начата ни одна транзакция, то нетранзакционные операторы пишутся напрямую в двоичный журнал, минуя кэш транзакций. Если существует незавершенная транзакция, то правила обработки оператора следующие:

1. Если оператор помечен как транзакционный, он пишется в кэш транзакций.
2. Если оператор не помечен как транзакционный и кэш транзакций пуст, этот оператор пишется напрямую в двоичный журнал.
3. Если оператор не помечен как транзакционный, но в кэше транзакций есть операторы, оператор пишется в кэш транзакций.

Третье правило может показаться странным, поясним его на примере лист. 3-14, вспомнив таблицы-примеры *employee* и *log*. Код с лист. 3-14 в одной транзакции модифицирует сначала таблицу, поддерживающую транзакции, а затем таблицу, не поддерживающую транзакции.

Лист. 3-14. Транзакционные и нетранзакционные операторы

```
1  START TRANSACTION;
2  SET @pass = PASSWORD('xyzzy');
3  INSERT INTO employee(name,email,password)
   VALUES ('mats','mats@example.com', @pass);
4  INSERT INTO log(email, message)
   VALUES ('root@example.com', 'This employee was bad');
5  COMMIT;
```

Согласно третьему правилу, оператор в строке 4 записывается в кэш транзакции, хотя таблица, которую он модифицирует, не поддерживает транзакции. Если бы этот оператор был записан напрямую в двоичный журнал, он оказался бы перед оператором из строки 3, т. к. последний не попал бы в двоичный журнал до фиксации транзакции (строка 5). Короче, на подчиненном сервере комментарий попал бы в таблицу *log* раньше, чем запись, для которой он предназначен, что явно приведет к рассинхронизации с главным сервером. Третье правило исключает такие ситуации. Слева на рис. 3-5 показаны нежелательные последствия в отсутствие этого правила, а справа — реальные события при действии третьего правила.

Правило 3 — это компромисс: когда нетранзакционный оператор кэшируется в процессе выполнения транзакции, есть риск, что две транзакции изменят нетранзакционную таблицу на главном сервере в порядке, отличном от того, в котором они записаны в двоичный журнал. Такая ситуация возможна при зависимости между первым транзакционным и вторым нетранзакцион-

ным оператором транзакции. Сервер не обрабатывает подобные ситуации, поскольку это потребовало бы синтаксического анализа всех операторов, включая сработавшие триггеры, с анализом зависимостей. Технически это возможно, но сильно увеличивает объем вычислений из-за необходимости обработки *всех* операторов, исполняемых до фиксации транзакции, что сильно снизило бы производительность. Тщательная проверка зависимостей на этапе разработки почти стопроцентно исключает эту проблему, поэтому создатели MySQL не пошли на такие издержки.

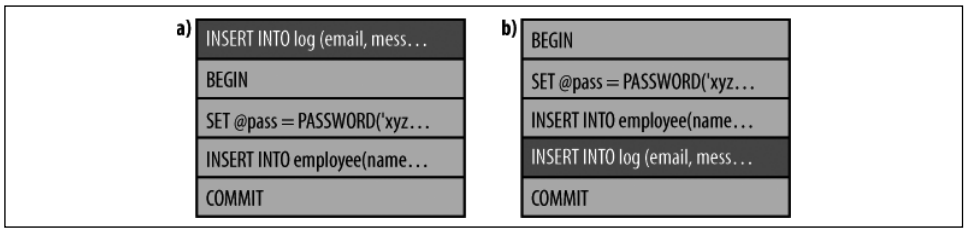


Рис. 3-5. Результат применения правила 3

Как избежать проблем с репликацией

Стратегия, позволяющая избежать вышеописанных проблем из-за зависимостей, такова. Операторы, модифицирующие нетранзакционные таблицы, должны выполняться в ходе транзакции первыми, тогда они сразу будут записаны в двоичный журнал, поскольку кэш транзакций будет пуст (см. правило 2 в предыдущем разделе). Известно, что у таких операторов нет зависимостей. Если какие-либо результаты исполнения этих операторов потребуются позже в ходе транзакции, можно записать их во временные таблицы или переменные.

Распределенная обработка транзакций с использованием XA

MySQL 5.0 способен координировать транзакции, использующие различные ресурсы, с использованием XA (расширенной архитектуры модели обработки транзакций X/Open Distributed Transaction Processing). В настоящее время XA не очень популярна, но удобна для координирования транзакций с любыми видами ресурсов. Модель XA используется в самом MySQL 5.0 для управления двоичным журналом и механизмами БД.

Особые команды открывают клиенту доступ к XA-синхронизации. XA позволяет обрабатывать операторы, вводимые разными пользователями, как единую транзакцию. Однако за эту возможность приходится расплачиваться производительностью, поэтому администраторы часто отключают эту функцию.

Команды XA не рассматриваются в этой книге, но архитектура этой модели кратко описана ниже — это необходимо, чтобы разобраться в том, как XA

влияет на двоичный журнал. Расширенная архитектура включает *диспетчер транзакций*, координирующий работу *диспетчеров ресурсов*, обеспечивая фиксацию глобальной транзакции как единой операции. Каждой транзакции присваивается уникальный XID-идентификатор, который используется диспетчером транзакций и диспетчерами ресурсов. В сервере MySQL роль диспетчера транзакций обычно исполняет двоичный журнал, а роли диспетчеров ресурсов — механизмы БД. Процесс фиксации XA-транзакции показан на рис. 3-6, он состоит из двух фаз.

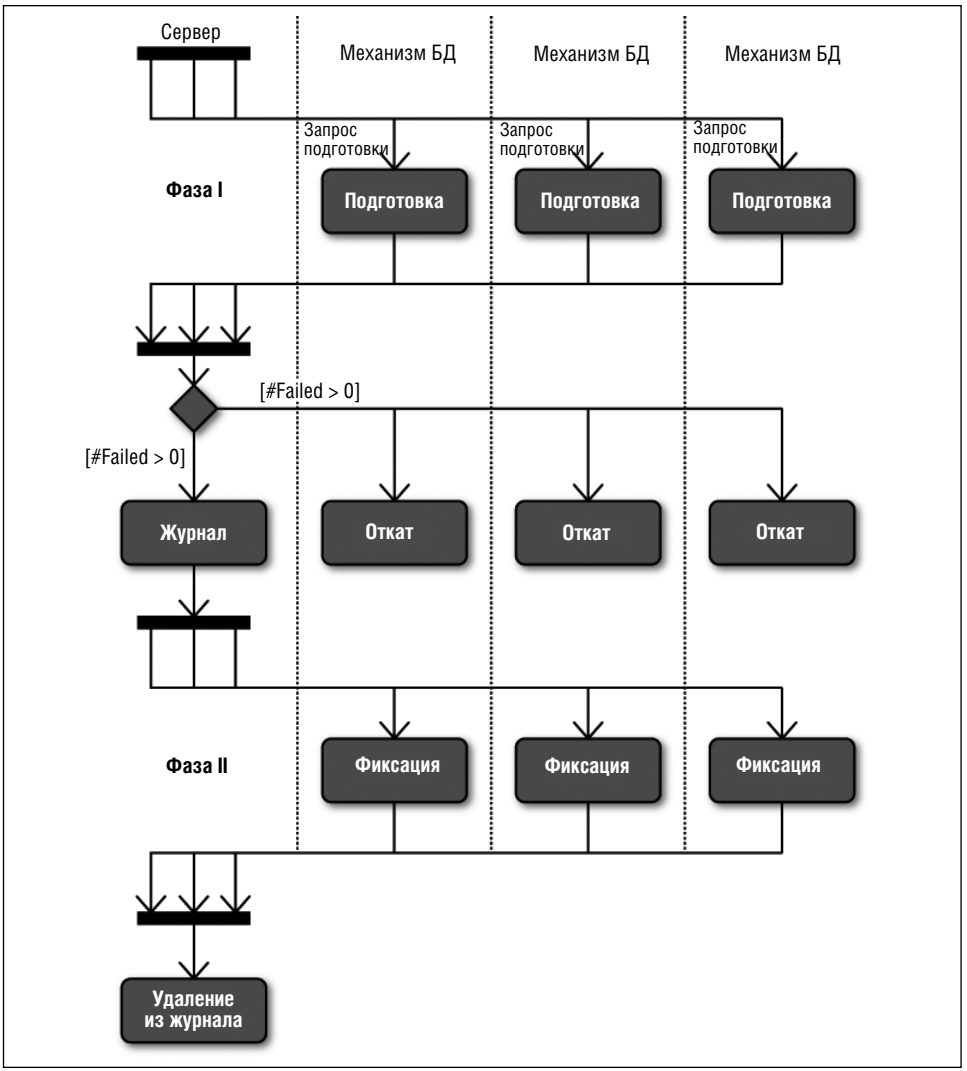


Рис. 3-6. Фиксация распределенной транзакции с использованием XA

На фазе 1 все механизмы БД получают запрос на подготовку к фиксации. Получив запрос, механизмы БД сбрасывают все данные, необходимые

для фиксации, безопасное хранилище, и отправляют подтверждение. Если какой-либо механизм БД не подтверждает готовность и не может зафиксировать транзакцию, происходит отмена фиксации и все механизмы БД получают указание откатить транзакцию.

После подтверждения всеми механизмами БД готовности к фиксации и до начала фазы 2 содержимое кэша транзакций записывается в двоичный журнал. В отличие от обычных транзакций, для которых записывается событие Query с оператором COMMIT, XA-транзакция оканчивается записью события Xid, содержащего XID-идентификатор.

На фазе 2 все механизмы БД, подготовившиеся к фиксации на фазе 1, получают запрос на выполнение и подтверждение фиксации транзакции. Важно понимать, что фиксация не может окончиться неудачей: еще на фазе 1 механизм БД подтвердил, что может зафиксировать транзакцию, и потому не имеет права сообщить о неудаче на фазе 2. Конечно, возможен отказ оборудования, но данные уже должны быть записаны в надежном месте, что позволит восстановить их после перезапуска сервера (об этом — ниже).

После фазы 2 диспетчер транзакций может отбросить любые общие ресурсы, двоичному журналу это не требуется, поэтому на данном этапе журнал не выполняет никаких действий, связанных с XA.

Если сбой происходит во время фиксации XA-транзакции, после запуска сервера будет выполнена процедура восстановления (рис. 3-7). При запуске сервер откроет последний двоичный журнал и проверит событие Format description. Если установлен флаг binlog-in-use, на сервере произошел сбой, и требуется восстановление расширенной архитектуры.

После запуска сервер просматривает двоичный журнал и выявляет XID-идентификаторы всех транзакций, читая из журнала события Xid. Далее все механизмы БД, работающие на сервере, получат запросы фиксации найденных транзакций. Для каждого XID-идентификатора из журнала будет проверена готовность транзакции к фиксации. Если транзакция готова, но не зафиксирована, будет выполнена фиксация этой транзакции. Если транзакция существует, но ее XID-идентификатора нет в журнале, эта транзакция была прервана отказом сервера, и ее нужно откатить.

Управление двоичным журналом

Вышеперечисленные события несут информацию о модификации данных на главном сервере. Есть и другие события, влияющие на репликацию, но не отражающие изменения данных на главном сервере. Например, остановка сервера может отразиться на репликации, поскольку во время простоя сервера файлы данных могут изменяться, например, в результате восстановления резервной копии или манипулирования файлами. Такие изменения не реплицируются, потому что сервер не работает и просто не «видит» их.

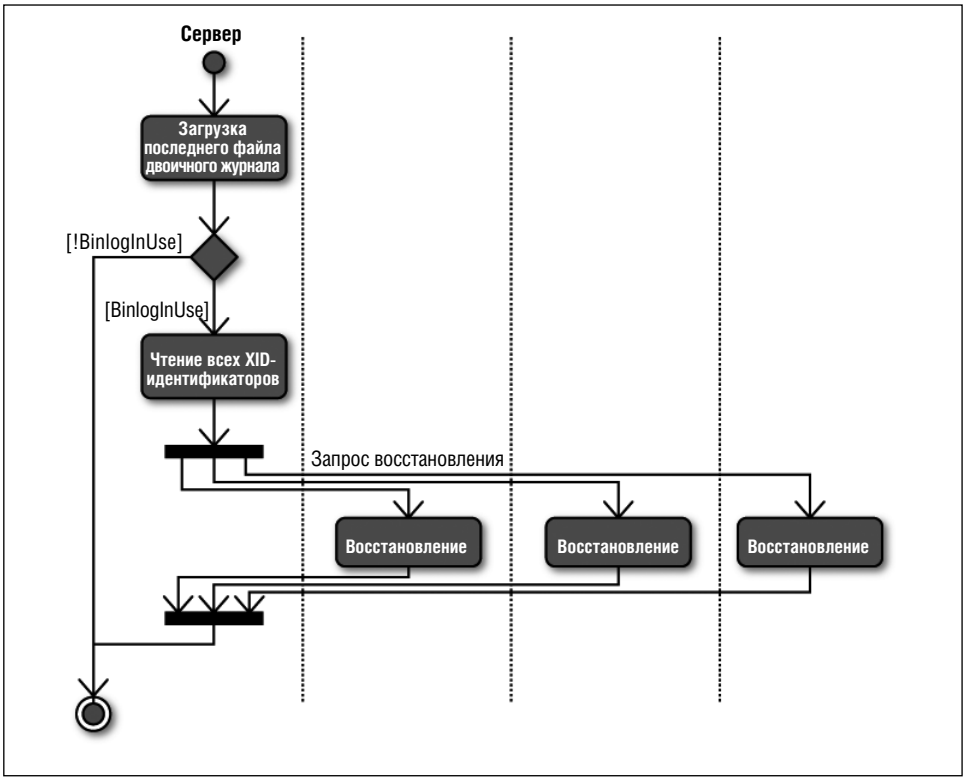


Рис. 3-7. Процедура восстановления расширенной архитектуры

События необходимы и для других целей. Поскольку двоичный журнал состоит из нескольких файлов, необходимо деление журнала на группы в подходящих местах. Для безопасности деления в журнал добавляют специальные события.

Отказоустойчивость двоичного журнала

Как показано выше, модификации двоичного журнала могут не полностью совпадать с таковыми в БД главного сервера. Важно сохранить согласованность БД и двоичного журнала в случае отказов. Другими словами, в БД не должно быть модификаций, отсутствующих в двоичном журнале, и наоборот.

Механизмы БД, не поддерживающие транзакции, исходно уязвимы для сбоев. Например, невозможно обеспечить согласованность двоичного журнала и таблиц MyISAM, поскольку MyISAM не поддерживает транзакции, и выполняет любые модификации до попытки регистрации соответствующих операторов.

В механизмах БД MySQL, поддерживающих транзакции, предусмотрены меры, исключающие потерю при отказе значительной порции двоичного журнала. Как сказано выше, события записываются в двоичный журнал до снятия блокировки с таблицы, но после модификации таблицы в БД. Если

отказ произойдет до снятия блокировки, сервер проверит, что все изменения, отраженные в двоичном журнале, внесены в файл таблицы на диске, и только потом разрешит фиксацию транзакции или вызова оператора. Эта процедура требует согласования со стандартными механизмами обеспечения согласованности файловой системы.

Поскольку обращение к диску обходится намного «дороже» обращения к оперативной памяти, в операционные системы встраивают поддержку кэширования файлов в специальной области основной памяти, называемой *страничным кэшем*. Содержимое страничного кэша сбрасывается на диск только при необходимости. Необходимость возникает, когда с диска требуется загрузить другую страницу, а кэш уже заполнен, либо по запросу программы, явно требующей сбросить страницы с содержимым файла на диск.

Как сказано в описании XA, по завершении фазы 1 все данные должны быть записаны в постоянную память, то есть на диск, на случай отказа. Это означает, что при фиксации каждой транзакции страничный кэш должен быть сброшен на диск. Эта операция может быть очень «дорогой» в терминах расхода системных ресурсов, и не всегда необходимой. Частотой сброса данных на диск управляют с помощью параметра sync-binlog, принимающего целочисленные значения. Если параметру присвоено значение 5, двоичный журнал будет сбрасываться на диск после фиксации каждой пятой транзакции или вызова. Значение по умолчанию равно 0. Это означает, что сервер не сбрасывает двоичный журнал на диск явно, а запись происходит по усмотрению операционной системы.

Для механизмов БД, поддерживающих XA, таких как InnoDB, достаточно установить параметр sync-binlog в 1, чтобы исключить потерю транзакций при обычных отказах (механизмы БД, не поддерживающие XA, могут потерять при этом максимум одну транзакцию).

Но при записи на диск каждой группы событий страдает производительность, и, как правило, очень сильно. Ни для кого не секрет, как медленно выполняется обращение к диску, и кэш-память служит для повышения производительности именно за счет исключения необходимости постоянной записи данных на диск. Если вас не смущает риск потери нескольких транзакций или операторов (например, если можно восстановить их вручную или эти данные не так важны), можно присваивать sync-binlog более высокие значения или вовсе оставить ему значения по умолчанию.

Ротация файлов двоичного журнала

MySQL регулярно создает новый файл двоичного журнала примерно через равные периоды. Запись в единственный файл невозможна как по техническим причинам, так и по соображениям удобства — операционные системы налагают ограничения на размер файлов. Как говорилось ранее, файл, в который сервер записывает текущие события, называется *активным*. Переключение на новый файл называется *ротацией*.

Ротацию вызывают:

- *Остановка сервера* Каждый раз, когда сервер останавливается, он начинает новый двоичный журнал. Ниже мы обсудим почему.
- *Достижение предельного размера файла двоичного журнала* Если файл двоичного журнала становится слишком большим, происходит автоматическая ротация. Размером файлов двоичного журнала управляют с использованием серверной переменной binlog-cache-size.
- *Явный сброс двоичного журнала на диск* Команда FLUSH LOGS сбрасывает все записи журнала на диск и создает новый файл для продолжения двоичного журнала. Это удобно при восстановлении состояния данных на заданный момент времени. Чтение открытого файла двоичного журнала может иметь непредсказуемые результаты, поэтому рекомендуется явно сбросить журнал на диск, прежде чем использовать его для восстановления.
- *Инцидент на сервере* Помимо полной остановки, на сервере возникают другие инциденты, вызывающие ротацию двоичного журнала. Иногда эти инциденты требуют вмешательства администратора, поскольку они чреваты «выпадением» данных из репликации. Администратору БД легче устранить инцидент, если сервер продолжит работу с новым файлом двоичного журнала.

Первое событие каждого файла двоичного журнала — это событие Format description, которое содержит версию сервера, сгенерировавшего файл наряду с информацией о содержимом и состоянии этого файла.

Особый интерес здесь представляют три элемента:

- *Флаг binlog-in-use* Поскольку отказ может произойти в момент записи в файл двоичного журнала, важно закрывать файл журнала штатным образом. Без этого администратор БД может воспроизвести поврежденный журнала на главном или подчиненном сервере, вызвать еще больше ошибок. Атрибутом, гарантирующим целостность файлов, является флаг binlog-in-use, который устанавливается при создании файла и снимается после записи закрывающего файл события (Rotate). По этому флагу любая программа увидит, правильно закрыт ли файл двоичного журнала.
- *Версия формата файла двоичного журнала* В разных версиях MySQL формат двоичного журнала менялся не раз и, несомненно, еще будет меняться. Разработчики увеличивают номер версии формата после внесения значительных изменений (особенно в общих заголовках), препятствующих чтению нового журнала прежними версиями сервера. Текущая версия формата, поддерживаемая MySQL 5.0 и выше, имеет номер 4. В поле версии формата указывается номер версии, и сервер, неспособный обработать файл этой версии, он просто отказывается его читать.
- *Версия сервера* Это строка, обозначающая версию сервера, создавшего файл. В этой главе использованы версии 5.1.37-1ubuntu5-log и 5.1.40-debug-log. Эта строка включает версию сервера MySQL и дополнитель-

ную информацию о сборке. В некоторых ситуациях эта информация поможет вам или разработчикам в диагностике и устранении мелких неполадок репликации между разными версиями сервера. Для отказоустойчивости ротация двоичного журнала выполняется заблаговременно, то есть предполагаемые к сбросу на диск данные записываются во временный файл, называемый *файл очистки индекса* (такое имя выбрано потому, что этот файл также используется для очистки файлов двоичного журнала, подробнее об этому — чуть ниже). Его имя содержит имя файла индекса, например, если имя файла индекса — *master-bin.index*, то имя файла очистки будет *master-bin.~rec~*. После создания нового файла двоичного журнала и добавления в файл индекса ссылки на него сервер удаляет файл очистки.

В случае отказа, сервер проверяет наличие файла очистки и, обнаружив его, сравнивает файл очистки с файлом индекса, чтобы выяснить, какие из предполагаемых к записи данных фактически были записаны.

В версиях MySQL до 5.1.43 при ротации и очистке файла двоичного журнала могут возникать «забытые» файлы, т. е. файлы, существующие в файловой системе, но не упомянутые в файле индекса. Такие файлы не удаляются автоматически, их приходится удалять вручную. «Забытые» файлы не вызывают проблем при репликации, но мешают работать. Вышеописанный механизм исключает их возникновение.

Инциденты

Под «инцидентами» понимают события, не изменяющие данные на сервере, но подлежащие записи в двоичный журнал из-за их потенциального влияния на репликацию. Большинство инцидентов не требует вмешательства администратора БД, пример — перезапуск сервера без модификации БД, но есть инциденты, требующие особых действий.

В настоящее время два типа инцидентов заносятся в двоичный журнал:

- *Stop* Показывает, что сервер был остановлен штатно. При крахе сервера такое событие не записывается, даже если сервер загрузится снова. Это событие записывается в старый файл двоичного журнала (после перезапуска сервера начинает новый файл) и содержит только общий заголовок и никакой другой информации.

  При воспроизведении двоичного журнала на подчиненном сервере все события Stop игнорируются. Обычно остановка сервера не требует особого внимания, и репликация продолжается, как обычно. Но если во время остановки было обновление версии сервера, это будет указано в следующем файле двоичного журнала. Сервер, не поддерживающий новую версию, не сможет прочитать этот журнал. Событие Stop не говорит о выпадении данных из репликации. Однако его стоит записывать, поскольку перед началом репликации возможно ручное восстановление резервных копий и внесение других изменений. Событие Stop поможет администратору БД обнаружить этот факт и скорректировать воспроизведение журналов.

- *Incident* Этот тип событий введен в версии 5.1 и представляет обобщенное событие-инцидент. В отличие от Stop, это событие содержит идентификатор типа инцидента. Свидетельствует о том, что сервер был вынужден выполнить действия, почти наверняка вызвавшие выпадение изменений из двоичного журнала.

  Например, в версии 5.1 инциденты регистрируются при перезагрузке БД или записи не защищенного транзакцией события, переполнившего журнал. MySQL Cluster генерирует это событие, когда один из узлов вынужденно перезагружает БД, чем вызывает риск рассинхронизации.

  Воспроизведение двоичного журнала на подчиненном сервере останавливается с ошибкой при встрече события Incident. Событие-перезагрузка в MySQL Cluster указывает на необходимость синхронизации кластера и, возможно, поиска событий, отсутствующих в двоичном журнале.

Очистка файла двоичного журнала

Если не стирать старые файлы двоичного журнала, они со временем накапливаются на сервере. Сервер может удалять старые файлы автоматически либо по команде оператора.

Чтобы сервер автоматически удалял старые файлы двоичного журнала, запишите в параметр expire-logs-days или одноименную серверную переменную срок хранения файлов двоичного журнала (в днях). Помните: как и все серверные переменные, эта настройка не сохраняется после перезапуска сервера. Чтобы сохранить ее, добавьте этот параметр в файл my.cnf сервера.

Для того, чтобы очистить файлы двоичного журнала вручную, используйте команду PURGE BINARY LOGS:

- *PURGE BINARY LOGS BEFORE дата* Эта команда удаляет все файлы старше заданной даты. Если заданная дата приходится на событие в середине файла журнала (так обычно и бывает), удаляются более старые файлы.
- *PURGE BINARY LOGS TO 'файл'* Эта команда удаляет все файлы, следующие по порядку перед заданным файлом. Другими словами, все файлы перед заданным в перечне, отображаемом командой SHOW MASTER LOGS, будут удалены, и заданный файл станет первым файлом двоичного журнала.

Старые файлы двоичного журнала удаляются при запуске сервера либо при ротации. Обнаружив файлы, подлежащие удалению (потому что они старше, чем задано expire-logs-days, или по команде PURGE BINARY LOGS), сервер сначала запишет содержимое удаляемых файлов в файле очистки индекса (например, *master-bin.~rec~*). Затем эти файлы удаляются из файловой системы, и в завершение удаляется файл очистки.

В случае отказа сервер сможет завершить удаление, сравнив содержимое файла очистки и файла индекса, и удалив ненужные файлы, оставшиеся из-за отказа. Как сказано выше, файл очистки используется и во время рота-

ции. Если отказ происходит до модификации файла индекса, новый файл двоичного журнала будет удален и заново создан.

Утилита mysqlbinlog

Одним из наиболее удобных утилит администратора является клиентская программа mysqlbinlog. Это небольшая программа для анализа содержимого файлов двоичного журнала и журнала ретрансляции (подробнее о них — в главе 6). Помимо локальных файлов, mysqlbinlog может читать файлы двоичного журнала и с удаленных серверов.

Кроме удобства для диагностики проблем с репликацией, он обеспечивает восстановление состояния данных на заданный момент времени (см. главу 2).

Как правило, инструмент mysqlbinlog извлекает из двоичного журнала SQL-код и отправляет его на сервер для исполнения, что используется для логической репликации. Для построчной репликации (см. главу 6) mysqlbinlog генерирует дополнительные данные. Поскольку эта глава посвящена логической репликации, в ней утилита используется с параметрами, отключающими вывод данных для построчной репликации.

О некоторых параметрах mysqlbinlog рассказывается в этом разделе, полный список см. в электронной справке по MySQL (*http://dev.mysql.com/doc/refman/5.1/en/mysqlbinlog.html*).

Примеры использования mysqlbinlog

Начнем с простого примера: создадим файл двоичного журнала и откроем его с помощью mysqlbinlog. Для этого запустим клиент, подключенный к главному серверу, выполним следующие команды и изучим, как они отразятся в двоичном журнале:

```
mysqld1> RESET MASTER;
Запрос OK, 0 строк affected (0.01 sec)

mysqld1> CREATE TABLE employee (
      ->    id INT AUTO_INCREMENT,
      ->    name CHAR(64) NOT NULL,
      ->    email CHAR(64),
      ->    password CHAR(64),
      ->    PRIMARY KEY (id)
      -> );
Query OK, 0 строк affected (0.00 сек)

mysqld1> SET @password = PASSWORD('xyzzy');
Query OK, 0 rows affected (0.00 sec)

mysqld1> INSERT INTO employee(name,email,password)
      ->    VALUES ('mats','mats@example.com',@password);
Запрок OK, 1 строк affected (0.01 сек)
```

```
mysqld1> SHOW BINARY LOGS;
+--------------------+-----------+
| Log_name           | File_size |
+--------------------+-----------+
| mysqld1-bin.000038 | 670       |
+--------------------+-----------+
1 row in set (0.00 sec)
```

Теперь с помощью mysqlbinlog откроем содержимое файла двоичного журнала *master-bin.000038* (результат после небольшого редактирования показан на лист. 3-15).

Лист. 3-15. Результат выполнения mysqlbinlog

```
$ sudo mysqlbinlog                          \
>     --short-form                          \
>     --force-if-open                       \
>     --base64-output=never                 \
>     /var/lib/mysql1/mysqld1-bin.000038
1  /*!40019 SET @@session.max_insert_delayed_threads=0*/;
2  /*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
3  DELIMITER /*!*/;
4  ROLLBACK/*!*/;
5  use test/*!*/;
6  SET TIMESTAMP=1264227693/*!*/;
7  SET @@session.pseudo_thread_id=999999999/*!*/;
8  SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1,
       @@session.unique_checks=1, @@session.autocommit=1/*!*/;
9  SET @@session.sql_mode=0/*!*/;
10 SET @@session.auto_increment_increment=1,
       @@session.auto_increment_offset=1/*!*/;
11 /*!\C latin1 *//*!*/;
12 SET @@session.character_set_client=8,@@session.collation_connection=8,
       @@session.collation_server=8/*!*/;
13 SET @@session.lc_time_names=0/*!*/;
14 SET @@session.collation_database=DEFAULT/*!*/;
15 CREATE TABLE employee (
16    id INT AUTO_INCREMENT,
17    name CHAR(64) NOT NULL,
18    email CHAR(64),
19    password CHAR(64),
20    PRIMARY KEY (id)
21 ) ENGINE=InnoDB
22 /*!*/;
23 SET TIMESTAMP=1264227693/*!*/;
24 BEGIN
25 /*!*/;
26 SET INSERT_ID=1/*!*/;
```

```
27 SET @`password`:=_latin1 0x2A31353141463... COLLATE `latin1_swedish_ci`/*!*/;
28 SET TIMESTAMP=1264227693/*!*/;
29 INSERT INTO employee(name,email,password)
30 VALUES ('mats','mats@example.com',@password)
31 /*!*/;
32 COMMIT/*!*/;
33 DELIMITER ;
34 # End of log file
35 ROLLBACK /* added by mysqlbinlog */;
36 /*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
```

Для получения этого вывода использовались следующие параметры:

- *--short-form* заставляет mysqlbinlog выводит только операторы SQL, исключая комментарии о событиях. Удобен, когда требуется только воспроизвести события. При диагностике, когда комментарии необходимы, этот параметр использовать не следует.
- *--force-if-open* отключает вывод предупреждения о том, что журнал еще открыт из-за сбоя или продолжающейся записи.
- *--base64-output=never* отключает вывод событий в кодировке base64. Если вывод в этой кодировке не отключен, mysqlbinlog выводит и событие Format description с указанием использованной кодировки (для логической репликации, когда это не требуется, используется этот параметр).

Строки 1–4 из лист. 3-15 выводятся каждый раз. В строке 3 отображается разделитель, не встречающийся более нигде в файле. Анализаторами, не поддерживающими разделители, разделитель интерпретируется как комментарий.

Оператор отката (строка 4) страхует от обработки журнала в транзакции, не завершенной на клиенте до начала анализа журнала.

Теперь перейдем к концу вывода: строки 33–35 восстанавливают исходные значения параметров, измененных строками 1–4, и откатывают любую незавершенную транзакцию. Это необходимо, если файл двоичного журнала открыт в середине транзакции, чтобы предотвратить включение в транзакцию SQL-кода из журнала.

Оператор use в строке 5 выводится при модификации БД. Несмотря на то, что в двоичном журнале текущая БД указывается перед каждым SQL-оператором, mysqlbinlog показывает только изменения в текущей БД. Оператор use всегда открывает новое событие.

Первая строка, присутствующая в выводе каждого события, — SET TIMESTAMP (строки 6 и 23). Этот оператор отмечает время начала исполнения события (в секундах с начала отсчета).

Строки 8–14 содержат общие настройки; как и use в строке 5, они выводятся только для первого события, а также всякий раз, когда изменяются их значения.

Поскольку оператор INSERT (строки 29–30) вставляет строку в таблицу с полем, для которого включено автоприращение с пользовательской переменной, выше устанавливается значение сеансовой переменной INSERT_ID (строка 26) и пользовательской переменной (строка 27). Это происходит в результате событий Intvar и User_var в двоичном журнале.

Если параметр --short-form не указан, то каждое событие в выводе предваряется комментариями, начинающимися знаком «решетка», # (см. лист. 3-16).

Лист. 3-16. Комментарии в выводе mysqlbinlog

```
$ sudo mysqlbinlog                      \
>     --force-if-open                   \
>     --base64-output=never             \
>     /var/lib/mysql1/mysqld1-bin.000038
   .
   .
   .
1  # at 386
2  #100123 7:21:33 server id 1 end_log_pos 414 Intvar
3  SET INSERT_ID=1/*!*/;
4  # at 414
5  #100123 7:21:33 server id 1 end_log_pos 496 User_var
6  SET @`password`:=_latin1 0x2A313531...838 COLLATE `latin1_swedish_ci`/*!*/;
7  # at 496
8  #100123 7:21:33 server id 1 end_log_pos 643
   Query thread_id=6 exec_time=0 error_code=0
9  SET TIMESTAMP=1264227693/*!*/;
10 INSERT INTO employee(name,email,password)
11 VALUES ('mats','mats@example.com',@password)
12 /*!*/;
13 # at 643
14 #100123 7:21:33 server id 1 end_log_pos 670 Xid = 218
15 COMMIT/*!*/;
16 DELIMITER ;
17 # End of log file
18 ROLLBACK /* added by mysqlbinlog */;
19 /*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
```

Первая строка комментария сообщает байтовое смещение события, а вторая — информацию о событии. Рассмотрим, например, строку оператора INSERT:

```
# at 496
#100123 7:21:33 server id 1 end_log_pos 643 Query thread_id=6
   exec_time=0    error_code=0
```

Части строк комментариев имеют следующие значения:

- *at 496* Байтовое смещение начала (первого байта) события.

- *100123 7:21:33* Время и дата события (начала исполнения запроса или записи события в двоичный журнал).
- *server_id 1* Идентификатор сервера, сгенерировавшего событие. Служит для установки сеансовой переменной pseudo_thread_id. Соответствующий оператор выводится, если событие является специфичным для потока с новым идентификатором сервера.
- *end_log_pos 643* Байтовое смещение следующего события. Разность смещений текущего и следующего события дает длину текущего события.
- *Query* Тип события. В лист. 3-16 показано несколько типов событий: User_var, Intvar и Xid. После них следуют поля, специфичные для событий. У события Query два дополнительных поля:
- *vthread_id=6* Это идентификатор потока, исполняющего событие. Используется для обработки специфичных для потока запросов, например обращений к временным таблицам.
- *vexec_time=0* Время выполнения запроса в секундах.

В примерах 3-15 и 3-16 открывается один файл, но mysqlbinlog позволяет обрабатывать группы файлов.

Содержимое файлов выводится в том порядке, в каком эти файлы указаны при вызове утилиты, и без проверки последовательности событий `Rotate`. За подбор правильных файлов журнала отвечает пользователь.

Схема именования файлов двоичного журнала облегчает групповую обработку файлов с помощью mysqlbinlog, например при использовании подстановочного знака *. Но вот что происходит при переходе счетчика файлов двоичного журнала, который используется в качестве расширения, с 999999 на 1000000:

```
$ ls mysqld1-bin.[0-9]*
mysqld1-bin.000007 mysqld1-bin.000011  mysqld1-bin.000039
mysqld1-bin.000008 mysqld1-bin.000035  mysqld1-bin.1000000
mysqld1-bin.000009 mysqld1-bin.000037  mysqld1-bin.999998
mysqld1-bin.000010 mysqld1-bin.000038  mysqld1-bin.999999
```

Видно, что последний файл двоичного журнала обрабатывается до предыдущих, поэтому внимательно проверяйте имена файлов перед использованием подстановочных знаков.

Как правило, файлы двоичного журнала велики, и выводить содержимое файла целиком обычно не требуется. Вместо этого можно выбрать определенный диапазон событий с помощью следующих параметров:

- *start-position=смещение* смещение первого события, подлежащего выводу. Если задано несколько файлов двоичного журнала, это смещение действует для *первого* файла из группы. Если смещение задано неверно, mysqlbinlog все равно будет интерпретировать файл, генерируя бесполезный набор знаков.
- *stop-position=bytepos* смещение последнего события, подлежащего выводу. Если смещение указывает за пределы файла, последним выводится событие, ближайшее к заданному смещению. Если указана обработка группы файлов, это смещение действует для *последнего* файла группы.

- *start-datetime=datetime* пропускает все события, записанные до временной отметки, заданной параметром datetime. Этот параметр действует на всю группу файлов, если таковая задана, но не проверяет реальную хронологию событий по их временным отметкам.
- *stop-datetime=datetime* пропускает все события, записанные после временной отметки, заданной параметром datetime, включительно. Т. е., если для параметра задано значение 2010-01-24 07:58:32, то событие с такой временной отметкой *не* выводится.

Заметьте, что в журнале проставляется время начала исполнения оператора, но порядок событий в двоичном журнале определяется временем их фиксации, т. е. не исключено, что события с более поздней временной отметкой окажутся впереди событий с более ранней отметкой. Поскольку mysqlbinlog останавливает вывод на первом же событии вне заданного временного диапазона, возможны пропуски подходящих под заданные критерии событий.

Чтение файлов с удаленных серверов

Утилита mysqlbinlog может читать файлы двоичного журнала не только с локального, но и с удаленного сервера. Это делается с помощью того же механизма, который используется для связи подчиненных серверов с главным. В некоторых случаях это удобно, так как не требует наличия на машине учетной записи оболочки для чтения файлов двоичного журнала — достаточно полномочий REPLICATION SLAVE.

Для чтения двоичного журнала удаленного сервера включите параметр --read-from-remote-server, имена хоста и пользователя для подключения к серверу, порт (если он отличается от порта по умолчанию) и пароль. При чтении с удаленного сервера достаточно указать имя файла двоичного журнала, полный путь не обязателен.

Для чтения события Query из примера 3-16 с удаленного сервера используется такая команда (сервер запросит пароль, но при вводе пароль не отображается):

```
$ sudo mysqlbinlog
>  --read-from-remote-server
>  --host=master.example.com
>  --base64-output=never
>  --user=repl_user --password
>  --start-position=386 --stop-position=643
>  mysqld1-bin.000038
Enter password:
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
#  at 386
```

```
#100123 7:21:33 server id 1 end_log_pos 0 Start: binlog v 4,
   server v 5.1.37-1ubuntu5-log created 100123 7:21:33
#  at 386
#100123 7:21:33 server id 1 end_log_pos 414 Intvar
SET INSERT_ID=1/*!*/;
#  at 414
#100123 7:21:33 server id 1 end_log_pos 496 User_var
SET @`password`:=_latin1 0x2A3135314146364...38 COLLATE `latin1_swedish_ci`/*!*/;
#  at 496
#100123 7:21:33 server id 1 end_log_pos 643 Query thread_id=6
   exec_time=0    error_code=0
use test/*!*/;
SET TIMESTAMP=1264227693/*!*/;
SET @@session.pseudo_thread_id=6/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1,
    @@session.unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=0/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C latin1 *//*!*/;
SET @@session.character_set_client=8, @@session.collation_connection=8,
    @@session.collation_server=8/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
INSERT INTO employee(name,email,password)
   VALUES ('mats','mats@example.com',@password)
/*!*/;
DELIMITER ;
#  End of log file
ROLLBACK /* added by mysqlbinlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
```

Интерпретация событий

Иногда стандартной информации, выводимой mysqlbinlog, недостаточно для выявления проблемы, и требуется вникнуть в детали события, проанализировав его содержимое. Для этого служит параметр --hexdump, заставляющий mysqlbin log выдать составляющие событие байты.

Перед анализом кода событий следует знать общий формат данных в двоичном журнале:

- *Целочисленные данные* Целочисленные поля в двоичном журнале выводятся в обратном порядке байтов, поэтому их читают справа налево. Например, 32-битовый блок 03 01 00 00 обозначает шестнадцатеричное число 103.
- *Строковые данные* Вместе со строками обычно записывается их длина, строка оканчивается нулем. Иногда длина указывается в начале строки или в постзаголовке.

Ниже описаны наиболее распространенные события, но описание формата всех событий не поместится в этой книге. В электронной справке MySQL Internals guide (*http://forge.mysql.com/wiki/MySQL_Internals_Binary_Log*) вы найдете полный список всех событий и их полей.

Наиболее распространенное из всех событий – Query, поэтому начнем с него. На лист. 3-17 показан байтовый код такого события.

Лист. 3-17. Выходные данные с использованием параметры --hexdump

```
   $ sudo mysqlbinlog                  \
   >  --force-if-open                  \
   >  --hexdump                        \
   >  --base64-output=never            \
   >  /var/lib/mysql1/mysqld1-bin.000038
      .
      .
      .
1  # at 496
2  #100123 7:21:33 server id 1 end_log_pos 643
3  # Position Timestamp Type Master ID Size Master Pos Flags
4  # 1f0 6d 95 5a 4b 02 01 00 00 00 93 00 00 00 83 02 00 00 10 00
5  # 203 06 00 00 00 00 00 00 00 04 00 00 1a 00 00 00 40 |               |
6  # 213 00 00 01 00 00 00 00 00 00 00 00 06 03 73 74 64 |            std|
7  # 223 04 08 00 08 00 08 00 74 65 73 74 00 49 4e 53 45 |      test.INSE|
8  # 233 52 54 20 49 4e 54 4f 20 75 73 65 72 28 6e 61 6d |RT.INTO.employee|
9  # 243 65 2c 65 6d 61 69 6c 2c 70 61 73 73 77 6f 72 64 |.name.email.pass|
10 # 253 29 0a 20 20 56 41 4c 55 45 53 20 28 27 6d 61 74 |word....VALUES..|
11 # 263 73 27 2c 27 6d 61 74 73 40 65 78 61 6d 70 6c 65 |.mats...mats.exa|
12 # 273 2e 63 6f 6d 27 2c 40 70 61 73 73 77 6f 72 64 29 |mple.com...passw|
13 # 283 6f 72 64 29                                     |ord.|
14 # Query thread_id=6 exec_time=0 error_code=0
   SET TIMESTAMP=1264227693/*!*/;
   INSERT INTO employee(name,email,password)
      VALUES ('mats','mats@example.com',@password)
```

Первые две строки и строка 13 являются комментариями, о которых говорилось выше. Обратите внимание, что при использовании параметра --hexdump общая и специфичная для события информация делится на две строки, а в обычном выводе она объединяется.

Строки 3 и 4 содержат общий заголовок:

- *Timestamp* Временная отметка события (целое число в «остроконечном» формате (т. е. значение начинается с с младшего байта)).
- *Type* байт, обозначающий тип события. О типах событий в MySQL 5.1.41 и выше см. в руководстве MySQL Internals guide (*http://forge.mysql.com/wiki/MySQL_Internals_Binary_Log*).
- *Master ID* Идентификатор сервера, записавшего событие (целое число). Для события, показанного на лист. 3-17, идентификатор сервера равен 1.

- *Size* Размер события в байтах (целое число).
- *Master Pos* То же, что end_log_pos: смещение следующего события.
- *Flags* 16-битная серия общих флагов события. Не используется кроме флага binlog-in-use. В примере выше флаг binlog-in-use установлен, что означает, что двоичный журнал не закрыт (здесь — потому что перед вызовом mysqlbinlog не выполнен сброс записей).

После общего заголовка идет постзаголовок и тело события. Как уже сказано, полное описание событий выходит за рамки этой книги, но мы расскажем о наиболее важных и востребованных — событиях Query и Format_description.

Тело и постзаголовок события Query

Событие Query, несомненно, является наиболее востребованным и сложным. Отчасти это объясняется тем, что оно несет много информации о контексте времени выполнения оператора. Как уже было показано, целочисленные и пользовательские переменные, а также исходные значения для псевдослучайных функций регистрируются в специализированных событиях, но полезно знать и способы представления другой информации в этом событии.

Постзаголовок для события Query состоит из пяти полей. Помните, что они имеют фиксированный размер и длина постзаголовка определена в событии Format description файла двоичного журнала. Это означает, что в более поздних версиях MySQL могут вводиться дополнительные поля.

- *Идентификатор* Четырехбайтовое целое число без знака, идентифицирующее поток, исполнивший оператор. Не всегда требуется для корректного выполнения оператора, но всегда записывается в событие.
- *Время выполнения* Число секунд с момента начала выполнения до момента записи запроса в двоичный журнал (четырехбайтовое целое число без знака).
- *Длина имени БД* Длина имени БД (однобайтовое целое число без знака). Само имя БД хранится в теле события, но длина имени — здесь.
- *Код ошибки* Код ошибки, возникающий в результате выполнения оператора (двухбайтовое целое число без знака). Используется, поскольку иногда в двоичный журнал необходимо записывать даже операторы, исполнение которых окончилось неудачей.
- *Длина переменной состояния* Длина блока в теле события, хранящего переменные состояния (двухбайтовое целое число без знака). Иногда используется для хранения различных переменных состояния, таких как SQL_MODE.

Тело события состоит из следующих полей разной длины:

- *Переменные состояния* Набор переменных со сведениями о состоянии. Каждая переменная представлена целым числом, за которым следует ее значение. Интерпретация и длина значения переменной индивидуальна.

Переменные состояния записываются не всегда, а только при необходимости. Вот некоторые переменных состояния:

- *Q_SQL_MODE_CODE* значение SQL_MODE, используемое при выполнении оператора.
- *Q_AUTO_INCREMENT* хранит значения auto_increment_increment и auto_increment_offset для оператора, отличные от значения по умолчанию (–1).
- *Q_CHARSET* хранит код набора символов и схем сопоставления, назначенных соединению и серверу во время исполнения оператора.

- *Текущая БД* Имя текущей БД (строка, заканчивающаяся null-знаком). Обратите внимание, что длина имени базы данных определена в постзаголовке.
- *Текст оператора* Код оператора. Длина кода оператора может быть вычислена по информации из общего заголовка и постзаголовка. Обычно идентичен исходному коду исполненного оператора, но иногда модифицируется перед сохранением в двоичном журнале (см. выше примеры с триггерами, хранимыми процедурами и секцией DEFINER).

Постзаголовок и тело события Format_description

Событие Format_description хранит важные сведения о формате файла двоичного журнала, событий и сервере. Это поле должно интерпретироваться любыми версиями сервера даже после изменения формата двоичного журнала, поэтому его изменения строго регламентированы.

Одним из наиболее важных является ограничение длины общего заголовка событий Format_description и Rotate — 19 байт. Это означает, что описание события невозможно расширить, добавив поля в общем заголовке.

Постзаголовок и тело события Format_description содержат следующие поля:

- *Версия файла двоичного журнала* Версия формата двоичного журнала, используемая этим файлом (в MySQL 5.0 и выше она равна 4).
- *Строка версии сервера* 50-байтовая строка, хранящая информацию о версии сервера. Обычно это трехчастный номер версии и сведения о сборке, например 5.1.37-1ubuntu5-log.
- *Время создания* Четырехбайтное целое число секунд с момента начала отсчета — времени записи первого файла двоичного журнала с момента запуска сервера. В более поздних файлах двоичного журнала это поле заполнено нулями.

  Данная схема позволяет подчиненному серверу перезапуск главного сервера, в случае которого подчиненный сервер должен сбросить сведения о состоянии и временные отметки, т. е. закрыть все транзакции и удалить все временные таблицы.
- *Длина общего заголовка* Длина общего для всех событий в файле двоичного журнала заголовка *кроме* Format_description и Rotate. Как сказано

ранее, длина общего заголовка для событий Format_description и Rotate жестко прописана и составляет 19 байтов.

- *Длина постзаголовка* Единственное поле переменной длины в описании события Format_description. Содержит массив однобайтных целых значений длин постзаголовков всех событий в файле двоичного журнала). Значение 255 зарезервировано для длины поля, поэтому максимальная длина постзаголовка составляет 254 байта.

Параметры и переменные двоичного журнала

Параметры и переменные позволяют конфигурировать множество аспектов ведения двоичных журналов. Некоторые параметры управляют именами файлов двоичного журнала и индекса. Большинство параметров можно настраивать, в том числе через серверные переменные. Некоторые параметры упоминались раньше, но здесь мы остановимся на них подробнее:

- *expire-log-days=число_дней* Время (в днях) хранения файлов двоичного журнала. Более старые файлы будут стерты при ротации двоичного журнала или перезапуске сервера. По умолчанию равно 0 (файлы двоичного журнала не удаляются).
- *log-bin*[*=базовое_имя*] Двоичный журнал включается посредством добавления параметра log-bin в файл *my.cnf* (см. главу 2). Помимо включения ведения журнала, этот параметр задает базовое имя файлов двоичного журнала (т. е. часть имени файла до точки). Если указано расширение, оно удаляется при формировании базового имени файлов двоичного журнала.

  Если параметр указан без базового имени, берется базовое имя по умолчанию host-bin, где host — базовое имя (имя файла без пути и расширения), заданное параметром pid-file, обычно как *gethostname(2)*. Например, если pid-file — это */usr/run/mysql/master.pid*, то по умолчанию файлам двоичного журнала будут назначаться имена *master-bin.000001*, *master-bin.000002*, и т. д.

  Поскольку значение по умолчанию параметра pid-file включает в себя имя хоста, настоятельно рекомендуем задавать значение параметру log-bin. В противном случае имена файлов двоичного журнала будут изменяться при изменении имени хоста (если pid-file не задан явно).
- *log-bin-index*[*=имя_файла*] Имя файла индекса. Удобно, если вы хотите поместить файл индекса в каталог, отличный от каталога по умолчанию.

  Значение по умолчанию — то же, что и у базового имени, заданного log-bin. Например, если базовое имя файлов двоичного журнала — это *master-bin*, то файл индекса будет называться *master-bin.index*.

  Как и в случае параметра log-bin, для генерации имени файла индекса ис-

пользуется имя хоста. Это означает, при переименовании хоста репликация прервется. По этой причине настоятельно рекомендуется определять этот параметр.

- *log-bin-trust-function-creators* Хранимые функции позволяют читать произвольные данные и как угодно манипулировать ими на подчиненном сервере. По этой причине создание хранимых функций требует привилегий SUPER. Тем не менее, хранимые функции полезны во многих обстоятельствах, и администраторы БД часто доверяют всем пользователям с привилегией CREATE ROUTINE создание хранимых функций. По этой причине имеется возможность отключить требование привилегии SUPER для создания хранимых функций (но CREATE ROUTINE попрежнему требуется).
- *binlog-cache-size=число_байтов* Размер части кэша транзакций в оперативной памяти. Данные, которые выходят за рамки этого предела, сбрасываются на диск.

  Частая запись на диск вызывает проблемы с производительностью, поэтому, увеличив значение этого параметра, можно поднять производительность, если обрабатывается множество объемных транзакций.

  Обратите внимание, что выделение очень большого буфера — не слишком удачная идея, поскольку остальным компонентам сервера достанется меньше памяти, что также вызовет снижение производительности.
- *max-binlog-cache-size=число_байтов* Этот параметр ограничивает размер транзакций в двоичном журнале. Большие транзакции могут заблокировать журнал на длительное время и вызвать блокировку других потоков на двоичном журнале, резко снизив производительность. Если размер транзакции превышает заданный, исполнение оператора будет прервано с ошибкой.
- *max-binlog-size=число_байтов* Указывает размер файлов двоичного журнала. Превышение этого размера при записи вызывает ротацию и запись продолжается в новый, пустой файл двоичного журнала.

  Обратите внимание, что если транзакция превысит максимальный размер файла двоичного журнала, после ротации повторится попытка записи транзакции целиком в новый файл, что снова вызовет превышение максимального размера файла. Это происходит потому, что транзакции никогда не делятся между файлами двоичного журнала.
- *sync-binlog=period* Указывает частоту сброса двоичного журнала на диск с использованием *fdatasync(2)*. Указывается число фиксированных транзакций для каждого реального вызова *fdatasync(2)*. Например, если задано значение 1, *fdatasync(2)* будет вызываться для каждой зафиксированной транзакции, а если задано значение 10, то *fdatasync(2)* будет вызываться через каждые 10 фиксаций.

  Значение, равное нулю, означает отмену вызовов *fdatasync(2)*, т. е. сервер доверит сброс двоичного журнала на диск операционной системе.

- *read-only* Запрет всем клиентским потокам, за исключением потоков подчиненного сервера и пользователей с привилегиями SUPER, обновления данных на сервере. Это полезно для подчиненных серверов для обеспечения продолжения репликации без угрозы повреждения данных клиентами, подключенными к подчиненному серверу.

Заключение

В этой главе описано множество аспектов двоичного журнала, включая устройство, работу с журналом и управление им. Материал этой главы закладывает основы для понимания механизмов двоичного журнала и его важной роли в регистрации модификации данных.

Джоэл открыл письмо от начальника, присланное с пустой строкой темы. «Терпеть не могу, когда так делают», — подумал он. Сообщения г-на Самерсона не отличались от его заданий — прямые и лаконичные. В сообщении было написано: «Спасибо за восстановление данных отдела маркетинга. Жду отчет завтра к утру, можно по электронной почте».

Джоэл пожал плечами и открыл новое сообщение, автор которого потрудился указать тему. Джоэл задумался, насколько подробно надо все изложить и надо ли ему писать все, что он узнал о двоичном журнале и утилите mysqlbinlog. Поразмыслив с минуту, он описал все так подробно, как только смог. «Все равно прикажут урезать текст до короткого списка», — подумал Джоэл, и составил «выжимку» из двух предложений и нескольких тезисов. Скопировав все это в новое письмо, он отослал сообщение боссу. «Может, стоит все-таки сохранить первоначальный текст — вдруг он потребует подробный доклад?..» — все еще колебался Джоэл.

ГЛАВА 4

Роль репликации в обеспечении высокой доступности

Джоэл слушал iPod и не заметил, как начальник подошел к его столу.

— Простите, сэр, — он снял наушники.

— Все в порядке, Джоэл, — улыбнулся Саммерсон. — Нужно наладить мониторинг реплицируемых серверов, чтобы исключить потерю данных и сократить простои. Разработчики стали жаловаться: говорят, система неудобная. С разработчиками можно договориться, но люди из техподдержки говорят, что на восстановление после сбоев уходит много времени. Так что это — твоя задача номер один.

— Хорошо, посмотрим, что у нас с балансировкой нагрузки, и как оптимизировать восстановление с помощью репликации, — кивнул Джоэл.

— Отлично. Подготовь мне отчет о том, что нужно для решения проблемы.

Джоэл проводил начальника глазами. «Эх... А что пишут там про высокую доступность?» — подумал он, открывая любимую книгу по MySQL.

Казалось бы, что нужно для обеспечения высокой доступности системы: приобрести дорогое, известное своей надежностью, оборудование и хорошие ИБП на случай перебоя в электроснабжении.

На самом деле, не все так просто. Чтобы система в действительности была доступна всегда, следует тщательно спланировать все чрезвычайные ситуации, обеспечив избыточность компонентов, уязвимых для сбоев. Истинную высокую доступность — когда работоспособность системы сохраняется даже в самых неожиданных обстоятельствах — реализовать очень сложно и дорого.

Принципы достижения высокой доступности довольно просты. Для этого необходимо три вещи:

- *Избыточность* На случай отказа компонента необходимо иметь для него замену. Замена может находиться в резерве, либо быть частью существующего развертывания.
- *Планы действия в чрезвычайных ситуациях* Сбой компонента не должен застать вас врасплох. Действия зависят от компонента и обстоятельств его выхода из строя.
- *Процедура* В случае сбоя компонента, вы должны его найти и применить готовый план быстро и эффективно.

Наличие в системе единой точки отказа, т. е. компонента, сбой которого «обрушивает» всю систему, серьезно осложняет достижение высокой доступности. Значит, одна из первоочередных задач — найти единые точки отказов и обеспечить избыточность для них.

Избыточность

Чтобы узнать, где необходима избыточность, нужно найти в развертывании все потенциальные точки отказов. Ведь это только сказать легко — на деле это требует немалого труда и, кстати, воображения. Коммутаторы, маршрутизаторы, сетевые карты и даже кабели — все это потенциальные единые точки отказа. Возможно, вас это и не касается, но не менее важные точки — источники питания и прочая матчасть. А службы, необходимые для поддержки развертывания? Представьте, что все функции управления сетью доступны через единый веб-интерфейс. Что, если некоторые сбои умеет устранять только один человек из всего обслуживающего персонала?

Найти точки отказа не означает взять на себя обязанность по их устранению. Иногда, это просто невозможно по экономическим, техническим или географическим причинам. Однако знание этих точек помогает в планировании действий в экстренных ситуациях.

Вот лишь некоторые аспекты, которые необходимо принять во внимание или, по меньшей мере, выработать по ним сознательное решение: стоимость резервирования компонентов, вероятность сбоя различных компонентов, время, требующееся на замену компонента, риск, связанный с ремонтом компонента. Если на ремонт компонента уйдет неделя, а вы в это время работаете на резерве, имеющем единую точку отказа, есть риск, что резервный компонент тоже даст сбой, а это может оказаться неприемлемо.

Определив позиции, по которым необходима избыточность, вам нужно выбрать один из двух базисных принципов: держать ли «под рукой» дубликат каждого компонента — в случае сбоя основного компонента, он быстро возьмет на себя его функции; либо иметь дополнительные мощности в системе, которые позволят удержать нагрузку при отказе компонента. Совсем не обязательно применять один и тот же принцип для всех случаев. Их можно комбинировать: дублировать одни компоненты, и использовать дополнительные мощности для других.

На первый взгляд, проще всего дублировать компоненты, но это дорого. Резервный компонент должен находиться поблизости, постоянно синхронизируясь с основным компонентом. Преимущества дублирования компонентов в том, что во время перехода не происходит снижение производительности, а сам переход на резервный компонент, как правило, выполняется быстрее, чем реструктуризация системы, необходимая, если проблемы решена путем создания резервных мощностей.

Создание резервной мощности позволяет использовать все компоненты для нужд предприятия и обрабатывать пиковые нагрузки. При выходе

компонента из строя нужно реструктурировать систему и задействовать все оставшиеся компоненты. При этом важно, чтобы доступная в любой мощность превышала типичные потребности.

Чтобы понять зачем, рассмотрим простой случай: главный сервер, обрабатывающий запросы на запись, — на самом деле, для избыточности серверов должно быть два, — и набор подключенных к нему подчиненных серверов, единственная задача которых — обслуживать запросы на чтение.

При выходе из строя одного из подчиненных серверов система будет отвечать, но ее мощность уменьшится. Если из 10 подчиненных серверов каждый загружен на 50%, отказ одного из них увеличит нагрузку на остальные до 55%, что не проблема. Но если подчиненные серверы используют 95% ресурса и один из них выйдет из строя, то на каждый сервер «навалится» по 105% от первоначальной нагрузки, а это неприемлемо: ресурс системы по выполнению операций чтения уменьшится, а время отклика увеличится.

Не планируйте потерю только одного сервера: необходимо учитывать возможность потери сразу нескольких серверов и быть готовым к такой ситуации. Возвращаясь к предыдущему примеру, даже если каждый сервер использует 80% ресурса, система справится с потерей одного сервера. Однако потеря двух серверов приведет к увеличению нагрузки на каждый из оставшихся серверов до 100%, не оставляя вам шансов при внезапном росте трафика. Если такое случается раз в год, то все поправимо, но вы же знаете, как часто бывает именно так!

В табл. 4-1 показана зависимость вероятности потери 1, 2 или 3 из 100 серверов от вероятности потери одного сервера. Видно, что при вероятности потери одного сервера 1% риск потерять следом три и более серверов равен 16%. Если не обращать на это внимания, то, в случае сбоя, все может закончиться весьма плачевно.

Вероятность для случайной переменной X (число потерянных серверов) рассчитывается по формуле

$$P(X \geq k) = \binom{n}{k} p^k$$

Табл. 4-1. Вероятность потери серверов

Вероятность потери одного сервера	1	2	3
1,00%	100,00%	49,50%	16,17%
0,50%	50,00%	12,38%	2,02%
0,10%	10,00%	0,50%	0,02%

Во избежание подобной ситуации нужно провести подробный мониторинг системы, чтобы определить нагрузки, опытным путем установить производительность системы и рассчитать момент, когда начнет расти время отклика.

Планирование

Одной избыточности недостаточно: на случай отказа компонентов нужны планы действий. В предыдущем примере выход из строя подчиненного сервера не представляет угрозы — новые подключения будут перенаправлены на работающие серверы, но остаются вопросы:

- Что произойдет с существующими подключениями? Не годится просто разорвать соединение и вернуть пользователю сообщение об ошибке. Как правило, пользователь и БД связаны на прикладном уровне, поэтому попытка повторного запроса на сервер должна выполняться на прикладном уровне.
- Что будет в случае отказа главного сервера? В предыдущем примере из строя выходили только подчиненные серверы, но главные серверы тоже ломаются. Если вы обеспечили избыточность, создав еще один главный сервер (о том, как это сделать, см. далее в этой главе), у вас должен быть план перевода всех подчиненных серверов на новый главный сервер.

В этой главе рассмотрены некоторые способы и топологии для различных ситуаций со сбоями серверов MySQL. В основном, для сервера отрабатывается три сценария: отказ главного сервера, отказ подчиненного сервера и отказ сервера ретрансляции. Отказ подчиненного сервера — это сбой подчиненного сервера, используемого для масштабирования операций чтения. Подчиненные серверы, выполняющие роль главных, — это подчиненные серверы ретрансляции, требующие особого рассмотрения. Отказ главного сервера — наиболее тяжелый, и должен быть устранен быстро: пока не восстановлен главный сервер, все развертывание будет недоступным.

Отказ подчиненного сервера

Считается, что сбой на подчиненном сервере исправить легче всего. Поскольку подчиненные серверы применяются только для обработки запросов чтения, важно информировать подсистему балансировки нагрузки об отказе сервера, а уж она перенаправит новые запросы на рабочие серверы. Подчиненных серверов должно быть достаточно, чтобы справится с «провалом» производительности, однако одиночный отказ подчиненного сервера, как правило, не влияет на топологию репликации и для его обработки не требуется создания специальных топологий.

Когда на подчиненном сервере происходит сбой, некоторые отправленные на него запросы непременно останутся в ожидании ответа. После того как запросы вернут сообщения об ошибке, обусловленной «потерей» сервера, запросы должны быть отправлены еще раз, уже на работающий сервер.

Отказ главного сервера

Если главный сервер выходит из строя, его необходимо заменить, и быстро. В момент сбоя главного сервера будут отменены все запросы записи, поэто-

му первым делом нужно подключить новый главный и направить на него всех клиентов. Поломка главного сервера означает, что все подчиненные серверы остались без главного, и что на них есть устаревшие данные, но они все еще в работе и способны отвечать на запросы чтения.

Тем не менее, некоторые запросы, ожидающие прихода изменений на подчиненный сервер, могут быть заблокированы. Некоторые запросы могут дойти до журнала ретрансляции и, в конечном счете, быть выполненными на подчиненном сервере. Такие запросы не требуют особых мер.

Куда хуже обстоит дело с запросами, которые ожидают события, не отправленного с главного сервера до его сбоя. При этом нужно обеспечить их обработку. Как правило, это значит, что в ответ на эти запросы возвращается ошибка, и пользователю придется отправить запрос заново.

Отказ сервера ретрансляции

К серверам, выполняющим роль серверов ретрансляции, требуется особый подход. В случае сбоя подчиненные серверы нужно перевести на другой сервер ретрансляции или непосредственно на главный сервер. Ретранслятор добавляют, чтобы уменьшить нагрузку на главный сервер, поэтому главный сервер, скорее всего, не справится с целой группой подчиненных, до сбоя подключенных к одному из ретрансляторов.

Аварийное восстановление

В мире высокой доступности слово «авария» не обязательно означает землетрясение или наводнение — оно всего лишь означает серьезную неисправность компьютера.

Простой пример: перебои энергоснабжения центра данных может произойти не только из-за отключения всего города — достаточно отключить одно здание. Рассказ о реальных событиях с оценкой последствий для MySQL вы найдете в разделе «Отключение mysql.com».

Сущность аварии в том, что происходит отказ множества систем сразу, сводя на нет эффект от дублирующих серверов, расположенных в этом же центре обработки данных. Вместо этого необходимо защитить данные, разместив их на другом, географически удаленном сервере. И эта рекомендация находит отклик: различные компоненты стараются держать в разных офисах, даже если компания невелика.

Процедуры

После того как вы избавились от всех единых точек сбоя, обеспечили в системе достаточную избыточность и составили планы на все случаи жизни, вы готовы к заключительному шагу.

Все ваши ресурсы и тщательное планирование не спасут, если вы не сумеете их правильно применить. Как правило, небольшим сайтом с несколько-

ми серверами можно управлять вручную, не зацикливаясь на планировании, но с ростом числа серверов автоматизация становится необходимостью, а в успешном бизнесе количество серверов растет ой как быстро.

Наверное, лучше сразу уволиться, чем планировать «начать автоматизацию с понедельника». На растущем предприятии и без того хватает дел — времени на автоматизации ИТ-системы просто не будет.

Некоторые основные процедуры мы уже обсудили, но у вас должны быть готовые автоматизированные процедуры для решения хотя бы этих задач:

- *Добавление новых подчиненных серверов* Основополагающий принцип масштабирования крупного сайта — создание новых подчиненных серверов. Есть несколько возможностей сделать это. В основе всех способов лежит создание снимка существующего сервера, — как правило, подчиненного, — восстановление снимка на новом сервере и запуск репликации с соответствующей позиции.

  Разумеется, от времени, потраченного на создание снимка, будет зависеть то, как быстро вы сможете запустить новый подчиненный сервер. Если архивация займет слишком много времени, главный сервер сможет отправить очень много изменений, и новому подчиненному серверу придется дольше нагонять его. Здесь важна скорость создания снимка. На рис. 4-1 показано время создания снимка и наверстывания изменений. Видно, что после останова подчиненного сервера для создания снимка изменения накапливаются, что приводит к росту числа невнесенных изменений. После перезапуска подчиненный сервер начнет вносить изменения и их очередь уменьшится.

  Среди разнообразных методов создания снимков есть следующие:

- *Утилита mysqldump* Безопасный, но медленный способ. Позволяет восстанавливать данные в БД, основанной на другом механизме. Так, если исходная БД использует InnoDB, можно будет создать согласованный снимок, не отключая сервер.
- *Копирование файлов БД* Сравнительно быстрый способ, однако перед копированием файлов требуется отключить сервер.
- *Метод оперативной архивации* Существуют различные способы, например, InnoDB Hot Backup.
- *Создание снимка с помощью LVM* В Linux можно создать снимок тома при помощи менеджера логических томов (Logical Volume Manager — LVM). Для этого нужна предварительная подготовка, так как будет создан особый LVM-том.
- *Снимки состояния файловой системы* ОС Solaris ZFS, например, имеет встроенную поддержку создания снимков. Это очень быстрый способ создания резервных копий, и похож на остальные приведенный выше способы (кроме mysqldump), но восстановление на другом компьютере выполнить не удастся.

Способы создания новых подчиненных серверов вы найдете в главе 2, а различные способы архивации — в главе 12.

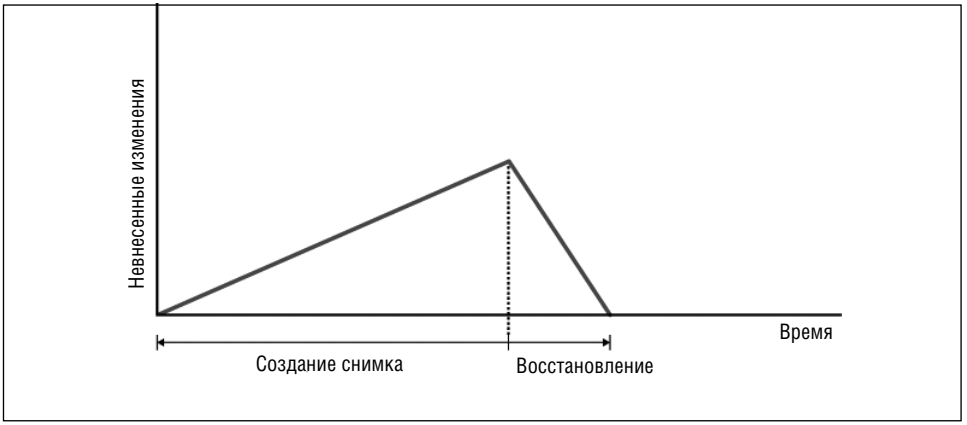


Рис. 4-1. Изменения, не выполненные во время создания снимка

- *Удаление подчиненных серверов из топологии* Удаляя подчиненные серверы из установки, необходимо всего лишь уведомить подсистему балансировки нагрузки об отсутствии сервера. Пример подсистемы балансировки с методами добавления и удаления серверов вы найдете в главе 5.
- *Смена главного сервера* Часто при выполнении повседневных задач нужно перевести все подчиненные серверы с исходного главного сервера на новый, уведомив подсистемы балансировки нагрузки об отсутствии главного сервера. Данную процедуру следует выполнять без простоя — она не должна влиять на обычные операции.

  Один из способов решения этой проблемы — повышение подчиненного сервера (см. ниже в этой главе), но, бывает, что проще воспользоваться горячим резервом (также описанным в этой главе).
- *Обработка сбоев на подчиненном сервере* Подчиненные серверы ломаются — вопрос лишь в том, как часто. Обработка сбоев на починенном сервере — обычное дело в любом развертывании. Необходимо всего лишь обнаружить отсутствие сервера и удалить его из пула подсистемы балансировки нагрузки.
- *Обработка сбоев на главном сервере* При внезапном отказе главного сервера вы должны обнаружить сбой и переключить все подчиненные серверы на резервный главный сервер или повысить один из подчиненных серверов до главного. Способы сделать это описаны далее в этой главе.
- *Обновление подчиненных серверов* Обновление MySQL на подчиненных серверах, как правило, проходит без проблем. Тем не менее, с изъятием подчиненного сервера из системы для обновления требуется удалить его из подсистемы балансировки нагрузки и, возможно, уведомить другие системы об отсутствии сервера.

- *Обновление главных серверов* Перед обновлением главного сервера, как правило, нужно обновить все подчиненные серверы. Однако бывают исключения. Как правило, на время выполнения обновления вводят в строй резервный главный сервер или повышают до главного один из подчиненных серверов.

Отключение mysql.com

Команда разработчиков MySQL состоит из разносторонних и преданных людей, способных работать с любыми системами и оборудованием. В отличие от многих других ИТ-команд, с которыми мне приходилось работать, ребята легко управляются с целым парком компьютеров, накопленным проектом MySQL за годы рбаоты: от высококлассных Windows-компьютеров до древних SGI Irix и HP-UX, всеми доступными средствами заставляя их работать вместе.

Тестирование сервера MySQL на различных компьютерах происходило в нашей «святая святых», в родном вычислительном центре, но с ростом проекта в нем стало тесновато. Тогда мы создали новый ВЦ в лучшем (и более дорогом) помещении в Стокгольме. Переезд запланировали на выходные, но за несколько дней до этого дело приняло неприятный оборот.

Обычно я работаю дома, но именно в тот день у меня было несколько встреч в офисе в городе Уппсала. Утром я обратил внимание на то, что сайт mysql.com был недоступен, но все равно поехал в офис, надеясь на лучшее.

По приезде я увидел, что часть команды MySQL занята прокладкой силовых кабелей до ВЦ. Очевидно, во всем здании отключился свет, однако в соседних зданиях электричество было, и силами наших ИТ-шников была проложена «времянка» к критическим серверам разработчиков.

Авария была довольно серьезной, и к моему приезду ИБП уже разрядились. Участок электросети, ведущей к зданию, был поврежден, а инженеры из электрокомпании не могли назвать сроки починки. Поэтому часть ИТ-команды приняла решение сразу везти веб-оборудование. Загрузив все автомобили, которые удалось найти, они повезли оборудование в новый ВЦ в Стокгольм — это примерно на 100 км южнее.

После подключения веб-оборудования в Стокгольме сайт mysql.com был восстановлен, но чтобы восстановить компьютеры разработчиков при помощи доступных источников питания — электросеть все еще находилась в ремонте — пришлось изрядно попотеть. Группа работала 48 часов практически без сна, в результате компьютеры были полностью восстановлены и приведены в рабочее состояние, что выше всяких похвал.

Команда MySQL вернулась к своей обычной работе...

Горячий резерв

Наиболее простая топология дублирования серверов — это топология горячего резерва. Она состоит (рис. 4-2) из главного сервера и выделенного сервера, называемого «горячим резервом» и дублирующего главный сервер.

Сервер горячего резерва подключен к главному серверу как подчиненный, читая и внося все изменения.

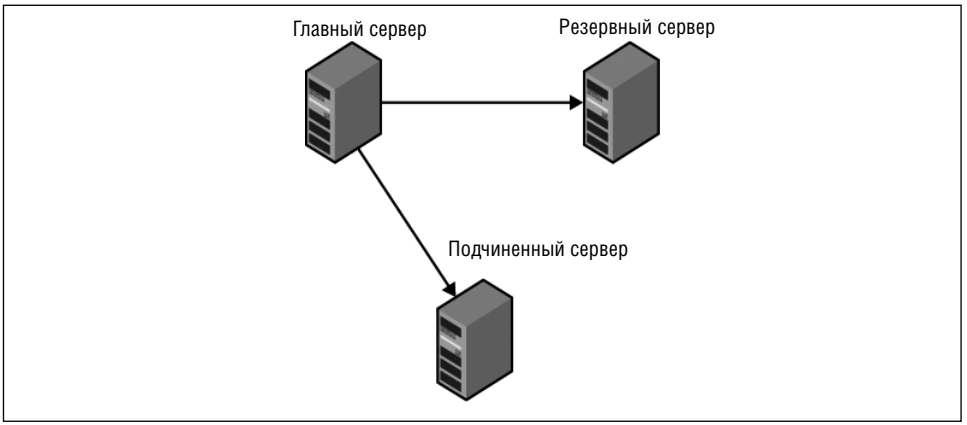


Рис. 4-2. Главный сервер и сервер горячего резерва

Идея состоит в том, что в случае отказа главного сервера горячий резерв — точная копия главного сервера — заменит отказавший сервер. Это даст возможность переключить всех клиентов и подчиненные серверы на резервный сервер и продолжить работу. Но, как и со многими идеями, скоро сказка сказывается, да не скоро дело делается...

Сбой неизбежен, по крайней мере, если речь идет о крупном развертывании. Сервер без сомнения «упадет», вопрос лишь, *когда* это произойдет, и *с какой периодичностью* будет повторяться. Отказ главного сервера, независимо от причины, не должен привести к останову развертывания. Чтобы обеспечить продолжение работы в случае сбоя главного сервера, необходимо иметь работающий сервер в горячем резерве и перевести на него все подчиненные серверы. Это даст вам возможность узнать, что случилось с главным сервером, исправить или заменить его. После ремонта главный сервер нужно вернуть в сеть и либо оставить в горячем резерве, либо перенаправить на него — исходный главный сервер — все подчиненные серверы.

Как все просто на словах. К сожалению, на деле вас может ждать ряд проблем:

- После перехода на резервный сервер будет выполняться репликация нового главного сервера. Поэтому будет необходимо сопоставлять позиции в двоичных журналах исходного главного сервера и резервного сервера.
- После перевода подчиненного сервера на резервный сервер на последнем могут оказаться не все изменения, внесенные на подчиненном сервере.
- По возвращении в строй восстановленного главного сервера в его двоичном журнале могут оказаться изменения, не отправленные до сбоя.

Все эти вопросы крайне важны, но для начала давайте рассмотрим простой случай — переход с работающего главного сервера на резервный для профилактики главного сервера (рис. 4-3). В этом случае главный сервер

остается в работе, что существенно облегчает положение, так как мы можем управлять главным сервером и заставить его работать на нас, а не против нас. Позднее мы рассмотрим случаи отключения главного сервера из-за ошибки ПО, из-за того, что сотрудник, будучи не в духе, пнул сервер ногой, или уборщица задела сетевой шнур.

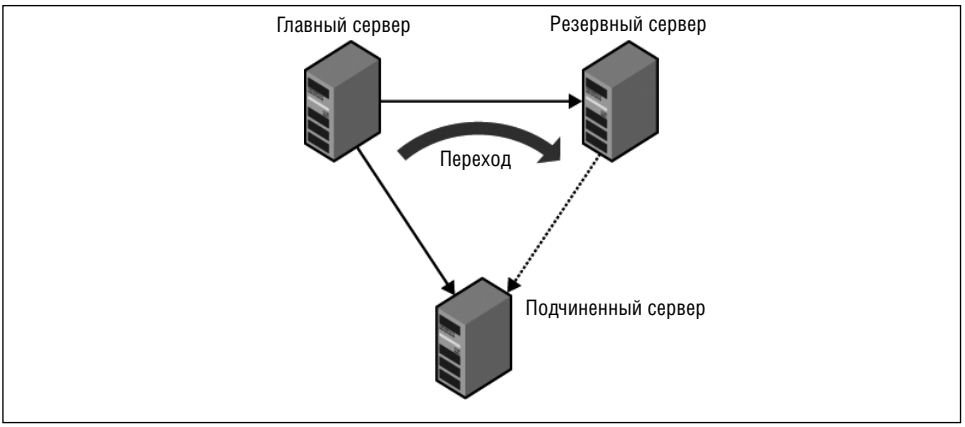


Рис. 4-3. Переход с работающего главного сервера на резервный

По умолчанию события, выполняемые потоком подчиненного сервера, не регистрируются в двоичном журнале. Это чревато проблемой, если данный подчиненный сервер находится в горячем резерве главного сервера. В таком случае необходимо, чтобы все изменения, посылаемые главным сервером на резервный, записывались в двоичный журнал резервного сервера. В противном случае, нечего будет реплицировать. Для этого нужно настроить резервный сервер, добавив с файл *my.cnf* параметр *log-slave-updates*. Этот параметр обеспечит запись команд, присланных главным сервером, в двоичный журнал подчиненного сервера.

```
[mysqld]
user            = mysql
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
port            = 3306
basedir         = /usr
datadir         = /var/lib/mysql
tmpdir          = /tmp
log-bin         = master-bin
log-bin-index   = master-bin.index
server-id       = 1
log-slave-updates
```

После обновления файла параметров перезапустите сервер.

Основная проблема перехода на резервный сервер в данном случае заключается в следующем. Нужно выполнить переход так, чтобы репликация

на резервном сервере началась ровно с той позиции, на которой она прервалась на главном. Если бы позиции легко сопоставлялись (например, если они совпадают), проблемы бы не было. К сожалению, позиции на главном и резервном серверах могут отличаться по целому ряду причин. Наиболее распространенный случай: резервный сервер не был подключен к главному на момент запуска главного сервера. И даже если он был подключен, нет гарантии, что в двоичном журнале резервного сервера события записаны точно так же, как в двоичном журнале главного сервера.

Основной принцип перехода таков: нужно остановить подчиненный и резервный серверы на одной и той же позиции, после чего просто перенаправить подчиненный сервер на резервный. Так как резервный сервер не вносил никаких изменений после позиции, на которой вы его остановили, то можно просто взять позицию из двоичного журнала резервного сервера и запустить подчиненный сервер с этой позиции. Тем не менее, простой останов подчиненного и резервного серверов не гарантирует их синхронизации — это нужно сделать вручную.

Для этого остановите подчиненный и резервный серверы и сравните позиции в их двоичных журналах. Так как обе позиции ссылаются на позиции главного сервера, — подчиненный и резервный серверы подключены к одному и тому же главному серверу, — то для сверки позиций достаточно сравнить имена файлов и позицию (смещение) — именно в таком порядке.

```
standby> SHOW SLAVE STATUS\G
         ...
   Relay_Master_Log_File: master-bin.000096
         ...
      Exec_Master_Log_Pos: 756648
   1 row in set (0.00 sec)

slave> SHOW SLAVE STATUS\G
         ...
   Relay_Master_Log_File: master-bin.000096
         ...
      Exec_Master_Log_Pos: 743456
   1 row in set (0.00 sec)
```

В данном случае резервный сервер опережает подчиненный, поэтому запишите позицию резервного сервера и запустите подчиненный сервер до этой позиции. Чтобы заставить подчиненный сервер догнать резервный и остановиться в заданной позиции, используйте команду START SLAVE UNTIL, известную нам по останову сервера отчетов:

```
slave> START SLAVE UNTIL
    ->  MASTER_LOG_FILE = 'master-bin.000096',
    ->  MASTER_LOG_POS = 756648;
Query OK, 0 rows affected (0.18 sec)
```

```
slave> SELECT MASTER_POS_WAIT('master-bin.000096', 756648);
Query OK, 0 rows affected (1.12 sec)
```

Теперь подчиненный и резервный серверы остановлены на одной и той же позиции, и все готово для перехода на резервный сервер: при помощи команды CHANGE MASTER TO будет выполнено перенаправление подчиненного сервера на резервный и запуск подчиненного. Но остается вопрос: какую выбрать позицию? Из-за отличий имени файла и позиции, соответствующих точке останова на главном и на резервном сервере необходимо извлечь позицию, записанную резервным сервером в момент записи изменений в качестве главного сервера. Для этого выполните на резервном сервере команду SHOW MASTER STATUS:

```
standby> SHOW MASTER STATUS\G
*************************** 1. row ***************************
         File: standby-bin.000019
     Position: 56447
   Binlog_Do_DB:
Binlog_Ignore_DB:
1 row in set (0.00 sec)
```

Теперь перенаправьте подчиненный сервер на резервный, используя верную позицию:

```
slave> CHANGE MASTER TO
    ->  MASTER_HOST = 'standby-1',
    ->  MASTER_PORT = 3306,
    ->  MASTER_USER = 'repl_user',
    ->  MASTER_PASSWORD = 'xyzzy',
    ->  MASTER_LOG_FILE =
' standby-bin.000019',
    ->  MASTER_LOG_POS = 56447;
   Query OK, 0 rows affected (0.18 sec)

 slave> START SLAVE;
Query OK, 0 rows affected (0.25 sec)
```

Если наоборот, подчиненный сервер опережает резервный, можно просто поменять роли резервного и подчиненного серверов в предыдущих шагах. Такое случается, так как главный сервер находится в работе, и недостающие изменения могут быть перенесены с него как на подчиненный, так и на резервный сервер. В следующем разделе мы рассмотрим случай непредвиденного останова главного сервера, когда некоторые изменения отсутствуют и на подчиненном и на резервном сервере.

Управление переходом средствами Python

В лист. 4-1 приведен образец кода на языке Python для перенаправления подчиненного сервера на другой главный сервер. Функция replicate_to_position

предписывает серверу выполнять чтение из журнала главного сервера только до заданной позиции. При возврате процедуры подчиненный сервер остановится точно на данной позиции. Функция switch_to_master нацеливает подчиненный сервер на новый главный сервер. Для выполнения процедуры нужно, чтобы целевой сервер и главный сервер, оба были подключены к одному исходному главному серверу. Если это не так, невозможно будет сравнивать их позиции, и процедура создаст исключение.

Лист. 4-1. Процедура перехода на новый главный сервер

```
def replicate_to_position(server, pos):
   server.sql("START SLAVE UNTIL MASTER_LOG_FILE=%s, MASTER_LOG_POS=%s",
              (pos.file, pos.pos))
   server.sql("SELECT MASTER_POS_WAIT(%s,%s)", (pos.file, pos.pos))

def switch_to_master(server, standby):
    stop_slave(server)
    stop_slave(standby)
    server_pos = fetch_slave_position(server)
    standby_pos = fetch_slave_position(standby)
    if server_pos < standby_pos:
      replicate_to_position(server, standby_pos)
    elif server_pos > standby_pos:
      replicate_to_position(standby, server_pos)
    master_pos = fetch_master_position(standby)
    change_master(server, standby, master_pos)
    start_slave(standby)
    start_slave(server)
```

Два главных сервера

Популярный вариант систем с высокой доступностью — топология *«два главных сервера» (dual masters)*, В которой два главных сервера реплицируют изменения друг друга. Эта система проста в использовании, поскольку симметрична: для перехода на резервный главный сервер не требуется менять конфигурацию главного сервера, и вернуться на исходный главный сервер при отказе резервного очень просто.

Бывают активные и пассивные серверы. Активный сервер принимает запросы на запись, которые, вероятно, распространяются при помощи репликации. Если сервер пассивный, он не получает запросов на запись, и просто следует за активным главным сервером, как правило, готовый заменить его в случае сбоя.

При работе с двумя главными серверами возможно два варианта:

- *Два активных сервера* Оба сервера получают запросы на запись, передавая изменения друг другу.
- *Активный и пассивный сервер* Один из серверов, называемый активный главный сервер, обрабатывает запросы записи, в то время как другой —

пассивный главный сервер — поддерживает синхронизацию с активным главным сервером.

Такая система очень похожа на горячий резерв, но благодаря симметрии, взаимное переключение и переход в активное состояние выполняется легко. Обратите внимание, что в ней пассивный главный сервер не всегда может отвечать на запросы. В некоторых решениях, рассмотренных в этом разделе, пассивный главный сервер находится в холодном резерве.

В данных системах синхронизация серверов не всегда поддерживается с помощью репликации — есть и другие способы. В некоторых вариантах поддерживается установка с двумя активными серверами, а в других — с активным и пассивным.

Вариант с двумя главными активными серверами применяется для размещения серверов вблизи от географически удаленных групп пользователей, например, в офисах, находящихся в разных частях света. Каждая группа пользователей работает со своим локальным сервером, изменения реплицируются на другой главный сервер, и оба главных сервера работают синхронно. Благодаря локальному применению транзакций, увеличивается быстродействие системы. Важно знать, что при локальной обработке транзакций главные серверы действуют не строго последовательно, то есть информация на них отличается. Изменения, внесенные на одном из серверов, со временем будут перенесены на другой, но до того момента данные на этих серверах будут несогласованными.

Отсюда вытекают два важных следствия:

- Если на обоих главных серверах будет обновлена одна и та же информация, — к примеру, на оба сервера случайно добавили запись пользователя, — между обновлениями возникнет конфликт, который может привести к останову репликации.
- Если в период несогласованности данных произойдет отказ сервера, некоторые транзакции будут потеряны.

Частично исключить проблему конфликтов изменений можно, разрешив запросы на запись только на одном из серверов, т. е. сделав пассивным один из серверов. Так мы приходим к варианту с активным и пассивным сервером: активный сервер называют *первичный (primary)*, а пассивный — *вторичный (secondary)*.

Потери транзакций при отказе сервера — неизбежные издержки асинхронной репликации, но в некоторых случаях проблему можно смягчить. Ограничить число потерянных при сбое сервера транзакций можно благодаря новой возможности MySQL 5.5 под названием *полусинхронная репликация (semisynchronous replication)*. Идея полусинхронной репликации заключается в блокировке потока, применяющего транзакцию, до тех пор, пока транзакция не будет принята хотя бы одним из подчиненных серверов. События транзакции отправляются на подчиненный сервер после обработки транзакции механизмом БД. Поэтому число потерянных транзакций можно сократить до минимума — одной транзакция на поток.

Установка с активным и пассивным серверами, как и установка с двумя активными, симметрична, что позволяет легко переключаться с главного на резервный сервер и обратно. В зависимости от способа поддержки зеркал, пассивный главный сервер можно использовать для административных задач, таких как обновление сервера с последующим использованием обновленного сервера в качестве активного без простоя.

Одна из главных проблем, которую необходимо решить при использовании установки с активным и пассивным серверами такова: оба сервера могут решить, что они первичные — возникает т. н. синдром «двух начальников» (split-brain syndrome). Такое может случиться из-за кратковременной потери связи, в результате которой вторичный сервер повысит свои полномочия до первичного, а затем первичный снова вернется в сеть. Если оба сервера, будучи первичными, получат изменения, вероятен конфликт. Если используется общий диск, одновременная запись двумя серверами может привести к очень «интересным» проблемам с БД — весьма серьезным и трудно локализуемым.

Общие диски

На рис. 4-4 представлена простая система с двумя главными серверами. Здесь пара главных серверов подключена к SAN. В нашем случае оба сервера подключены к одной сети SAN и используют одинаковые файлы. Один из главных серверов пассивный, он не выполняет запись в файлы, пока активный главный сервер работает. В случае останова активного сервера его место займет резервный.

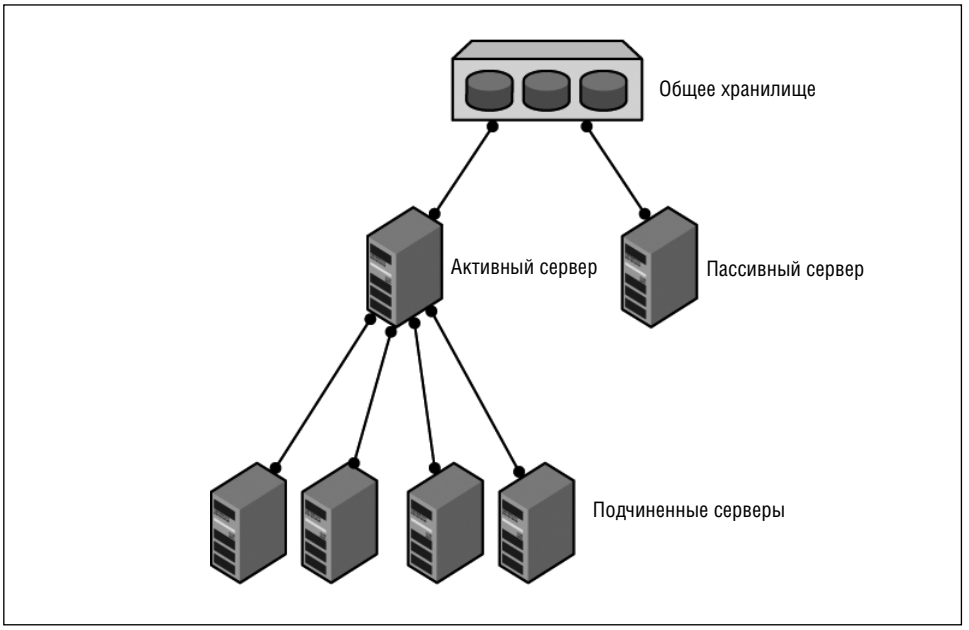


Рис. 4-4. Два главных сервера используют общий диск

Преимущество такого подхода в том, что нет необходимости сопоставлять позиции в двоичном журнале — файлы двоичного журнала хранятся на общем диске. Образы серверов — зеркальные отражения друг друга, но работают они на разных компьютерах. А это значит, что переход с работающего главного сервера на резервный займет минимум времени. Подчиненным серверам не придется транслировать позиции на новый главный сервер. Все, что нужно — это отметить позицию, на которой остановился подчиненный сервер, выполнить команду CHANGE MASTER и снова запустить репликацию.

В случае перехода на резервный сервер понадобится выполнить восстановление таблиц, так как весьма вероятно, что модификации данных завершены не будут. Разные механизмы БД реагируют на эту ситуацию по-разному. Например, с InnoDB, как и в случае сбоя, понадобится выполнить обычное восстановление по журналу транзакций, а если используется MyISAM, то для продолжения работы, возможно, понадобится исправить таблицы. Из этих двух вариантов InnoDB является предпочтительным, потому что восстановление гораздо быстрее, чем исправление таблиц MyISAM.

На лист. 4-2 представлен сценарий для выполнения подобного перехода, написанный на Python с использованием библиотеки Replicant. Обратите внимание, что в качестве позиции использован ID главного сервера, но поскольку оба сервера используют одни и те же файлы, резервный сервер — не что иное, как зеркальное отражение работающего сервера. ID сервера есть и в позиции, поэтому все допущенные пользователем ошибки будут реплицированы. Например, в команде можно передать главный сервер, который не является зеркалом работающего сервера.

Лист. 4-2. Процедура повышения полномочий подчиненного сервера с использованием общего диска

```
def remaster_slave(подчиненный сервер, главный сервер):
    position = fetch_slave_position(подчиненный сервер)
    change_master(подчиненный сервер, главный сервер, позиция)
```

Возможность установки двух главных серверов с использованием общих дисков зависит от типа общего хранилища, его рассмотрение выходит за рамки данной книги.

Проблема использования общего хранилища заключается в том, что два главных сервера хранят данные в одних и тех же файлах, следовательно, выполнять любые административные задачи на пассивном главном сервере нужно крайне осторожно. Перезапись, даже случайная, файлов конфигурации может стать фатальной.

Преодоление синдрома «двух начальников» зависит от типа решения, на котором основано общее хранилище, и в этой книге не рассматривается. Тем не менее, приведем пример для интерфейса SCSI, поддерживающего резервирование дисков серверами: сервер узнает о том, что он больше не является первичным, по дискам, которые зарезервированы другим сервером.

Диски, копируемые при помощи DRBD

В проекте Linux High Availability *(http://www.linux-ha.org)* есть ряд полезных средств по обслуживанию систем высокой доступности. Большинство из них не для этой книги, но одно средство — DRBD (Distributed Replicated Block Device — Распределённое копируемое блочное устройство) — весьма для нас интересно. Это ПО для репликации блочных устройств по сети.

На рис. 4-5 показана типичная система из двух узлов, в которой DRBD применяется для репликации диска на вторичный сервер. Система состоит из двух блочных устройств DRBD — по одному в каждом узле, —записывающих данные на реальные диски. Процессы DRBD устанавливают связь по сети, обеспечивая репликацию на вторичный сервер всех изменений первичного сервера. Для сервера MySQL репликация устройств выполняется явно. Устройства DRBD выглядят и работают так же, как и обычные диски, поэтому для серверов не требуется особого конфигурирования.

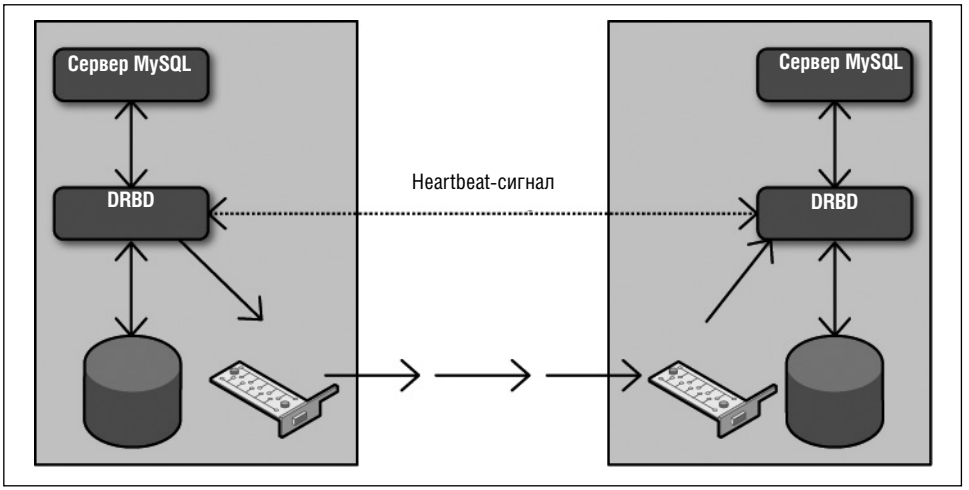


Рис. 4-5. Репликация дисков при помощи DRBD

Технология DRBD применима только в установке с активным и пассивным серверами, при этом полностью отсутствует доступ к пассивному диску. В отличие от рассмотренного ранее решения и двунаправленной репликации (см. далее в этой главе), пассивный главный сервер не может быть использован, даже в задачах, связанных только с чтением.

По аналогии с общими дисками, при использовании DRBD трансляция позиций между двумя главными серверами не нужна — они используют одни и те же файлы. Тем не менее, переход на резервный главный сервер занимает больше времени, чем в установке с общими дисками.

Перед подключением к сети серверов как в системе с общими дисками, так и в DRBD необходимо выполнить восстановление. Восстановление таблиц MyISAM обходится «недешево», поэтому рекомендуется применять тран-

закции, упрощающие восстановление таблиц БД. В данном случае InnoDB — проверенное решение, но его «догоняют» и другие механизмы БД, например PBXT. Так что время, потраченное на изучение альтернатив, окупится.

Системная БД *mysql* содержит только таблицы MyISAM, поэтому возьмите за правило не изменять эти таблицы без необходимости во время обычной работы. Разумеется, без этого не обойтись при выполнении административных задач.

Преимущество DRBD перед общими дисками в том, что общие диски — это единая точка отказа. В случае отказа сети между общим дисковым массивом и серверами есть вероятность, что сервер вообще остановится. Репликация дисков, наоборот, означает доступность данных на обоих серверах, что снижает риск полного отказа. Решение DRBD обладает «врожденным» иммунитетом против «синдрома двух начальников», восстановление после такого сбоя может выполняться автоматически.

Двунаправленная репликация

Когда два главных сервера используются в варианте с активным и пассивным серверами, такая система не слишком отличается от рассмотренного ранее решения с горячим резервом. Однако, в отличие от других решений на базе пары главных серверов (см. выше), возможна установка с двумя активными серверами (рис. 4-6).

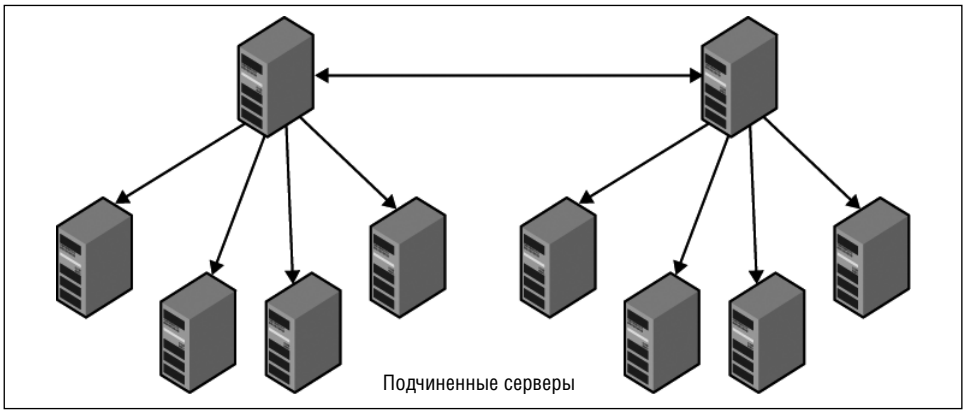


Рис. 4-6. Двунаправленная репликация

Несмотря на споры в определенных кругах, система с двумя активными серверами имеет свои преимущества. Типичный случай, когда два офиса работают с локальной информацией в одной БД, к примеру, с данными о продажах или о служащих. Требуется уменьшить время отклика БД, обеспечив доступность данных в обоих филиалах. В этом случае данные действительно локальны для обоих офисов: торговые представители обычно работают со своими продажами и крайне редко вносят изменения в данные других представителей.

Выполните следующие действия для настройки двунаправленной репликации:

1. Проверьте, что оба сервера имеют разные ID.
2. Проверьте, что данные на обоих серверах одинаковые (и что до начала репликации в их системы не были внесены изменения).
3. Создайте запись пользователя репликации и подготовьте репликацию (см. главу 1) на обоих серверах.
4. Запустите репликацию на обоих серверах.

Работая с двунаправленной репликацией, следует знать о конфликтах. Если оба сервера обновят одни и те же данные, возникнет конфликт, который и не обнаружится. Если вам повезет, репликация остановится из-за ошибки команды, но я бы на это не рассчитывал. Для построения системы высокой доступности следует запретить серверам обновлять одни и те же данные на прикладном уровне.

Даже если данные разделены, как в предыдущем примере с двумя удаленными офисами, важно принять меры по недопущению случайной модификации данных на соседнем сервере.

В нашем случае приложение должно связаться с сервером, ответственным за обработку данных о служащих, и обновить информацию на нем, а не просто обновить локальную БД в надежде на лучшее.

Для подключения подчиненных серверов к любому из главных серверов необходимо включить параметр log-slaveupdates. Исходя из того, что соседний главный сервер тоже подключен в качестве подчиненного, возникает вопрос: что же будет с рассылаемыми сервером событиями, когда те вернутся обратно на сервер?

В ходе репликации к событию прикрепляется ID сервера, на котором было создано событие. Этот ID распространяется дальше, когда подчиненный сервер записывает событие в свой двоичный журнал. Когда сервер видит событие в котором ID сервера совпадает с его собственным ID, это событие просто пропускается, и репликация продолжается со следующего события.

Иногда такие события нуждаются в обработке. Такое случается, когда вы, удалив старый сервер и создав новый с идентичным ID, выполняете восстановление на определенный момент времени. В подобных случаях можно отключить проверку с помощью переменной конфигурации replicate-same-server-id. Но чтобы не перехитрить самого себя, не устанавливайте данный параметр совместно с параметром log-slave-updates, иначе события будут ходить по кругу и переполнят серверы. Во избежание этого при использовании параметра replicate-same-server-id запрещается пересылка событий.

Система с двумя активными серверами требует безопасного разрешения конфликтов. Самый простой и единственный способ, рекомендуемый для данного типа установки, — сделать так, чтобы разные активные серверы выполняли запись в разные области.

Одно из возможных решений — назначить главным серверам разные БД или таблицы. В лист. 4-3 показан вариант с двумя таблицами, обновляемых

разными главными серверами. Для удобства просмотра разделенных данных создано представление, объединяющее две таблицы.

Лист. 4-3. Своя таблица для каждого офиса

```
CREATE TABLE Employee_Sweden (
   uid INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(20)
);

CREATE TABLE Employee_USA (
   uid INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(20)
);

-- Данное представление используется при  одновременном чтении из двух таблиц.
CREATE VIEW Employee AS
   SELECT 'Swe', uid, name FROM Employee_Sweden
 UNION
   SELECT 'USA', uid, name FROM Employee_USA;
```

Такой подход наиболее приемлем, например, когда каждому офису выделены отдельные таблицы под локальные данные, объединяемые только при создании отчетов. На первый взгляд, все довольно просто, однако использование и администрирование таблиц может быть затруднено следующими моментами:

- *Чтение и запись в отдельные таблицы* Представление определено так, что обновить его невозможно. Операции записи должны выполняться в реальных таблицах, в то время как для операций чтения можно использовать представление либо сами таблицы. Поэтому для разделения чтения и записи по разным таблицам может понадобиться знание логики приложения.
- *Точность и актуальность данных* Двумя таблицами управляют разные сайты, поэтому одновременное обновление таблиц погрузит систему в состояние, в котором на обоих серверах какое-то время будет информация, которой нет на другом сервере. Если в этот момент создать снимок информации, он будет неточным. Если требуется актуальная информация, ее должна обеспечивать прикладная логика, на которой мы не останавливаемся.
- *Оптимизация представлений* Когда используются представления, построить результирующий набор можно двумя методами. Первый — MERGE: представление расширяется «по месту», оптимизируется и выполняется, как запрос SELECT. Второй — TEMPTABLE: создается временная таблица, которая заполняется данными.

  При использовании сервером TEMPTABLE производительность очень низка, а быстродействие MERGE приближается к быстродействию соответствующего запроса SELECT. В MySQL TEMPTABLE применяет-

ся, когда в представление объявлено так, что его строки не соответствуют строкам таблицы, по которой оно построено. Примерами могут быть определения представлений, содержащие вложенные запросы UNION, GROUP BY или агрегатные функции. Вот почему тщательная разработка представлений так важна для производительности.

В любом случае, необходимо учитывать последствия использования представления, от этого зависит производительность.

Если каждому серверу назначены отдельные таблицы, риск конфликта отсутствует. Если же все сайты должны обновлять одни и те же таблицы, такая схема не подойдет.

Сервер MySQL предусматривает возможность выхода из этой ситуации при помощи двух серверных переменных:

- *auto_increment_offset* Данная переменная управляет начальным значением всех столбцов AUTO_INCREMENT в таблице. Это — значение, первой вставленной в таблицу строки в столбце AUTO_INCREMENT. Значение последующих строк вычисляется при помощи функции auto_increment_increment.
- *auto_increment_increment* Приращение, используемое для вычисления следующего значения в столбце AUTO_INCREMENT.

Обе переменные могут быть как глобальными, так и сеансовыми. Они влияют на все таблицы на сервере, а не только на созданные вами. Всякий раз, когда в таблицу вставляется строка с полем `AUTO_INCREMENT`, для нее используется серийный номер, вычисляемый следующим образом:

```
valueN = auto_increment_offset + N*auto_increment_increment
```

Запомните: следующее значение вычисляется как показано выше, а не прибавлением значения auto_increment_increment к последнему значению в таблице.

Используйте функции auto_increment_offset и auto_increment_increment для нумерации добавляемых в таблицу строк согласно заданной для сервера нумерации. Например: первый сервер использует ряд 1,3,5 (четные числа), а второй сервер — 2,4,6 (нечетные числа).

В продолжение лист. 4-3: в лист. 4-4 используется две переменные, обеспечивающие использование разных ID при вставке записей служащих в таблицу Employee.

Лист. 4-4. Два сервера, выполняют запись в одну таблицу

```
-- Общая таблица, которую можно создать на любом из серверов.
CREATE TABLE Employee (
   uid INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(20),
   office VARCHAR(20)
);

-- Настройка первого главного сервера
SET GLOBAL AUTO_INCREMENT_INCREMENT = 2;
```

```
SET GLOBAL AUTO_INCREMENT_OFFSET = 1;

-- Настройка второго главного сервера
SET GLOBAL AUTO_INCREMENT_INCREMENT = 2;
SET GLOBAL AUTO_INCREMENT_OFFSET = 2;
```

Данная схема позволяет вставлять строки, но при обновлении существующих строк крайне важно обеспечить отправку соответствующих команд нужному серверу (серверу с данными о служащих), иначе возникнет несогласованность. Если же при ошибке в обновлении подчиненные серверы продолжат работу, ошибки расползутся на оба сервера.

К примеру, первый главный сервер выполняет инструкцию

```
master-1> UPDATE Employee SET office = 'Vancouver' WHERE uid = 3;
Query OK, 1 rows affected (0.00 sec)
```

...а в это время на втором главном сервере модифицирована та же строка:

```
master-2> UPDATE Employee SET office = 'Paris' WHERE uid = 3;
Query OK, 1 rows affected (0.00 sec)
```

В результате первый главный сервер поместит служащего в Париж, а второй в Ванкувер. Важно найти и предотвратить подобные противоречия, ибо из-за них «съедут» другие значения, и несоответствия будут только расти. Во время логической репликации инструкции выполняются с учетом данных на обоих серверах, поэтому одно противоречие потянет за собой другие.

Если вы позаботитесь о разделении модификаций как описано выше, изменения строк будут реплицированы, и оба главных сервера останутся согласованными. Если пользователи работают с разными таблицами на разных серверах, самый простой способ избежать подобных ошибок — назначить разные привилегии, чтобы пользователь не смог случайно изменить таблицы на «чужом» сервере. Однако это не всегда возможно, и не спасет от описанной выше ситуации.

Полусинхронная репликация

В Google есть широкий набор исправлений MySQL и InnoDB. Одно из них, предназначенное для MySQL 5.0, дает поддержку *полусинхронной репликации (semisynchronous replication)*. Эта функция была переработана и вошла в «штатный» функционал MySQL 5.5.

Идея полусинхронной репликации в том, чтобы записать любое изменение на диск, по меньшей мере, одного из подчиненных серверов, прежде чем разрешать его внесение. Это значит, что в случае отказа главного сервера каждое подключение потеряет не более одной транзакции.

Важно понимать, что поддержка полусинхронной репликации не откладывает фиксацию транзакций — отправка ответа клиенту задерживается до записи транзакции в журнал ретрансляции минимум одного подчиненного сервера. На рис. 4-7 показан порядок вызовов во время завершения транзак-

ции. Видно, что транзакция фиксируется в БД до отправки на подчиненный сервер, но ответ приходит клиенту лишь после того, как подчиненный сервер убедится, что транзакция сброшена в постоянную память.

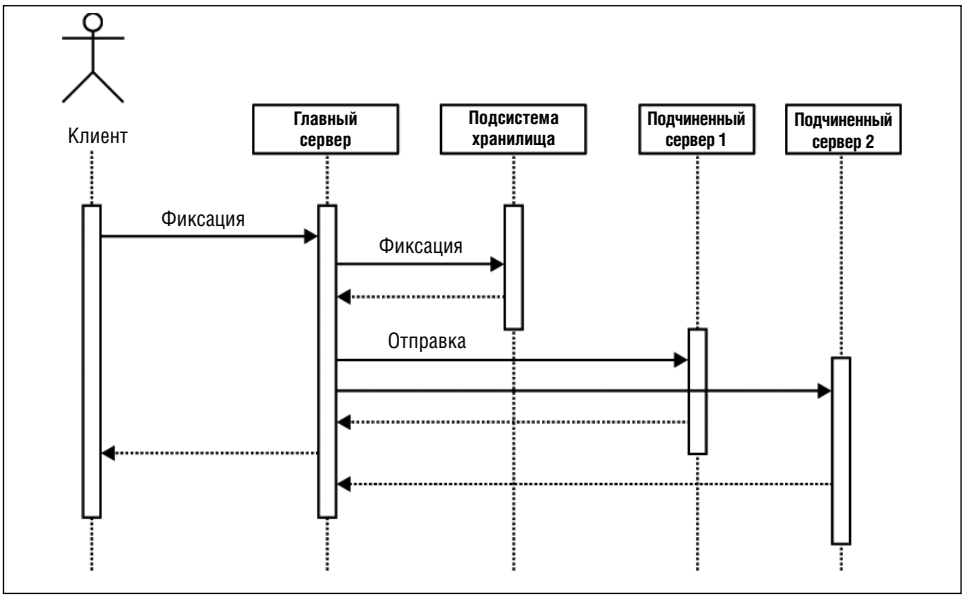


Рис. 4-7. Завершение транзакций при полусинхронной репликации

Следовательно, каждое подключение может потерять транзакцию, если отказ произойдет после завершения транзакции механизмом БД, но перед ее отправкой на подчиненный сервер. Но клиент получает подтверждение транзакции только после ее подтверждения подчиненным сервером, поэтому возможна потеря не более одной транзакции.

В большинстве случаев теряется одна транзакция клиента, но если клиент установил сразу несколько активных подключений к главному серверу, он может потерять по одной транзакции на каждое подключение, если в момент отказа сервера выполнялось несколько транзакций одновременно.

Настройка полусинхронной репликации

Для работы полусинхронной репликации она должна поддерживаться и на главном, и на подчиненном серверах. На обоих серверах должна работать СУБД MySQL версии 5.5 или выше и должна быть включена полусинхронная репликация. Если полусинхронная репликация не поддерживается одним из серверов, задействовать ее не удастся, и репликация будет выполняться как обычно. Следовательно, сохранится вероятность потери нескольких транзакций, если не принять меры, обеспечивающие доставку очередной транзакции на подчиненный сервер до начала следующей транзакции.

Выполните следующие действия для настройки полусинхронной репликации:

1. Установите подключаемый модуль на главный сервер:

```
master> INSTALL PLUGIN rpl_semi_sync_master SONAME'semisync_master.so';
```

2. Установите подключаемый модуль на подчиненный сервер:

```
slave> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

3. Установив модули, включите их на главном и подчиненном серверах. Данный процесс управляется двумя дополнительными серверными переменными. Отключите сервер и добавите нужные параметры в файл *my.cnf* главного сервера:

```
[mysqld]
rpl-semi-sync-master-enabled = 1
```

   и подчиненного сервера:

```
[mysqld]
rpl-semi-sync-slave-enabled = 1
```

4. Перезапустите серверы.

Теперь можно протестировать полусинхронную репликацию с оглядкой на следующее:

- Что будет, если откажут все подчиненные серверы (это вполне вероятно, если есть только один сервер), и подчиненный сервер не получит уведомления о том, что в журнале ретрансляции сохранена транзакция?
- Что будет, если отключатся все подчиненные серверы? В этом случае у главного сервера не останется подчиненных, на которых можно будет сохранить транзакции.

В дополнение к параметрам rpl-semi-sync-master-enabled и rpl-semi-sync-slave-enabled в вашем распоряжении есть два параметра для борьбы с описанными выше обстоятельствами:

- *rpl-semi-sync-master-timeout=миллисекунды* Во избежание блокировки полусинхронной репликации в случае отсутствия подтверждения установите параметр rpl-semi-sync-mastertimeout=миллисекунды.

  Если главный сервер не получит подтверждения до истечения таймаута, он вернется в режим обычной асинхронной репликации и продолжит работу без полусинхронной репликации.

  Данный параметр можно задать как серверную переменную, не отключая сервер. Но помните, что, как и значение любой серверной переменной, ее значение сохраняется до перезагрузки.
- *rpl-semi-sync-master-wait-no-slave={ON|OFF}* Если транзакция завершена, но к главному серверу не подключено ни одного подчиненного, то главный сервер не сможет отправить транзакцию на хранение. По умолчанию главный сервер будет ожидать подключения подчиненного сервера и подтверждения записи транзакции на диск в течение таймаута.

Параметр rpl-semi-sync-master-wait-no-slave={ON|OFF} позволяет изменить такое поведение: если у главного сервера нет подключенных подчиненных серверов, главный сервер вернется в режим асинхронной репликации.

Мониторинг полусинхронной репликации

С обоими модулями устанавливается ряд переменных состояния, позволяющих отслеживать полусинхронную репликацию. Мы коснемся лишь наиболее интересных из них, полный список см. в электронной документации по адресу: *http://dev.mysql.com/doc/refman/5.5/en/replication-semisync-interface.html*.

- *rpl_semi_sync_master_clients* Возвращает число подключенных подчиненных серверов, поддерживающих полусинхронную репликацию, и зарегистрированных для ее выполнения.
- *rpl_semi_sync_master_status* Состояние полусинхронной репликации на главном сервере: 1 — активна, 0 — неактивна (не включена либо включена, но после выполнен возврат к асинхронной репликации).
- *rpl_semi_sync_slave_status* Состояние полусинхронной репликации на подчиненном сервере: 1 — активна, то есть включена и работают потоки ввода-вывода; 0 — неактивна.

Узнать значения этих переменных можно либо с помощью команды SHOW STATUS, либо в информационной таблице схемы GLOBAL_STATUS. Если полученные значения предполагается использовать для других целей, то команда SHOW STATUS довольно сложна, проще SELECT-запросом (лист. 4-5) извлечь значения из информационной таблицы схемы в пользовательскую переменную.

Лист. 4-5. Выборка значений с помощью информационной схемы

```
master> SELECT Variable_value INTO @value
   ->  FROM INFORMATION_SCHEMA.GLOBAL_STATUS
   -> WHERE Variable_name = 'Rpl_semi_sync_master_status';
Query OK, 1 row affected (0.00 sec)
```

Повышение подчиненного сервера

Описанные выше процедуры работают, если функционирует главный сервер, позволяя синхронизировать резервный и подчиненный серверы до перехода. Но что, если главный сервер «отвалился» внезапно? Вследствие мгновенной остановки репликации со всеми подчиненными (включая резервный) серверами получение изменений для синхронизации главного сервера станет невозможным.

Если резервный сервер опережает свои подчиненные серверы, нет проблем: начните репликацию на каждом подчиненном сервер с того места, на котором остановился резервный сервер. Вы потеряете все изменения, сделан-

ные на главном сервере, но не отправленные на резервный. Восстановление главного сервера в такой ситуации будет рассмотрено отдельно.

Если один из подчиненных серверов опережает резервный сервер, не используйте резервный сервер в качестве нового главного, так как подчиненный сервер «знает» больше, чем новый главный. Будет лучше, если новым главным станет более «осведомленный» подчиненный сервер — ведь на него реплицировано больше событий с главного сервера, чем на остальные серверы.

Именно так можно преодолеть отказ главного сервера путем повышения подчиненного: вместо привязки к выделенному резервному серверу постарайтесь, чтобы все подключенные к главному подчиненные серверы могли быть повышены и взять на себя функции главного. Выбирая «более осведомленный» подчиненный сервер в качестве нового главного, найдите сервер, опережающий все остальные подчиненные серверы, которые будут подключены к новому главному и будут считывать его события.

Это критически важное решение для синхронизации всех подчиненных с новым главным сервером, от него зависит, будут ли потери и дублирование событий. В данной ситуации проблема заключается в том, что все подчиненные серверы считывают события с нового главного, но позиции в журнале нового главного сервера отличаются от таковой в журнале исходного главного сервера. Так что же делать бедному администратору БД?

Традиционный метод повышения подчиненного сервера

Перед выработкой окончательного решения ознакомимся с рекомендациями по повышению подчиненного сервера. Это поможет лучше понять проблему и обойти подводные камни.

На рис. 4-8 показана обычная система с главным и несколькими подчиненными серверами.

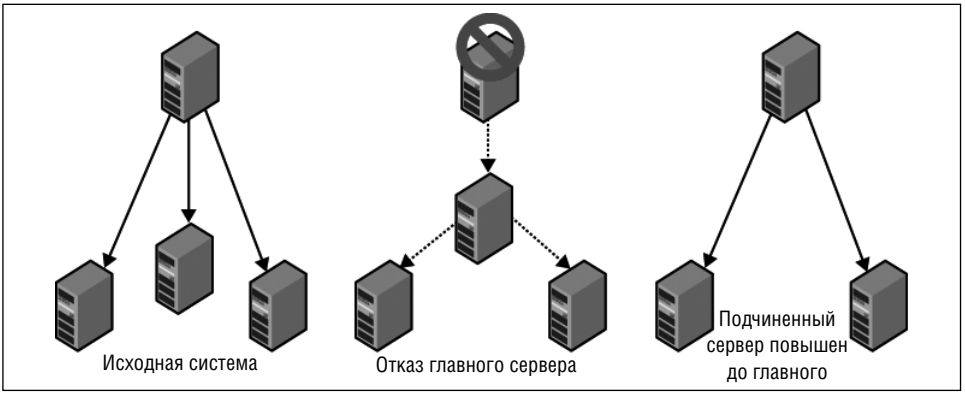


Рис. 4-8. Повышение подчиненного сервера для замены сбойного главного сервера

Для повышения подчиненного сервера традиционным способом необходимо следующее:

- На всех серверах, которые могут быть повышены, должна быть учетная запись пользователя репликации.
- Все такие серверы должны работать с параметром --log-bin, то есть с включенным двоичным журналом.
- На всех таких серверах должен быть отключен параметр --log-slave-updates (сейчас узнаете зачем).

Допустим, используется система, показанная на рис. 4-8, и вдруг происходит отказ главного сервера. Чтобы повысить подчиненный сервер до главного, нужно выполнить следующее:

1. Остановите подчиненный сервер, выполнив команду STOP SLAVE.
2. Выполните сброс подчиненного сервера — кандидата на роль главного командой RESET MASTER. Таким образом подчиненный сервер станет новым главным сервером, а все подключающиеся к нему подчиненные начнут считывать события с момента его повышения.
3. Подключите остальные подчиненные серверы к новому главному, выполнив CHANGE MASTER TO. Так как новый главный сервер был сброшен, репликацию можно запустить с начала двоичного журнала, и нет необходимости передавать позицию в команду CHANGE MASTER TO.

К сожалению, данный способ основан на зачастую ложном предположении о том, что подчиненные серверы получили все изменения, внесенные на главном сервере. В реальной системе подчиненные серверы по-разному отстают от главного, возможно, всего на несколько транзакций, но и это — отставание. Решить проблему попробуем в следующем разделе.

И все же данный способ так прост, что вполне подойдет, если потеря транзакций не слишком важна, или при работе на малой нагрузке.

Усовершенствованный способ повышения подчиненного сервера

Традиционный подход к повышению подчиненного сервера часто не срабатывает из-за отставания подчиненных серверов от главного. Типичная ситуация неожиданного отказа главного сервера показана на рис. 4-9. Прямоугольник «двоичный журнал» в центре — это двоичный журнал главного сервера, а стрелками отмечена часть журнала, воспроизведенная на подчиненных серверах.

Подчиненные серверы на рисунке остановились, каждый на своей позиции двоичного журнала. Чтобы решить этот вопрос и восстановить работу системы, один из подчиненных серверов (желательно с самой поздней позицией в двоичном журнале) нужно выбрать на роль нового главного. Остальные подчиненные серверы должны быть синхронизированы с новым главным сервером.

Критическая проблема в сопоставлении позиций в журналах подчиненных серверов: позицию в журнале приказавшего долго жить главного сервера нужно сопоставить позиции в журнале повышаемого сервера. К сожалению,

в процессе репликации теряется история обработанных событий и соответствующих им позиций двоичного журнала на подчиненных серверах: каждый раз, когда подчиненный сервер обрабатывает событие, полученное от главного, он записывает *новое* событие в свой двоичный журнал (естественно, в новой позиции). Позиции событий в журнале подчиненного сервера никак не связаны с позициями тех же событий в журнале главного сервера.

Рис. 4-9. Позиции в двоичном журнале главного и подключенных к нему подчиненных серверов

Нам остается одно — проверить двоичный журнал повышаемого подчиненного сервера. Чтобы сделать это:

- Включите двоичный журнал — в противном случае изменения реплицированы не будут.
- Включите протоколирование обновлений подчиненного сервера (параметр log-slave-updates) — иначе, изменения, полученные от исходного главного сервера, не будут пересылаться.
- Чтобы иметь возможность получить роль главного, на подчиненном сервере должна быть учетная запись пользователя репликации. Так к серверу, выбранному на роль главного, смогут подключиться другие подчиненные серверы для репликации.

На всех подчиненных серверах, повышать которые не планируется, выполните следующие действия:

1. Выясните, какая транзакция была выполнена последней.
2. Найдите эту транзакцию в двоичном журнале повышаемого сервера.
3. Найдите ее позицию в двоичном журнале повышаемого сервера.
4. На серверах, которые повышать не планируется, запустите репликацию с этой позиции в журнале повышаемого сервера.

Для того чтобы последняя транзакция на всех подчиненных серверах совпадала с соответствующим событием в двоичном журнале повышаемого сервера, всем транзакциям нужно присвоить метки. Содержание и структура меток не имеют значения — они просто должны быть уникальными. Не важно, какой из серверов выполнил транзакцию, главное, чтобы каждая транзакция была в двоичном журнале повышаемого сервера. У нас такие метки называются *глобальный ID транзакции (global transaction ID)*.

Самый простой способ сделать это — вставить в конце каждой транзакции команду вставки записи в специальную таблицу, отслеживающую позицию в журналах всех подчиненных серверов. Непосредственно перед завершением каждой транзакции эта инструкция записывает в таблицу уникальный номер транзакции.

Существует два способа работы с метками:

- Вставить необходимые команды в код приложения.
- Вызывать при фиксации каждой транзакции хранимую процедуру, записывающую метки.

Первый способ легче, его-то мы и покажем. Если вам по душе второй способ, см. раздел «Хранимые процедуры для завершения транзакций».

Для поддержки глобальных ID транзакций созданы две таблицы (лист. 4-6): *Global_Trans_ID* для генерации последовательности чисел и *Last_Exec_Trans* для записи глобальных ID транзакции.

Чтобы различать транзакции, завершенные на разных серверах, к определению таблицы *Last_Exec_Trans* добавлен ID сервера. Если, например, повышаемый сервер откажет до подключения всех подчиненных серверов, важно будет не перепутать ID транзакций исходного главного сервера и повышаемого сервера. В противном случае подчиненные серверы, не успевшие подключиться к повышаемому серверу, после перевода на второй повышаемый сервер могут начать выполнение с неверной позиции. В данном листинге таблица счетчиков обслуживается механизмом MyISAM, но можно использовать для этих целей и InnoDB.

Лист. 4-6. Таблицы для генерации и отслеживания глобальных ID транзакций

```
CREATE TABLE Global_Trans_ID (
   number INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
) ENGINE = MyISAM;

CREATE TABLE Last_Exec_Trans (
   server_id INT UNSIGNED,
   trans_id INT UNSIGNED
) ENGINE = InnoDB;

-- Вставка строки со NULL-значениями для последующего обновления.
INSERT INTO Last_Exec_Trans() VALUES ();
```

Следующий шаг — создать процедуру добавления глобального ID транзакции в двоичный журнал, чтобы программа, выполняющая повышение подчиненного сервера, смогла прочитать ID из журнала. Для этого подойдет следующая процедура:

1. Вставьте строку в таблицу-счетчик транзакций, предварительно отключив двоичный журнал, поскольку эту операцию не следует реплицировать на подчиненные серверы:

```
master> SET SQL_LOG_BIN = 0;
Query OK, 0 rows affected (0.00 sec)
master> INSERT INTO Global_Trans_ID() VALUES ();
Query OK, 1 row affected (0.00 sec)
```

2. Выполните выборку глобального ID транзакции при помощи функции LAST_INSERT_ID. Для упрощения кода одновременно извлекается ID сервера из серверной переменной server_id.

```
master> SELECT @@server_id as server_id, LAST_INSERT_ID() as trans_id;
+---------- +-----------+
| server_id | trans_id  |
+-----------+-----------+
|      0    |    235    |
+-----------+-----------+
1 row in set (0.00 sec)
```

3. Перед записью глобального ID транзакции в таблицу Last_Exec_Trans удалите соответствующую строку из таблицы-счетчика для экономии места. Этот дополнительный шаг актуален только для MyISAM. Если используется InnoDB, не удаляйте из таблицы последний глобальный ID транзакции.

```
master> DELETE FROM Global_Trans_ID WHERE number < 235;
Query OK, 1 row affected (0.00 sec)
```

4. Включите двоичный журнал:

```
master> SET SQL_LOG_BIN = 1;
Query OK, 0 rows affected (0.00 sec)
```

5. Добавьте в таблицу Last_Exec_Trans ID сервера и транзакции, полученные в шаге 2. Это последний шаг перед завершением транзакции с командой COMMIT:

```
master> UPDATE Last_Exec_Trans SET server_id = 0, trans_id = 235;
Query OK, 1 row affected (0.00 sec)

master> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Каждый глобальный ID транзакции — это точка, откуда можно продолжить репликацию, поэтому вышеописанную процедуру необходимо выполнить для каждой транзакции. Если этого не сделать, соответствующая транзакция не получит метки, и начать с нее репликацию будет невозможно.

Теперь, чтобы повысить подчиненный сервер после отказа главного, найдите среди подчиненных сервер с новейшими изменениями — номер позиции в двоичном журнале будет у него самым большим — и выполните его повышение до главного. Затем подключите к нему все подчиненные серверы.

Чтобы подключить подчиненный сервер к повышаемому серверу и запустить репликацию с нужной позиции, нужно найти в журнале повышаемого сервера позицию последней выполненной подчиненным сервером транзакции, используя ID транзакции.

Выполните восстановление следующим образом:

1. Остановите подчиненный сервер. Найдите глобальный ID последней транзакции в таблице *Last_Exec_Trans*.
2. Выберите подчиненный сервер с максимальным глобальным ID транзакции, чтобы повысить его до главного. Если таких серверов несколько, выберите из них один.
3. Найдите позицию этой транзакции в журналах повышаемого и подчиненного серверов при помощи команды SHOW MASTER LOGS. Обратите внимание, что последняя строка вывода команды SHOW MASTER LOGS идентична выводу команды SHOW MASTER STATUS.
4. Подключите подчиненный сервер, чтобы начать его обновление.
5. Подключитесь к повышенному серверу, найдите в его журнале глобальный ID новейшей транзакции из двоичных журналов подчиненных серверов. Если у вас нет точной позиции, единственно правильным будет начать чтение двоичного журнала с начала. Поэтому просматривайте двоичные журналы в обратном порядке — с конца. Так вы получите позиции в двоичном журнале повышенного сервера для всех глобальных ID транзакций, собранных в шаге 1.
6. Подключите все подчиненные серверы к повышенному серверу, начиная с позиции, с которой должен стартовать подчиненный сервер, чтобы восстановить всю информацию. Используйте сведения, полученные на шаге 5.

Первые четыре шага просты, сложность представляет шаг 5. Чтобы проиллюстрировать ситуацию, возьмем для примера базовую информацию, собранную на первых трех шагах. В табл. 4-2 представлены три подчиненных сервера с глобальным ID последней транзакции на каждом.

Табл. 4-2. Глобальный ID транзакции для всех подключенных подчиненных серверов

	Server ID	Trans ID
slave-1	1	245
slave-2	1	248
slave-3	1	256

Как видно из табл. 4-2, судя по ID, новейшая транзакция находится на сервере slave-3, поэтому повысить следует именно его. Следовательно, необходимо найти позицию в двоичном журнала сервера slave-3 для последней транзакции на каждом подчиненном сервере, используя глобальный ID. Для этого понадобится информация о двоичном журнале на сервере slave-3, которую мы получим, выполнив код с лист. 4-7.

Лист. 4-7. Позиции двоичного журнала повышаемого сервера Slave-3

```
slave-3> SHOW MASTER LOGS;
+--------------------+-----------+
| Log_name           | File_size |
+--------------------+-----------+
| slave-3-bin.000001 | 3115      |
| slave-3-bin.000002 | 345217    |
| slave-3-bin.000003 | 24665     |
| slave-3-bin.000004 | 788243    |
| slave-3-bin.000005 | 1778      |
+--------------------+-----------+
5 row in set (0.00 sec)
```

В выводе команды SHOW MASTER LOGS нас больше всего интересуют имена журналов для поиска в них глобальных ID транзакции. Пример чтения файла slave-3-bin.000005 с помощью команды mysqlbinlog и часть вывода показан в лист. 4-8. Глобальный ID транзакции, полученной сервером slave-3 и начинающейся с позиции 596 (выделен в первой строке вывода), был получен сервером slave-1 (см. поле UPDATE в таблице *Last_Exec_Trans*).

Лист. 4-8. Вывод команды mysqlbinlog для одной транзакции

```
#  at 596
#091018 18:35:42 server id 1 end_log_pos 664 Query  thread_id=952  ...
SET TIMESTAMP=1255883742/*!*/;
BEGIN
/*!*/;
#  at 664
#091018 18:35:42 server id 1 end_log_pos 779 Query  thread_id=952  ...
SET TIMESTAMP=1255883742/*!*/;
UPDATE user SET messages = messages + 1 WHERE id = 1
/*!*/;
#  at 779
#091018 18:35:42 server id 1 end_log_pos 904 Query  thread_id=952  ...
SET TIMESTAMP=1255883742/*!*/;
INSERT INTO message VALUES (1,'MySQL Python Replicant rules!')
/*!*/;
#  at 904
#091018 18:35:42 server id 1 end_log_pos 1021 Query  thread_id=952  ...
SET TIMESTAMP=1255883742/*!*/;
UPDATE Last_Exec_Trans SET server_id = 1, trans_id = 245
```

```
/*!*/;
#  at 1021
#091018 18:35:42 server id 1 end_log_pos 1048 Xid = 1433 COMMIT/*!*/;
```

Как видно из табл. 4-2, последней транзакцией, которую «увидел» сервер slave-1, была транзакция с trans_id=245. Отсюда следует, что начальная позиция для сервера slave-1 находится в файле slave-3-bin.000005 по смещению 1048. Поэтому, чтобы запустить slave-1 с правильной позиции, достаточно выполнить команды CHANGE MASTER TO и START SLAVE:

```
slave-1> CHANGE MASTER TO
         ->  MASTER_HOST = 'slave-3',
         ->  MASTER_LOG_FILE = 'slave-3-bin.000005',
         ->  MASTER_LOG_POS = 1048;
Query OK, 0 rows affected (0.04 sec)

slave-1> START SLAVE;
Query OK, 0 rows affected (0.17 sec)
```

Продолжая в том же духе, ищите позиции транзакций, записанных вами на шаге 1. Так вы сможете поочередно подключить подчиненные серверы к новому главному, запустив репликацию с правильной позиции.

Данный способ работает хорошо, если в завершение каждой транзакции исполняется команда UPDATE. К сожалению, существуют команды, выполняющие неявную фиксацию до выполнения такой команды. Типичные примеры: CREATE TABLE, DROP TABLE и ALTER TABLE. Неявный характер фиксации не позволяет присвоить этим командам метки, следовательно, возобновить работу сразу после них будет невозможно. Это значит, что при отказе после выполнения кода с лист. 4-9, у вас могут быть проблемы.

Если на подчиненном сервере выполнена инструкция CREATE TABLE, а затем связь с главным сервером прервалась, глобальный ID транзакции, которую сервер «увидел» последней, соответствует команде INSERT INTO, предшествующей CREATE TABLE. Поэтому подчиненный сервер будет пытаться повторно подключиться к повышенному серверу с позиции транзакции, соответствующей команде INSERT INTO. Данная позиция будет найдена в двоичном журнале повышенного сервера, и репликация инструкции CREATE TABLE начнется заново, что приведет к остановке подчиненного сервера с ошибкой.

Избежать подобных проблем поможет тщательно продуманный код. К примеру, если CREATE TABLE заменить на CREATE TABLE IF NOT EXISTS, подчиненный сервер увидит, что таблица уже существует, и пропустит эту команду.

Лист. 4-9. Команды, которым невозможно назначить глобальный ID транзакции

```
INSERT INTO message_board VALUES ('mats@sun.com', 'Hello World!');
CREATE TABLE admin_table (a INT UNSIGNED);
INSERT INTO message_board VALUES ('', '');
```

Повышение подчиненного сервера средствами Python

Мы рассмотрели два способа повышения сервера: традиционный, недостаток которого — потеря транзакций на некоторых подчиненных серверах; и усложненный, позволяющий восстановить все доступные транзакции. Традиционный способ легко реализовать на Python, поэтому давайте разберем более сложный. Итак, чтобы повысить подчиненный сервер, необходимо:

- Правильно настроить все подчиненные серверы.
- На главный сервер добавить таблицы *Global_Trans_ID* и *Last_Exec_Trans*.
- В коде приложения обеспечить правильную фиксацию транзакции.
- Написать код, автоматизирующий повышение подчиненного сервера.

Для работы с новым типом сервера используется класс Promotable (лист. 4-10). Видно, что в листинге использован уже знакомый нам вспомогательный метод _enable_binlog и добавлен метод для определения параметра log-slave-updates. Так как на сервере, пригодном для повышения, должны быть таблицы, использованные ранее для главного сервера, следует добавить такие таблицы на главный сервер. Для этого создадим функцию _add_global_id_tables. Если эти таблицы существуют, считается, что они определены правильно и пересоздаваться не будут. Однако для модификации таблицы *Last_Exec_Trans* в ней должна быть хотя бы одна строка. Поэтому, если таблица не существует, мы создадим таблицу и добавим строку с NULL-значением.

Лист. 4-10. Определение роли повышаемого сервера

```
_GLOBAL_TRANS_ID_DEF = """
CREATE TABLE IF NOT EXISTS Global_Trans_ID (
   number INT UNSIGNED NOT NULL AUTO_INCREMENT,
   PRIMARY KEY (number)
)  ENGINE=MyISAM
"""

_LAST_EXEC_TRANS_DEF = """
CREATE TABLE IF NOT EXISTS Last_Exec_Trans (
   server_id INT UNSIGNED DEFAULT NULL,
   trans_id INT UNSIGNED DEFAULT NULL
) ENGINE=InnoDB
"""

class Promotable(Role):
   def __init__(self, repl_user, master):
      self.__master = master
      self.__user = repl_user
   def _add_global_id_tables(self, master):
      master.sql(_GLOBAL_TRANS_ID_DEF)
      master.sql(_LAST_EXEC_TRANS_DEF)
      if not master.sql("SELECT @@warning_count"):
         master.sql("INSERT INTO Last_Exec_Trans() VALUES ()")
```

```
def _relay_events(self, server, config):
   config.set('mysqld', 'log-slave-updates')

def imbue(self, server):
   # Извлечение и обновление конфигурации.
   config = server.get_config()
   self._set_server_id(server, config)
   self._enable_binlog(server, config)
   self._relay_event(server, config)

   # Подстановка новой конфигурации.
   server.stop()
   server.put_config(config)
   server.start()

   # Добавление таблиц на главный сервер
   self._add_global_id_tables(self.__master)

   server.repl_user = self.__master.repl_user
```

Этот код настраивает подчиненный и главный серверы для работы с глобальными ID транзакции. Вам по-прежнему придется обновлять таблицу *Last_Exec_Trans* во время завершения каждой транзакции. На лист. 4-11 представлен пример завершения транзакций, реализованный на PHP. Данный код написан на PHP, так как является частью кода приложения, а не сценария, управляющего развертыванием.

Лист. 4-11. Код начала, завершения и отмены транзакций

```
function start_trans($link) {
   mysql_query("START TRANSACTION", $link);
}

function commit_trans($link) {
   mysql_select_db("common", $link);
   mysql_query("SET SQL_LOG_BIN = 0", $link);
   mysql_query("INSERT INTO Global_Trans_ID() VALUES ()", $link);
   $trans_id = mysql_insert_id($link);
   $result = mysql_query("SELECT @@server_id as server_id", $link);
   $row = mysql_fetch_row($result);
   $server_id = $row[0];

   $delete_query = "DELETE FROM Global_Trans_ID WHERE number = %d";
   mysql_query(sprintf($delete_query, $trans_id),
            $link);
   mysql_query("SET SQL_LOG_BIN = 1", $link);

   $update_query = "UPDATE Last_Exec_Trans SET server_id = %d, trans_id = %d";
   mysql_query(sprintf($update_query, $server_id, $trans_id), $link);
   mysql_query("COMMIT", $link);
}
```

```
function rollback_trans($link) {
   mysql_query("ROLLBACK", $link);
}
```

Данный код можно использовать для завершения транзакций вместо привычных команд COMMIT и ROLLBACK. К примеру, можно написать функцию на PHP, позволяющую записывать сообщения к БД и обновлять счетчик сообщений для пользователя.

```
function add_message($email, $message, $link) {
   start_trans($link);
   mysql_select_db("common", $link);
   $query = sprintf("SELECT user_id FROM user WHERE email = '%s'", $email);
   $result = mysql_query($query, $link);
   $row = mysql_fetch_row($result);
   $user_id = $row[0];

   $update_user = "UPDATE user SET messages = messages + 1 WHERE user_id = %d";
   mysql_query(sprintf($update_user, $user_id), $link);
   $insert_message = "INSERT INTO message VALUES (%d,'%s')";
   mysql_query(sprintf($insert_message, $user_id, $message), $link);
   commit_trans($link);
}

$conn = mysql_connect(":/var/run/mysqld/mysqld1.sock", "root");
add_message('mats@example.com', "MySQL Python Replicant rules!", $conn);
```

Остается лишь выполнить реальное повышение подчиненного сервера в случае сбоя. План мы уже составили, только вот с реализацией будет немного посложнее.

Первый шаг: удаленно извлечь файлы двоичного журнала, как показано в главе 2. В данном случае, нам нужно извлечь двоичный файл целиком, так как мы не знаем, откуда начать чтение. Функция fetch_remote_binlog (лист. 4-12) возвращает итератор для строк двоичного журнала.

Лист. 4-12. Извлечение удаленного двоичного журнала

```
def fetch_remote_binlog(server, binlog_file):
    command = ["mysqlbinlog",
               "--read-from-remote-server",
               "--force",
               "--host=%s" % (server.host),
               "--user=%s" % (server.sql_user.name)]
   if server.sql_user.passwd:
      command.append("--password=%s" % (server.sql_user.passwd))
   command.append(binlog_file)
   return iter(subprocess.Popen(command, stdout=subprocess.PIPE).stdout)
```

Итератор одну за другой возвращает строки двоичного журнала. Строки нужно разделить на транзакции и события, чтобы облегчить работу с двоич-

ным журналом. В лист. 4-13 представлена функция group_by_event, группирующая строки, принадлежащие одному событию, в одну; и функция group_by_trans для группировки потока событий (возвращаемого функцией group_by_event) в списки, где каждый список представляет транзакцию.

Лист. 4-13. Извлечение транзакций посредством синтаксического анализа вывода инструкции mysqlbinlog

```
delimiter = "/*!*/;"

def group_by_event(lines):
   event_lines = []
   for line in lines:
      if line.startswith('#'):
         if line.startswith("# End of log file"):
            del event_lines[-1]
            yield ''.join(event_lines)
            return
         if line.startswith("# at"):
            yield ''.join(event_lines)
            event_lines = []
      event_lines.append(line)
def group_by_trans(lines):
   group = []
   in_transaction = False
   for event in group_by_event(lines):
      group.append(event)
      if event.find(delimiter + "\nBEGIN\n" + delimiter) >= 0:
         in_transaction = True
      elif not in_transaction:
         yield group
         group = []
      else:
         p = event.find("\nCOMMIT")
         if p >= 0 and (event.startswith(delimiter, p+7)
                        or event.startswith(delimiter, p+8)):
            yield group
            group = []
            in_transaction = False
```

В лист. 4-14 функция scan_logfile предназначена для поиска вводимых глобальных ID транзакции в выводе команды mysqlbinlog. В функцию передается главный сервер, файл двоичного журнала которого нужно извлечь, имя файла журнала для просмотра (имя файла — это имя двоичного журнала на главном сервере), и функция обратного вызова on_gid, которая будет вызываться всякий раз, когда будет найден глобальный ID транзакции. В функцию on_gid будут переданы: глобальный ID транзакции (состоящий из server_id и trans_id) и позиция, где транзакция завершается, в двоичном журнале.

Лист. 4-14. Поиск глобальных ID транзакции в файле двоичного журнала

```
_GIDCRE = re.compile(r"^UPDATE Last_Exec_Trans SET\s+"
                     r"server_id = (?P<server_id>\d+),\s+"
                     r"trans_id = (?P<trans_id>\d+)$", re.MULTILINE)
_HEADCRE = re.compile(r"#\d{6}\s+\d?\d:\d\d:\d\d\s+"
                     r"server id\s+(?P<sid>\d+)\s+"
                     r"end_log_pos\s+(?P<end_pos>\d+)\s+"
                     r"(?P<type>\w+)")

def scan_logfile(master, logfile, on_gid):
   from mysqlrep import Position
   lines = fetch_remote_binlog(master, logfile)
   # Поиск инструкций обновления глобального ID транзакции в #выводе.
   for trans in group_by_trans(lines):
      if len(trans) < 3:
         continue
      # Проверка обновлений таблицы Last_Exec_Trans
      m = _GIDCRE.search(trans[-2])
      if m:
         server_id = int(m.group("server_id"))
         trans_id = int(m.group("trans_id"))
         Проверка информационного комментария для end_log_pos.
         Мы рассматриваем только таблицы InnoDB, поэтому можно сказать,
         что транзакции завершаются событием Xid.
         m = _HEADCRE.search(trans[-1])
         if m and m.group("type") == "Xid":
         pos = Position(server_id, logfile, int(m.group("end_pos"))
         on_gid(server_id, trans_id, pos)
```

Код заключительного шага представлен в лист. 4-15. Функция promote_slave берет список подчиненных серверов, оставшихся без главного сервера, и выполняет повышение, выбирая из подчиненных новый главный сервер. И наконец, она подключает все остальные подчиненные серверы к повышенному серверу путем просмотра двоичных журналов. В коде задействована вспомогательная функция fetch_global_trans_id для извлечения глобального ID транзакции из заданной таблицы.

Лист. 4-15. Поиск нового главного сервера и подключение к нему всех подчиненных

```
def fetch_global_trans_id(slave):
    result = slave.sql("SELECT server_id, trans_id FROM Last_Exec_Trans")
    return (int(result["server_id"]), int(result["trans_id"]))

def promote_slave(slaves):
    slave_info = {}

   # Сбор глобальных ID транзакции каждого подчиненного сервера
   for slave in slaves:
       slave.connect()
```

```
        server_id, trans_id = fetch_global_trans_id(slave)
        slave_info.setdefault(trans_id, []).append((server_id, trans_id, slave))
        slave.disconnect()

# Выбор повышаемого сервера по наивысшему глобальному ID транзакций
new_master = slave_info[max(slave_info)].pop()[2]

def maybe_change_master(server_id, trans_id, position):
   from mysqlrep.utility import change_master
   try:
      for sid, tid, slave in slave_info[trans_id]:
         if slave is not new_master:
            change_master(slave, new_master, position)
   except KeyError:
      pass

#  Чтение файлов журнала на новом главном сервере.
new_master.connect()
logs = [row["Log_name"] for row in new_master.sql("SHOW MASTER LOGS")]
new_master.disconnect()

# Поочередное чтение файлов журнала на главном сервере в
# обратном порядке — начиная с новейшего файла.
logs.reverse()
for log in logs:
   scan_logfile(new_master, log, maybe_change_master)
```

Стоит отметить, что в коде подчиненные серверы собраны в словарь, ключом которого является ID, взятый из глобального ID транзакции. Так как с одним ключом может быть связано несколько подчиненных серверов, мы воспользовались готовым рецептом от Алекса Мартелли (Alex Martelli) из книги «Python Cookbook» изд-ва O'Reilly. Каждому ключу сопоставлен список серверов, что позволяет выполнить быстрый просмотр и обработку в функции maybe_change_master только по ID транзакции.

Код (лист. 4-15) не сортирует идентификаторы транзакции. Если последовательность имеет значение, понадобятся дополнительные меры. Порядок идентификаторов транзакции может быть нарушен, если одна транзакция извлекла глобальный ID, но была прервана до завершения другой транзакции, которая извлекла глобальный ID и завершилась. Чтобы идентификаторы транзакций соответствовали порядку начала транзакций, перед извлечением глобального ID транзакции добавьте инструкцию SELECT ... FOR UPDATE, изменив код следующим образом:

```
def commit_trans(cur):
   cur.execute("SELECT * FROM Last_Exec_Trans FOR UPDATE")
   cur.execute("SET SQL_LOG_BIN = 0")
      cur.execute(«INSERT INTO Global_Trans_ID() VALUES ()»)
      .
      .
   cur.commit()
```

Таким образом строка будет заблокирована до завершения транзакции. Это замедлит работу системы, что неоправданно, если порядок не важен.

Хранимые процедуры для завершения транзакций

Основной способ синхронизации серверов, показанный в этой главе, —реализация процедуры завершения транзакции в приложении. Это значит, что в код приложения нужно заложить имена таблиц и все хитрости по созданию и управлению глобальным ID транзакции. Поняв эти «хитрости», вы увидите, что все не так сложно, как кажется на первый взгляд. Зачастую, все сравнительно просто: в коде приложения создаются функции, которые программист вызывает в приложении, не вдаваясь в подробности.

Другой способ: включить код завершения транзакции в сервер БД при помощи хранимых процедур. В зависимости от ситуации, такая альтернатива может оказаться более подходящей. Так, для изменения процедуры завершения не обязательно изменять код приложения.

Для этого необходимо перенести ID транзакции из таблицы *Global_Trans_ID* и ID сервера в пользовательскую переменную или локальную переменную в хранимой процедуре. В зависимости от выбранного вами способа, в двоичном журнале запрос примет несколько иной вид.

Вероятность влияния локальных переменных на окружающий код меньше — пользовательские переменные «просачиваются» из хранимой процедуры.

Процедура завершения транзакции:

```
CREATE PROCEDURE commit_trans ()
   SQL SECURITY DEFINER
BEGIN
   DECLARE trans_id, server_id INT UNSIGNED;
   SET SQL_LOG_BIN = 0;
   INSERT INTO global_trans_id() values ();
   SELECT LAST_INSERT_ID() INTO trans_id,
         @@server_id INTO server_id;
   SET SQL_LOG_BIN = 1;
   INSERT INTO last_exec_trans(server_id, trans_id)
      VALUES (server_id, trans_id);
   COMMIT;
END
```

Завершение транзакции из кода приложения:

```
 CALL Commit_Trans();
```

Остается решить задачу по изменению процедуры поиска глобального ID транзакции в двоичном журнале. Как вызов данной функции выйдет в двоичном журнале? Итак, результат быстрого вызова mysqlbinlog:

```
#  at 1724
#091129 18:35:11 server id 1 end_log_pos 1899 Query thread_id=75
   exec_time=0 error_code=0
SET TIMESTAMP=1259516111/*!*/;
```

```
INSERT INTO last_exec_trans(server_id, trans_id)
   VALUES ( NAME_CONST('server_id',1), NAME_CONST('trans_id',13))
/*!*/;
#  at 1899
#091129 18:35:11 server id 1 end_log_pos 1926 Xid = 1444
COMMIT/*!*/;
```

Как видите, ID сервера и ID транзакции — оба хорошо видны в выводе. Как составить данную инструкцию при помощи регулярного выражения — ваше домашнее задание.

Круговая репликация

Вариант с двумя главными серверами может подать идею о возможности системы с несколькими главными серверами, где более двух «главных» реплицируют друг друга. У любого подчиненного сервера может быть только один главный сервер, поэтому единственный способ этого достичь — настроить репликацию по кругу. Это не рекомендуется, но вполне возможно. Не рекомендуют этот вариант потому, что очень сложно добиться его правильной работы при сбое. Скоро вы поймете, о чем речь.

Использование круговой репликации в системе с тремя и более серверами весьма полезно для обеспечения локальности данных. В качестве примера из жизни возьмем оператора сотовой связи с абонентами по всей Европе. Владельцы мобильных телефонов много путешествуют, поэтому реестр абонентов неплохо бы иметь поблизости к месту регистрации телефона в сети.

Разместив центры обработки данных в стратегических местах Европы, можно быстро проверять данные звонков и регистрировать новые звонки локально. Изменения можно реплицировать на все серверы в кольце, и постепенно точные учетные сведения появятся на всех серверах. В этом случае круговая репликация — безупречное решение: все данные абонента реплицируются на все сайты, а обновления данных разрешено во всех ЦОД.

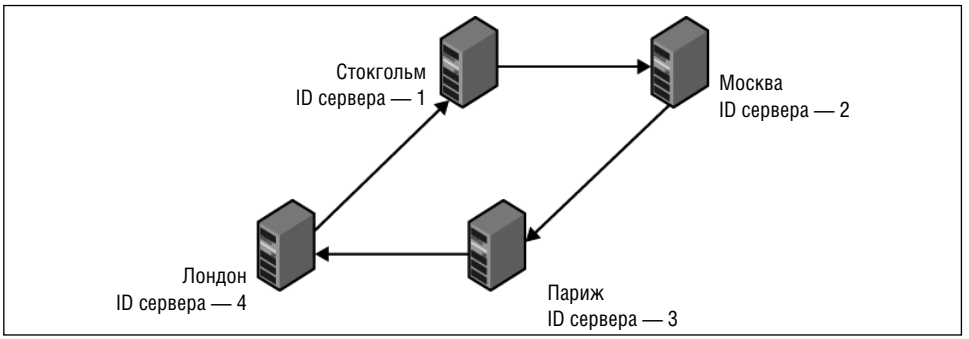


Рис. 4-10. Схема круговой репликации

Настройка круговой репликации (рис. 4-10) довольно проста. В лист. 4-16 представлен сценарий автоматической настройки круговой репликации.

Так в чем подвох? Как всегда, следует спросить себя: «Что будет, если что-нибудь пойдет не так?»

Лист. 4-16. Настройка круговой репликации

```
def circular_replication(server_list):
   count = len(server_list)
   for i in range(0, count):
      change_master(server_list[(i+1) % count], server_list[i])
```

На рис. 4-10 показаны четыре сервера, названные по именам городов, в которых они находятся (имена выбраны произвольно). Репликация идет по кругу: «Стокгольм» — «Москва» — «Париж» — «Лондон» — «Стокгольм». Таким образом, «Москва» — главный сервер для «Парижа» и подчиненный для «Стокгольма». Представьте себе, что «Москва» отключается неожиданно. Для продолжения репликации необходимо подключить подчиненный сервер «Париж» к главному серверу «Стокгольм».

На рис. 4-11 показан пример отказа одного сервера, и повторное подключение серверов для продолжения репликации. Вроде бы ничего сложного, но не спешите с выводами. Вам предстоит решить три основных вопроса:

- Подчиненный сервер, чей главный сервер перестал работать, должен подключиться к главному серверу и начать репликацию с последней увиденной им позиции. Как узнать эту позицию?

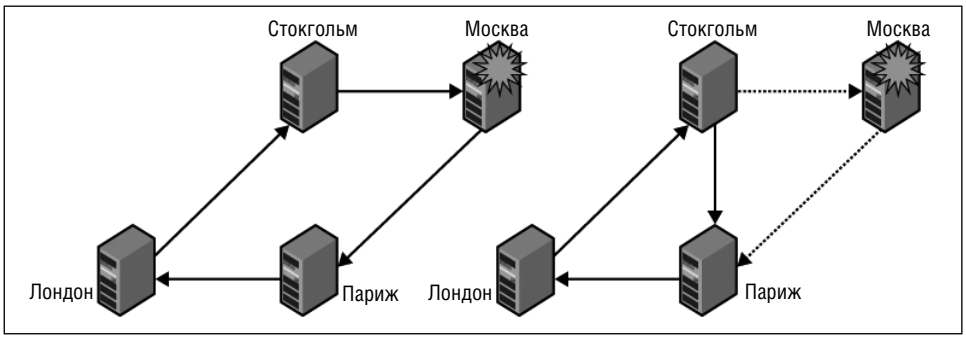


Рис. 4-11. Изменение топологии при отказе сервера

- Представьте себе, что сбойный сервер, перед тем как отключиться, сумел послать какие-то события. Что будет с этими событиями?
- Следует обдумать, как вернуть сбойный сервер обратно в топологию. Что, если сервер выполнил некоторые свои транзакции, записанные в двоичный журнал, но еще не отправленные? Ясно, что эти транзакции будут потеряны, и с этим надо что-то делать.

Когда обнаружено отключение одного из серверов, нужно подключить подчиненный сервер к главному с помощью команды CHANGE MASTER несложно. Однако для корректной работы репликации мы должны определить правильную позицию в двоичном журнале. Для ее поиска использу-

ются аналогичные методы просмотра двоичного журнала, что и при повышении сервера. В уже знакомой нам таблице *Last_Exec_Trans* содержатся ID сервера и глобальный ID транзакции, видимый с данного сервера.

Вторая проблема будет посложнее. Если сервер во время сбоя смог послать событие, ничто не сможет «выдернуть» это событие из потока репликации — оно так и будет циркулировать по топологии репликации. Если инструкция идемпотентна, и ее многократное повторение не создаст проблем, то на какое-то время ситуация останется под контролем, но вообще, инструкцию нужно удалить тем или иным способом.

В MySQL 5.5 в команду CHANGE MASTER добавлен параметр IGNORE_SERVER_IDS. Данный параметр позволяет серверу удалять из потока репликации не только события, ID сервера которых совпадают с сервером, но и другие события. Итак, ID серверов показаны на рис. 4-11, тогда, чтобы подключить сервер «Париж» к серверу «Стокгольм», нужно выполнить следующую команду:

```
paris> CHANGE MASTER TO
    ->  MASTER_HOST='stockholm.example.com',
    ->  IGNORE_SERVER_IDS = (2);
```

В предыдущих версиях MySQL (до 5.5) такой поддержки нет, и вы должны придумать другие способы удаления ошибочного события. Наверное, самый простой из них — это временно подключить сервер, чей ID сервера идентичен ID сбойного сервера, с единственной целью — удалить ошибочное событие.

Полная процедура сужения кольца в круговой установке с использованием MySQL 5.5:

1. Определите глобальные ID последних завершенных транзакций на всех оставшихся в работе подчиненных серверах.

```
paris> SELECT Server_ID, Trans_ID FROM Last_Exec_Trans WHERE Server_ID != 2;
+-----------+----------+
| Server_ID | Trans_ID |
+-----------+----------+
| 1         | 5768     |
| 3         | 4563     |
| 4         | 768      |
+-----------+----------+
3 rows in set (0.00 sec)
```

2. В двоичном журнале главного сервера найдите последний глобальный ID транзакции в таблице *Last_Exec_Trans*.
3. Подключите подчиненный сервер с этой позиции при помощи команды CHANGE MASTER.

```
paris> CHANGE MASTER TO
    ->  MASTER_HOST='stockholm.example.com',
    ->  IGNORE_SERVER_IDS = (2);
```

В нашем «альтернативном будущем» все серверы одинаковые, поэтому самый простой способ вернуть отказавший сервер в круг — это восстановить на сервере один из серверов кольца и подключить кольцо заново, вернув в него новый сервер. Вот как это сделать:

4. Восстановите сервер из копии одного из существующих серверов (будущего главного сервера) кольца и подключите его к этому серверу в качестве подчиненного сервера.

```
moscow> CHANGE MASTER TO MASTER_HOST='stockholm.example.com';
Query OK, 0 rows affected (0.18 sec)

moscow> START SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

5. После того, как сервер полностью «догонит» главный сервер, разорвите кольцо, отключив подчиненный сервер. Этот сервер больше не будет получать обновления.

```
paris> STOP SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

6. На восстановленном сервере могут оказаться не все события, которые есть на подчиненном, поэтому необходимо подождать, пока на восстановленном сервере соберутся, по меньшей мере, все события, имеющиеся на подчиненном сервере. Так как позиции представлены для одного и того же сервера, сделать это можно при помощи комбинации команд SHOW SLAVE STATUS и MASTER_POS_WAIT.

```
paris> SHOW SLAVE STATUS;
      ...
   Relay_Master_Log_File: stockholm-bin.000096
      ...
   Exec_Master_Log_Pos: 756648
1 row in set (0.00 sec)

moscow> SELECT MASTER_POS_WAIT('stockholm-bin.000096', 756648);
+-----------------------------------------------+
| MASTER_POS_WAIT('stockholm-bin.000096', 756648) |
+-----------------------------------------------+
|                                        985761 |
+-----------------------------------------------+
1 row in set (156.32 sec)
```

7. На восстановленном сервере определите позицию события, найдя в его двоичном журнале последний «увиденный» подчиненным сервером глобальный ID.
8. Подключите подчиненный сервер к восстановленному и запустите репликацию.

```
paris> CHANGE MASTER TO
    ->  MASTER_HOST='moscow.example.com',
    ->  MASTER_LOG_FILE='moscow-bin.000107',
    ->  MASTER_LOG_POS=196758,
Query OK, 0 rows affected (0.18 sec)

moscow> START SLAVE;
Query OK, 0 rows affected (0.00 sec)
```

Заключение

Практическое воплощение высокой доступности не тривиально. В этой главе мы рассмотрели высокую доступность и способы ее достижения в MySQL. В следующей главе знакомство с высокой доступностью будет продолжено в аспекте масштабирования.

> Джоэл услышал сигнал: пришла новая почта. Он открыл последнее сообщение. Оно было от Саммерсона — комментарии к его отчету. В конце Джоэл нашел то, чего так ждал: «Мне понравились идеи насчет избыточности, особенно стратегия горячего резерва. Приступай к реализации».
>
> Джоэл с сожалением подумал о том, что знакомство с некоторыми коллегами, которое он было запланировал, придется отложить — будет не до того...

ГЛАВА 5

Роль репликации MySQL в горизонтальном масштабировании

Джоэл встал, потянулся и решил, что пора глотнуть газировки. Он вылез из-за стола и направился в комнату отдыха, но у двери его перехватил босс.

— Добрый день, сэр.

— Привет, Джоэл. Мы только что продали уйму лицензий на наши новые продукты. Ребята из маркетингового отдела сказали, что следует ожидать как минимум десятикратного возрастания загруженности наших серверов баз данных.

Джоэл поднял брови. Не далее как на прошлой неделе он добавил одиночный подчиненный сервер, что несколько улучшило ситуацию, но полностью проблема пока не была решена.

— Джоэл, придется прибегнуть к горизонтальному масштабированию.

— Конечно, сэр. Займусь этим прямо сейчас.

Улыбнувшись, г-н Саммерсон похлопал Джоэла по плечу и удалился в свой офис.

Джоэл же лихорадочно пытался понять, что за зверь такой «горизонтальное масштабирование», и выработать план. «Придется еще посидеть над книгой», — пробормотал он, перед тем как все-таки отправиться в комнату отдыха.

Когда нагрузка начинает расти, — а если ваша ИТ-система успешно работает, это *вопрос времени*, — у вас есть два пути. Первый — постоянно заменять свои серверы более мощными, способными справиться с возрастающей нагрузкой, такой путь называется *вертикальным масштабированием* (*scaling up*). Второй — наращивать число серверов, которые сообща должны справиться с возрастающей нагрузкой, это называется *горизонтальным масштабированием* (*scaling out*). Горизонтальное масштабирование используют значительно чаще, поскольку для него достаточно купить несколько стандартных недорогих серверов.

Дополнительные серверы не только решают проблему роста загруженности системы, но также обеспечивают высокую доступность и соответствие нормативным требованиям. Критерием эффективности горизонтального масштабирования является рациональное использование объединенных ресурсов (к примеру, вычислительной мощности) всех серверов.

В этой главе мы не будем касаться таких аспектов масштабирования, как аппаратное обеспечение и сети — их рассмотрение не входит в задачи этой книги. К тому же эта тематика достаточно раскрыта в книге Бэрона Шварца с соавторами[1]. Мы же поговорим о том, как настроить репликацию MySQL, чтобы извлечь из горизонтального масштабирования максимальную выгоду. Вкратце обсудив основы репликации, приступим к разработке библиотеки на языке программирования Python, призванной облегчить администрирование репликации системы с большим количеством серверов. Наконец, мы проанализируем соответствие репликации требованиям бизнеса применительно к вашей организации.

Ниже перечислены задачи, для решения которых чаще всего обращаются к горизонтальному масштабированию и репликации:

- *Балансировка нагрузки при чтении данных* Раз главный сервер занят обновлением данных, имеет смысл подключить отдельные серверы для поддержки запросов на выборку. Поскольку запрос на выборку требует только чтения данных, можно воспользоваться репликацией для передачи изменений с главного сервера на подчиненные (которых может быть столько, сколько, по вашему мнению, требуется). Таким образом, подчиненные серверы смогут обрабатывать запросы на выборку с использованием актуальных данных.
- *Балансировка нагрузки при записи данных* Развертывания с солидным трафиком распределяют выполнение операций среди большого количества компьютеров (иногда среди нескольких тысяч). В этом случае репликация играет ключевую роль в распределении информации, подлежащей обработке. Существует много способов распределения информации в зависимости от назначения данных и от природы их использования:
  - Распределение может основываться на роли, которую играет та или иная информация в функционировании системы. Редко обновляемые таблицы лучше оставить на одиночном сервере, а таблицы, требующие частого обновления, секционируются по нескольким серверам.
  - Данные можно секционировать в зависимости от географического местоположения, что позволит направлять трафик ближайшему из серверов.
- *Горячий резерв* Если «упадет» главный сервер, то остановится все: не будет возможности обрабатывать транзакции (может быть, критически важные), получать сведения о клиентах или извлекать необходимые данные. Подобного сценария нужно избежать (почти) любой ценой, так как это может серьезно подорвать бизнес. Самый простой способ: настроить подчиненный сервер, единственная цель которого — быть в горячем резерве, готовым взять на себя функции главного сервера, если тот выйдет из строя.

---

[1] Baron Schwartz et al. *High Performance MySQL* (O'Reilly, *http://oreilly.com/catalog/9780596101718*).

- *Удаленная репликация* Любое развертывание подвержено риску выхода из строя центра обработки данных в результате катастрофы, будь то отключение электроэнергии, землетрясение или наводнение. Смягчить последствия такого события позволит репликация, передающая информацию между географически удаленными точками.
- *Выполнение резервного копирования* Многие держат под рукой дополнительный сервер для выполнения резервного копирования. Это позволяет организовать резервное копирование таким образом, чтобы главный сервер вообще не затрагивался, поскольку имеется возможность перевести резервный сервер в автономный режим и выполнить любые требуемые операции.
- *Создание отчетов* Создание отчетов на основе хранящихся на сервере данных снизит производительность сервера, в некоторых случаях, значительно. Если при создании отчетов выполняется много фоновых заданий, то для этих целей лучше создать дополнительный сервер. В заданное время останавливается репликация, и на сервере отчетов создается снимок БД, после чего «громоздкие» запросы можно адресовать ему, а не главному серверу предприятия. К примеру, если вы остановите репликацию после завершения последней транзакции дня, то получите отчет за день, в то время как остальная деятельность предприятия будет идти в обычном ритме.
- *Фильтрация или секционирование данных* Если сетевое подключение медленное или же необходимо отказать определенным клиентам в доступе к отдельным данным, имеет смысл добавить сервер, обеспечивающий фильтрацию данных. Также подобный вариант окажется полезным, если данные требуется секционировать и разместить на отдельные серверах.

Горизонтальное масштабирование операций чтения, а не записи

Важно осознавать, что варианты горизонтального масштабирования, описанные выше, влияют только на чтение данных, но не на запись. Для каждого вновь добавленного в систему подчиненного сервера нагрузка по операциям записи не будет отличаться от нагрузки по записи для главного сервера. Следующая формула представляет среднюю нагрузку системы:

$$СредняяНагрузка = \frac{\Sigma\ НагрузкаПоЧтению + \Sigma\ НагрузкаПоЗаписи}{Производительность}$$

Возьмем одиночный сервер с общей производительностью (или, как еще говорят, пропускной способностью) 10 000 транзакций в секунду и допустим, что нагрузка по записи составляет 4 000 транзакций в секунду, а нагрузка по чтению — 6 000 транзакций в секунду. Тогда средняя нагрузка будет равна:

$$СредняяНагрузка = \frac{\Sigma\ НагрузкаПоЧтению + \Sigma\ НагрузкаПоЗаписи}{Производительность} = \frac{6000 + 4000}{10000} = 100\%$$

Если добавить к главному серверу три подчиненных, общая производительность возрастет до 40 000 транзакций в секунду. Поскольку все запросы на запись реплицируются, каждый такой запрос выполняется по четыре раза, — один раз на главном сервере и по одному разу на каждом из трех подчиненных, — и значит, нагрузка на любой из подчиненных серверов по записи составит 4 000 транзакций в секунду. Однако общая нагрузка по чтению не возросла, а просто распределилась по подчиненным серверам. Теперь средняя нагрузка считается так:

$$СредняяНагрузка = \frac{\Sigma\ НагрузкаПоЧтению + \Sigma\ НагрузкаПоЗаписи}{Производительность} = \frac{6000 + 4 \times 4000}{4 \times 10000} = 55\%$$

Заметьте, что согласно формуле производительность увеличивается в четыре раза, поскольку общее количество серверов равно четырем. Вследствие репликации нагрузка по записи вырастает также в четыре раза.

О том, что при репликации все запросы на запись, поступающие главному серверу, передаются и подчиненным серверам, часто забывают. Очевидно, что нельзя подойти к масштабированию операций записи так же просто, как и к масштабированию операций чтения. Далее в этой главе будет обсуждаться масштабирование записи данных при помощи так называемого *шардинга* (sharding).

Смысл асинхронной репликации

Репликация, реализованная в MySQL, является *асинхронной*, — подобная разновидность репликации лучше всего подходит для современных приложений, например веб-сайтов.

Чтобы справиться с большим объемом операций чтения, в рамках репликации приложений для сайтов создаются копии главного сервера, затем все запросы на чтение данных передаются подчиненным серверам, в то время как главный сервер обрабатывает запросы на запись. Такая репликация считается асинхронной, поскольку главный сервер только рассылает запросы на каждое изменение подчиненным серверам, не дожидаясь, пока они примут и обработают их, а лишь предполагая, что в конце концов все подчиненные серверы получат все запросы и изменения будут реплицированы. Такой способ повышения производительности обычно неплохо подходит для целей горизонтального масштабирования.

При синхронной репликации, напротив, главный сервер синхронизируется с подчиненными и не фиксирует транзакцию до тех пор, пока не убедится, что подчиненные серверы готовы ее фиксировать. Таким образом, при синхронной репликации главному серверу приходится ждать, пока все подчиненные серверы не разберутся с операциями записи.

Из вышесказанного с очевидностью следует, что асинхронная репликация намного быстрее, чем синхронная. В отличие от асинхронной репликации, синхронная требует дополнительной синхронизации, позволяющей обеспечить согласованность данных. Обычно для реализации синхронной репликации используется протокол *двухфазной фиксации*, поддерживающий согласованность данных на главном и на подчиненных серверах за счет дополнительного обмена сообщениями между ними. В общем случае это работает примерно так:

1. При выполнении оператора фиксации транзакция рассылается подчиненным серверам, которым также передается и запрос на подготовку к фиксации.
2. Каждый подчиненный сервер обрабатывает транзакцию, готовя ее к фиксации, а затем посылает главному серверу сообщение OK (или ABORT), уведомляя главный сервер о том, что транзакция готова (или напротив, должна быть прервана).
3. Главный сервер дожидается, пока все подчиненные серверы пошлют ему одно из двух сообщений — OK или ABORT.
   а. Если главный сервер получает сообщение OK от всех подчиненных серверов, он рассылает подчиненным серверам указание о фиксации данной транзакции.
   б. Если главный север получает сообщение ABORT хотя бы от одного подчиненного сервера, он рассылает подчиненным серверам указание о прерывании данной транзакции.
4. Все подчиненные серверы дожидаются, пока главный сервер пошлет им одно из двух сообщений — OK или ABORT.
   а. Если подчиненные серверы получают запрос на фиксацию транзакции, они фиксируют транзакцию и посылают главному серверу уведомление о том, что транзакция фиксирована.
   б. Если подчиненные серверы получают запрос на прерывание транзакции, они отменяют транзакцию, откатив все изменения и освободив все захваченные ранее ресурсы, затем посылают главному серверу уведомление о том, что транзакция прервана.
5. Когда главный сервер получает уведомления от всех подчиненных серверов, он докладывает, что транзакция фиксирована (или же прервана) и приступает к обработке следующей транзакции.

Этот протокол работает медленно, поскольку он требует передачи четырех сообщений, включая сообщение с транзакцией и запрос на подготовку. Главной проблемой является не объем сетевого трафика, необходимого для обеспечения синхронизации, а задержка сети и задержка, вызванная фиксацией транзакции на подчиненных серверах, вкупе с блокировкой транзакции на главном сервере, накладываемой до поступления уведомлений от всех подчиненных серверов. Напротив, асинхронной репликации требуется лишь одно сообщение, посылаемое с транзакцией. Дополнительным плюсом является то,

что главный сервер не ждет подчиненных, а сразу докладывает о фиксации транзакции, что значительно улучшает быстродействие протокола.

Но почему устанавливаемая при синхронной репликации блокировка транзакции на время ее фиксации подчиненными серверами представляет проблему? Если подчиненные серверы расположены в сети близко к главному, те дополнительные сообщения, которые требуются для синхронной репликации, не окажут существенного влияния на производительность. Но совсем другое дело, когда подчиненные серверы находятся далеко, — в другом городе или даже на другом континенте.

В табл. 5-1 приведены примерные раскладки для сервера, фиксирующего 10 000 транзакций в секунду. Соответственно, время фиксации одной транзакции составляет 0,1 мс (хотя следует учесть, что некоторые реализации, например MySQL Cluster, способны фиксировать несколько независимых транзакций параллельно). Если задержка сети составляет 0,01 мс (это базовое значение, полученное ping-ом одного из компьютеров локальной сети), время фиксации транзакции вырастает до 0,14 мс, что при пересчете дает приблизительно 7 000 транзакций в секунду. Если задержка сети составляет 10 мс (это значение мы получили, протестировав связь с сервером в соседнем городе), время фиксации транзакции становится равным уже 40,1 мс, что соответствует 25 транзакциям в секунду! Напротив, асинхронная репликация вовсе не увеличивает задержку, поскольку фиксация транзакции подтверждается моментально, и таким образом показатель остается исходным — 10 000 транзакций в секунду, — как если бы подчиненные сервера вовсе не участвовали в процессе.

Табл. 5-1. Типичные задержки при синхронной репликации

Задержка (мс)	Время фиксации транзакции (мс)	Число транзакций в сек.	Тип системы
0,01	0,14	~7 100	На одном компьютере
0,1	0,5	~2 000	Небольшая LAN
1	4,1	~240	Крупная LAN
10	40,1	~25	Сеть уровня крупного города
100	400,1	~2	Сеть со спутниковыми каналами

Высокая производительность асинхронной репликации достигается посредством понижения уровня требований к согласованности данных. Вспомним, что при асинхронной репликации главный сервер докладывает о фиксации транзакции немедленно, *не* дожидаясь подтверждения от подчиненных серверов. Это означает, что главный сервер может считать транзакцию зафиксированной, а подчиненный сервер — нет. По сути, такая транзак-

ция может даже не выйти за пределы главного сервера, а только дожидаться отправки на подчиненный.

Поэтому не следует забывать о потенциальных проблемах асинхронной репликации, а именно:

- В случае сбоя главного сервера транзакции могут «исчезнуть».
- Запрос, выполненный на подчиненных серверах, может вернуть устаревшие данные.

Позднее мы обсудим, как гарантировать актуальность читаемых данных, но на данный момент необходимо осознать, что асинхронная репликации имеет теневые стороны, которые нельзя игнорировать.

Управление топологией репликации

Развертывание масштабируется посредством создания подчиненных серверов и добавления их в систему. Термин *топология репликации* описывает варианты подключения серверов, участвующих в репликации. На рис. 5-1 приведены примеры топологий репликации: простая топология, топология дерева, двунаправленная топология и круговая топология.

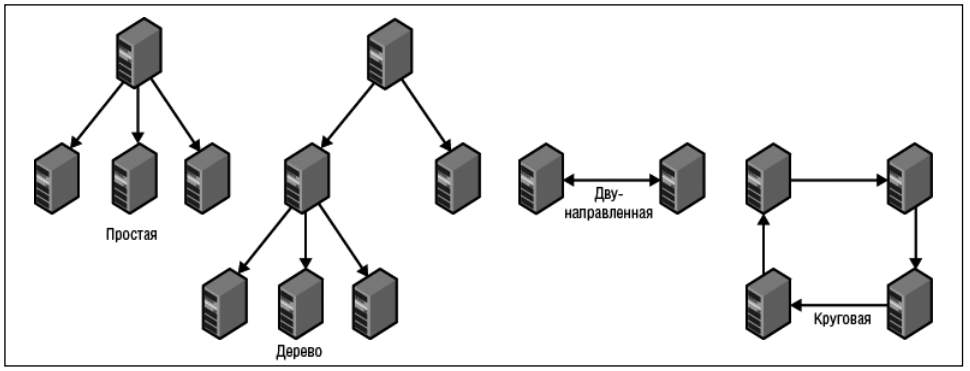


Рис. 5-1. Простая, дерево, двунаправленная и круговая топологии репликации

Каждая топология имеет свое назначение: двунаправленная топология, например, удобна для повышения отказоустойчивости, а круговая репликация, как и двунаправленная, позволяет одним сайтам функционировать автономно и реплицировать изменения на других сайтах.

Простая топология и топология дерева применяются для горизонтального масштабирования. Использование репликации ведет к значительному увеличению количества операций чтения по сравнению с операциями записи. Вследствие этого на развертывание накладываются два специальных требования:

- *Требуется балансировка нагрузки* Термин *балансировка нагрузки* здесь обозначает распределение запросов по серверам. Репликация не только создает основания для балансировки нагрузки, но и предоставляет способы ее выполнения. Начнем с того, что репликация предполагает первич-

ное распределение нагрузки, когда запись данных выполняется главными серверами, а чтение — подчиненными. К тому же, иногда необходимо отправлять отдельный взятый запрос на конкретный подчиненный сервер.

- *Требуется управление топологией* Рано или поздно серверы выходят из строя, и их приходится заменять. Замену подчиненного сервера не обязательно выполнять в экстренном порядке, но отказавший главный сервер следует заменить очень быстро.

  К тому же, при сбое главного сервера необходимо перенаправить клиентов на новый главный сервер. При отказе подчиненного сервера достаточно исключить его из пула балансировщиков нагрузки, чтобы запросы к нему не отправлялись.

Чтобы обеспечить балансировку и контроль нагрузки, необходимо правильно подобрать и настроить инструментарий для управления топологией репликации. Особенно важны инструменты мониторинга состояния и производительности серверов, а также обеспечение, поддерживающее распределение запросов.

Балансировка нагрузки будет эффективна только в случае наличия у серверов резервных мощностей. Ниже перечислены ситуации, в которых наличие резервных мощностей необходимо:

- *Поддержка пиковой нагрузки* Чтобы справиться с пиковыми нагрузками, необходимо иметь «пространство для маневров». Нагрузка на систему не бывает стабильной, она колеблется — растет и уменьшается. Объем требуемой для крупного развертывания резервной мощности в каждом случае свой, поэтому необходимо тщательно отслеживать тот момент, когда время отклика выйдет за пределы допустимых значений.
- *Цена распределения* Чтобы настроить репликацию, необходимо иметь резервные мощности. Репликация всегда приводит к некоторой «потере» производительности вследствие накладных расходов, присущих распределенным системам. Для управления распределенной системой служат дополнительные запросы, например для уточнения, какой из серверов произведет выборку данных.

  Тот факт, что любой подчиненный сервер должен выполнить все те же операции записи, которые выполняет главный сервер, легко ускользает из памяти. Запросы от главного сервера выполняются последовательно, поэтому конфликт обновлений исключен, но для репликации подчиненным серверам понадобятся резервные мощности.
- *Задачи администрирования* Реструктурирование конфигурации репликации требует наличия резервных мощностей, позволяющих временно поменять местами главный и подчиненные серверы (в этом состоит двунаправленность), например, для переноса данных между серверами.

Можно выделить два основных способа организации балансировки нагрузки: либо само приложение запрашивает определенный сервер в зависимости от типа запроса, либо некий промежуточный слой — обычно опреде-

ляемый как *прокси* (*proxy*) — анализирует запрос и отсылает его подходящему серверу.

Подход с использованием промежуточного слоя для анализа и распределения запросов (см. рис. 5-2) гораздо более гибок, но ему присущи два недостатка:

- Ресурсы, предназначенные для обработки, расходуются на анализ запросов. Запрос обрабатывается дольше, поскольку разбор и анализ выполняются дважды — в первый раз прокси и повторно сервером MySQL. Чем анализ глубже, тем больше задерживается запрос. Насколько эта проблема серьезна, зависит от конкретного приложения.

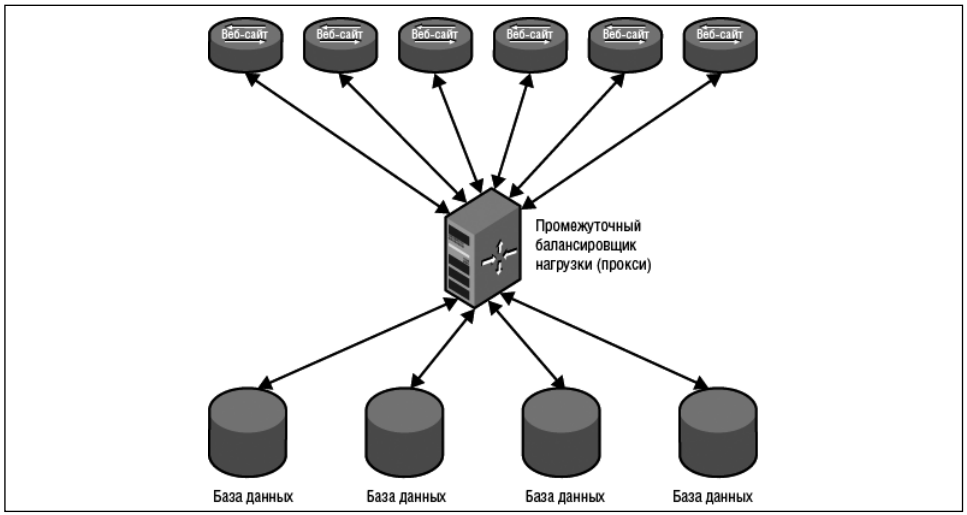


Рис. 5-2. Использование прокси для распределения запросов

- Корректный анализ запроса реализовать сложно, иногда почти невозможно. Прокси часто скрывает внутреннюю структуру развертывания от прикладного программиста, который, таким образом, освобождается от сложного выбора. Поэтому клиент может послать запрос, с трудом поддающийся анализу и требующий переработки до отправки на серверы.

Один из инструментов, предлагаемых для балансировки нагрузки с участием прокси, — MySQL Proxy. Он содержит полную реализацию клиентского протокола MySQL, вследствие чего может выступать в роли сервера, если к нему подключится настоящий клиент, и в роли клиента, соединяясь с сервером MySQL. Это означает, что прокси полностью прозрачен — клиент не отличит его от «настоящего» сервера.

Для управления MySQL Proxy применяется язык программирования Lua. MySQL Proxy оснащен встроенным интерпретатором этого языка, выполняющим маленькие (а иногда и не очень маленькие) программы, призванные перехватывать и видоизменять как запросы, так и результирующие наборы. Поскольку для работы с прокси используется полноценный язык

программирования, прокси способен решать разнообразные нетривиальные задачи. В их число входят анализ, фильтрация, переработка и распределение запросов.

Конфигурирование MySQL Proxy и программирование для него выходят за рамки этой книги, но в Интернете вы найдете подробные материалы. Ниже приведены некоторые полезные ссылки:

- *http://dev.mysql.com/tech-resources/articles/proxy-gettingstarted.html* «Приступая к работе с MySQL Proxy» — классическая статья Джузеппе Максиа (Giuseppe Maxia), подходящая для знакомства с этим инструментом.
- *http://forge.mysql.com/wiki/MySQL_Proxy* Вики-страница на сайте MySQL Forge — информация о MySQL Proxy, в том числе множество ссылок и примеров[2].
- *http://forge.mysql.com/wiki/MySQL_Proxy_RW_Splitting* Вики-страница на сайте MySQL Forge, описывающая алгоритм разделения чтения и записи посредством MySQL Proxy, который заключается в отправке запросов чтения некоторому подмножеству серверов, а запросов записи — главному серверу.

Конкретные приемы работы с прокси полностью зависят от его разновидности, поэтому здесь мы не будем на них останавливаться. Напротив, мы подробно обсудим балансировщик нагрузки уровня приложения. Существуют различные варианты балансировщика нагрузки, в том числе:

- Аппаратные балансировщики.
- Несложные балансировщики нагрузки, реализованные программно, например Balance (*http://www.inlab.de/balance.html*).
- Пиринговые (peer-based) системы, например Wackamole (*http://www.backhand.org/wackamole*).
- Полномасштабные кластерные решения, например Linux Virtual Server (*http://www.li nuxvirtualserver.org*).

Можно также выполнять распределение нагрузки на уровне DNS или реализовать поддержку распределения непосредственно в приложении.

Пример балансировщика нагрузки уровня приложения

Вот подходящая для нас задача — спроектировать и реализовать простой балансировщик нагрузки уровня приложения, чтобы понять, как он работает. В этом разделе мы реализуем разделение чтения и записи. По ходу в наш балансировщик будет добавлена поддержка секционирования данных.

Самым простым вариантом можно считать приложение, получающее от балансировщика нагрузки то или иное подключение в зависимости от типа

---

[2] Адрес актуальной страницы: *https://launchpad.net/mysql-proxy.*

запроса, подготовленного к отправке. В большинстве случаев приложение заранее знает, будет ли запрос читать данные или записывать их, какие таблицы он затронет. На самом деле, если разработчик решит так организовать приложение, это может дать и другие преимущества, например общее повышение производительности системы. Основываясь на информации о запросе, балансировщик нагрузки будет поставлять приложению подключение к подходящему серверу, который и выполнит этот запрос.

Балансировщику нагрузки на уровне приложения необходимо центральное хранилище, содержащее информацию о серверах и о запросах, на которых эти сервера специализируются. Функционал уровня приложения посылает запросы в центральное хранилище, а получает имя или IP-адрес подходящего для выполнения конкретного запроса сервера MySQL.

Итак, создадим простой балансировщик нагрузки уровня приложения (см. рис. 5-3.) Поскольку в программировании для веб-серверов часто используется язык PHP, распишем логику программы на нем. Нам придется разработать функции для обновления информации о пуле серверов и для получения информации о серверах из пула.

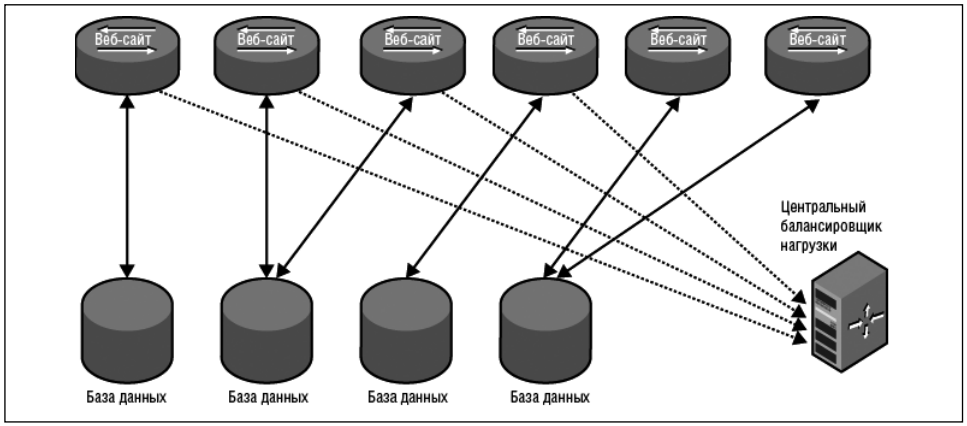


Рис. 5-3. Балансировка нагрузки на уровне приложения

Для реализации пула в общую для всех узлов базу данных добавляется таблица с перечнем всех серверов развертывания. В нашем случае сделаем первичным ключом таблицы хост и порт (вместо идентификатора хоста) и создадим базу данных *common*, содержащую таблицы с общими данными.

Следует дублировать центральное хранилище, чтобы не получить критичную точку, сбой в которой выведет всю систему из строя. К тому же, поскольку список доступных серверов меняется нечасто, информация о балансировке нагрузки является превосходным кандидатом на кэширование.

Чтобы не усложнять демонстрацию и к тому же не привлекать другие системы, наш балансировщик нагрузки на уровне приложения будет использовать только чистый MySQL.

Существуют другие технологии, позволяющие обойтись без MySQL. Наиболее распространено применение циклической DNS (DNS-карусели), альтернативный вариант

предполагает использование программы типа Memcached, хранящей в памяти хэш-таблицу значений, ассоциированных с ключами.

Заметьте также, что следует избегать дополнительных запросов, поскольку их использование ведет к существенной перегрузке высокопроизводительной системы.

Балансировщик нагрузки перечисляет серверы в пуле балансировки нагрузки, разбивая их на категории в зависимости от типа поддерживаемых запросов. Информация о серверах в пуле хранится в центральном репозитории. Наша реализация включает в себя таблицу в общей базе данных (см. определение в лист. 5-1), PHP-функции, предназначенные для отправки из приложения запросов к балансировщику нагрузки (лист. 5-2), а также функции на языке Python, обновляющие информацию о серверах, — их вы найдете в лист. 5-3.

Лист. 5-1. Таблицы базы данных, используемые балансировщиком нагрузки

```
CREATE TABLE nodes (
   host CHAR(28) NOT NULL,
   port INT UNSIGNED NOT NULL,
   sock CHAR(64) NOT NULL,
   type SET('READ','WRITE') NOT NULL DEFAULT '',
   PRIMARY KEY (host, port)
);
```

Независимо от того, принимает ли хост запросы на чтение, на запись, на чтение и на запись или не принимает никаких запросов, хост заносится в таблицу. Информация о типе поддерживаемых сервером запросов хранится в поле type. Задание пустого набора в качестве значения этого поля позволит перевести сервер в автономный режим, что важно для обслуживания системы.

Достаточно элементарного SELECT, чтобы отобрать все серверы, способные обработать определенный запрос. Поскольку нам нужен только один сервер, ограничим выборку единственной строкой при помощи модификатора LIMIT. Чтобы запросы распределялись по доступным серверам равномерно, воспользуемся также модификатором ORDER BY RAND().

Если используется модификатор ORDER BY RAND(), сервер сортирует строки таблицы. Это не самый эффективный (вернее, очень неудачный) способ получения случайного значения, но мы прибегнем к этому приему для наглядности.

В лист. 5-2 показана PHP-функция getServerConnection, которая запрашивает сервер и подключается к нему. Эта функция возвращает подключение к подходящему для выполнения запроса серверу, или же NULL, если нужный сервер не найден. Вспомогательная функция connect_to формирует строку подключения в соответствии с хостом, портом и сокетом Unix. Если в качестве хоста выступает local host, для подключения к серверу будет использоваться сокет — это более эффективно.

Лист. 5-2. PHP-функция, запрашивающая сервер у балансировщика нагрузки

```
function connect_to($host, $port, $socket) {
   $db_server = $host == "localhost" ? ":{$socket}" : "{$host}:{$port}";
   return mysql_connect($db_server, 'query_user');
}

$COMMON = connect_to(host, port, socket);
mysql_select_db('common', $COMMON);

define('DB_WRITE', 'WRITE');
define('DB_READ', 'READ');

function getServerConnection($queryType)
{
   global $COMMON;
   $query = <<<END_OF_SQL
SELECT host, port, sock FROM nodes
   WHERE FIND_IN_SET('$queryType', type)
ORDER BY RAND() LIMIT 1
END_OF_SQL;
   $result = mysql_query($query, $COMMON);
   if ($row = mysql_fetch_row($result))
      return connect_to($row[0], $row[1], $row[2]);
   return NULL;
}
```

В завершение этого этапа нам придется разработать утилиту для добавления и удаления серверов, а также для корректировки информации о назначении сервера. Подобные задачи обычно решает администратор, поэтому реализуем утилиту на языке Python с использованием библиотеки Replicant. Утилита включает три функции:

- *pool_add(common, server, type)* Добавляет сервер, заданный параметром server, к пулу. Пул хранится на сервере, передаваемом параметром common, и в качестве параметра type используется список (или другое перечисление) устанавливаемых значений.
- *pool_del(common, server)* Удаляет сервер, заданный параметром server, из пула.
- *pool_set(common, server, type)* Изменяет тип сервера, заданного параметром server.

Лист. 5-3. Функции утилиты, предназначенные для балансировщика нагрузки

```
class AlreadyInPoolError(replicant.Error):
   pass

_INSERT_SERVER = """
INSERT INTO nodes(host, port, sock, type)
   VALUES (%s, %s, %s, %s)"""
```

```
_DELETE_SERVER = "DELETE FROM nodes WHERE host = %s AND port = %s"

_UPDATE_SERVER = "UPDATE nodes SET type = %s WHERE host = %s AND port = %s"

def pool_add(common, server, type=[]):
   common.use("common")
   try:
      common.sql(_INSERT_SERVER,
            (server.host, server.port, server.socket, ','.join(type)));
   except MySQLdb.IntegrityError:
      raise AlreadyInPoolError

def pool_del(common, server):
   common.use("common")
   common.sql(_DELETE_SERVER, (server.host, server.port))

def pool_set(common, server, type):
   common.use("common")
   common.sql(_UPDATE_SERVER, (','.join(type), server.host, server.port))
```

Вызывать эти функции можно, к примеру, так:

```
pool_add(common, master, ['READ', 'WRITE'])

for slave in slaves:
   pool_add(common, slave, ['READ'])
```

Иерархическая репликация

Хотя главный сервер обладает мощностью, достаточной для поддержки множества подчиненных серверов, все-таки она не беспредельна. Рано или поздно нагрузка станет слишком высокой для нормальной работы (некий пользователь упомянул 70 подчиненных серверов как установленный опытным путем предел, но вы ведь понимаете, что этот предел сильно зависит от конкретного приложения), а переставший отвечать на запросы главный сервер — это всегда большая проблему. В такой ситуации имеет смысл добавить один или несколько дополнительных подчиненных серверов в качестве *подчиненных серверов-ретрансляторов* (или просто *ретрансляторов*), предназначенных исключительно для снижения нагрузки от репликации на главный сервер, — ретрансляторы возьмут на себя коммуникацию с частью подчиненных серверов. Подобное использование ретрансляторов носит название *иерархической репликации*. На рис. 5-4 представлен стандартный набор из главного сервера, ретранслятора и нескольких подчиненных серверов, подключенных к ретранслятору.

По умолчанию изменения, получаемые подчиненным сервером от главного, не заносятся в его двоичный журнал, поэтому при выполнении команды SHOW BINLOG EVENTS на подчиненном сервере из конфигурации, представленной

выше, событий в двоичном журнале не окажется. Дело в том, что не имеет смысла расходовать дисковое пространство, регистрируя такие изменения, — если произойдет сбой или даже отказ подчиненного сервера, для восстановления достаточно будет клонировать главный сервер или другой подчиненный.

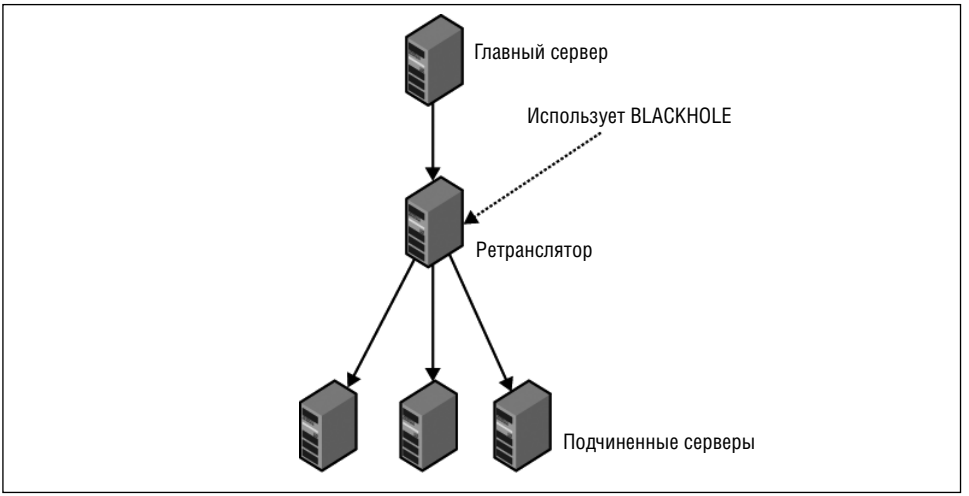


Рис. 5-4. Иерархическая топология, включающая главный сервер, ретранслятор и подчиненные серверы

С другой стороны, серверу-ретранслятору необходимо вести двоичный журнал со всеми изменениями, поскольку ретранслятор передает их другим подчиненным серверам. В отличие от обычных подчиненных серверов, ретранслятору не требуется вносить эти изменения в свою собственную базу, так как ему не придется отвечать на запросы.

Вкратце, обычный подчиненный сервер должен вносить изменения в свою базу данных, но не обязан вести двоичный журнал. Сервер-ретранслятор должен вести двоичный журнал, но изменений в базу данных не вносит.

Чтобы избежать внесения изменений в базу данных, необходимо иметь таблицы (чтобы формально выполнять операторы), но изменения просто пропускать. Для подобных целей разработан механизм БД под названием Blackhole. Этот механизм принимает любые операторы и докладывает об успешном их выполнении, но на самом деле отбрасывает все изменения. Сервер-ретранслятор дополнительно увеличивает задержку, вследствие которой его подчиненные серверы отстают от главного сервера несколько больше, чем подчиненные серверы, подключенные непосредственно к главному серверу. Это отставание следует скомпенсировать частичной разгрузкой главного сервера.

Настройка сервера-ретранслятора

Настроить сервер-ретранслятор не слишком сложно, но следует продумать организацию таблиц, создающихся на ретрансляторе, а также модификацию

таблиц, уже присутствующих на этом сервере к моменту смены его роли. То, что в базе не хранятся данные, ускоряет обработку событий и уменьшает запаздывание подчиненных серверов на последней стадии процесса репликации, поскольку данных для обновления нет. Чтобы настроить сервер-ретранслятор, необходимо:

1. Настроить подчиненный сервер так, чтобы он передавал все события, выполняемые потоком подчиненного сервера, записывая их в свой двоичный журнал.
2. Установить для всех таблиц сервера-ретранслятора механизм БД BLACKHOLE, что позволит как сберечь дисковое пространство, так и улучшить производительность.
3. Убедиться, что все новые таблицы, добавляемые на сервер-ретранслятор, также используют механизм БД BLACKHOLE.

Чтобы настроить сервер-ретранслятор на передачу событий, выполняемых потоком подчиненного сервера, достаточно добавить параметр log-slave-updates в файл *my.cnf*, как это было показано ранее.

Чтобы все таблицы, создаваемые на подчиненном сервере-ретрансляторе, также создавались с механизмом хранения BLACKHOLE, подключитесь к серверу и установите механизм БД по умолчанию:

```
relay> SET GLOBAL STORAGE_ENGINE = 'BLACKHOLE';
```

Наконец необходимо для всех таблиц, уже присутствующих на подчиненном сервере-ретрансляторе, заменить существующий механизм БД на механизм BLACKHOLE. Воспользуйтесь для этой цели оператором ALTER TABLE, применив его к каждой таблице сервера. Поскольку не следует заносить операторы ALTER TABLE в двоичный журнал (менее всего нам хотелось бы, чтобы подчиненные сервера сбрасывали получаемые ими изменения!), на время выполнения операторов ALTER TABLE отключите запись в двоичный журнал. Соответствующий сценарий приведен в лист. 5-4.

Лист. 5-4. Изменение механизма хранения для всех таблиц базы данных windy

```
relay> SHOW TABLES FROM windy;
+-----------------------------------+
| Таблицы_в_базе_данных_windy       |
+-----------------------------------+
| пользовательские_данные (user data) |
                  .
                  .
                  .
| профиль (profile)                 |
+-----------------------------------+
45 row in set (0.15 sec)
relay> SET SQL_LOG_BIN = 0;
relay> ALTER TABLE user_data ENGINE = 'BLACKHOLE';
```

```
                .
                .
                .
relay> ALTER TABLE profile ENGINE = 'BLACKHOLE';
relay> SET SQL_BIN_LOG = 1;
```

Собственно, вы уже знаете достаточно, чтобы преобразовать ваш сервер в сервер-ретранслятор. Большинство проходит такой путь — начинают с конфигурации, в которой все подчиненные серверы подключены непосредственно к главному, а спустя некоторое время понимают, что необходим подчиненный сервер-ретранслятор. Чаще всего стимулом оказывается перегрузка главного сервера, но иногда реконструкции требует сама архитектура системы. Итак, вы можете воспользоваться информацией из предыдущих разделов и добавить к существующему развертыванию сервер-ретранслятор:

1. Подключив подчиненный сервер-ретранслятор к главному серверу и настроив его в качестве сервера-ретранслятора.
2. Переключив на сервер-ретранслятор все подчиненные серверы по очереди.

Добавление ретранслятора программой на языке Python

Обратимся к задаче разработки средств администрирования ретрансляторов и пополним нашу библиотеку. Поскольку у нас уже есть система для создания новых ролей и присваивания этих ролей серверам, воспользуемся ей для определения специальной роли сервера-ретранслятора. Соответствующий код приведен в лист. 5-5.

Лист. 5-5. Определение роли ретранслятора

```
class Relay(role.Base):
   def __init__(self, master):
      pass

   def imbue(self, server):
      config = server.get_config()
      self._set_server_id(server, config)
      self._enable_binlog(server, config)
      config.set('mysqld', 'log-slave-updates' '1')
      server.put_config(config)
      server.sql("SET SQL_LOG_BIN = 0")
      for db in list of databases (списке баз данных):
         for table in server.sql("SHOW TABLES FROM %s", (db)):
            server.sql("ALTER TABLE %s.%s ENGINE=BLACKHOLE", (db,table))
      server.sql("SET SQL_LOG_BIN = 1")
```

Специализированные подчиненные серверы

В простом масштабируемом развертывании, — подобно тем, что обсуждались до сих пор — все подчиненные серверы получают все данные и, таким образом, могут обрабатывать запросы любого рода. Однако довольно редко частота запросов к данным распределяется равномерно. Обычно к некоторым данным обращаются очень часто, а к другим — достаточно редко. В качестве примера возьмем сайт веб-магазина:

- Каталог товаров просматривают почти постоянно.
- Данными о товарах на складе интересуются не так уж часто.
- Данные о пользователях запрашиваются не часто, поскольку бóльшая часть конфиденциальной информации обрабатывается как данные сеанса и хранится в cookie-файлах обозревателя.
- С другой стороны, если у пользователя cookies отключены, сервер будет запрашивать данные сеанса практически с каждым запросом страницы.
- Информация о новых товарах обычно востребуется чаще, чем о товарах, давно присутствующих на сайте. К примеру, «специальные предложения» пользуются особой популярностью.

Очевидно, что это расточительство — хранить данные, обращения к которым происходят редко, на всех подчиненных серверах, — просто на всякий случай. Гораздо правильнее организовать развертывание так, как показано на рис. 5-5, где одна группа серверов обслуживает редко запрашиваемые, а другая — востребованные данные.

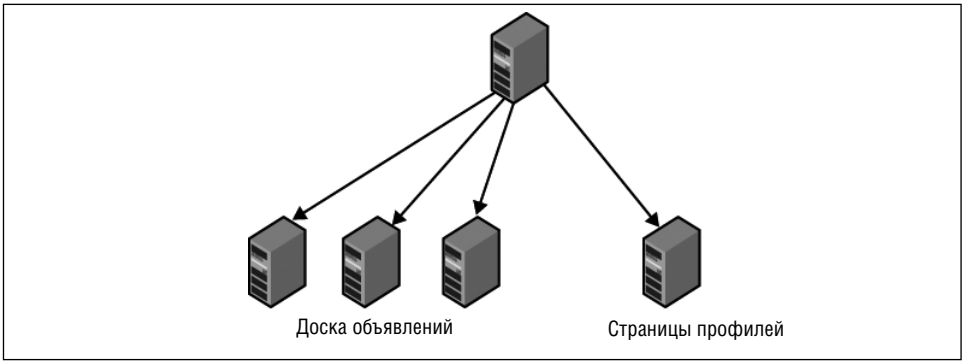


Рис. 5-5. Топология репликации, включающая главный и специализированные подчиненные серверы

Чтобы достигнуть этой цели, необходимо разделять таблицы при репликации. MySQL позволяет это сделать — либо посредством фильтрации событий, отправляемых главным сервером, либо (в качестве альтернативы) при помощи фильтрации событий, поступающих на подчиненный сервер.

Фильтрация событий репликации

Для выполнения фильтрации событий могут использоваться либо фильтры главного сервера (*master filters*), либо фильтры подчиненного сервера (*slave filters)*. Фильтры главного сервера проверяют информацию, которая пишется в двоичный журнал и затем отправляется на подчиненные серверы, в то время как фильтры подчиненного сервера контролируют события, выполняющиеся на этом подчиненном сервере. При фильтрации на главном сервере события для таблиц, не отвечающих условиям отбора, вообще не попадают в двоичный журнал, в то время как при фильтрации на подчиненных серверах события пишутся в двоичный журнал, посылаются на подчиненный сервер и отсеиваются только в тот момент, когда их следовало бы исполнить.

Если используются фильтры главного сервера, события вообще не хранятся в двоичном журнале. Это означает, что для восстановления баз данных невозможно использовать должным образом процедуру восстановления состояния на определенный момент времени [point-in-time recovery (PITR)], — если базы данных хранятся в архивном образе, их можно восстановить, используя резервную копию, но все изменения, внесенные в таблицы с момента копирования, восстановлены не будут, поскольку они не отражены в двоичном журнале.

Если используются фильтры подчиненного сервера, все изменения передаются по сети. Очевидно, что таким образом неэффективно расходуется пропускная способность сети, особенно в протяженных, глобальных сетях.

Далее в этой главе мы обсудим в деталях относительные преимущества фильтров главного и подчиненных серверов, а также предложим подход, позволяющий сохранить двоичный журнал без ущерба для пропускной способности сети.

Фильтры главного сервера

Для настройки фильтрации на главном сервере имеются два параметра:

- *binlog-do-db=db* Если база данных db является текущей для оператора, оператор заносится в двоичный журнал, в противном случае оператор пропускается.
- *binlog-ignore-db=db* Если база данных db является текущей для оператора, оператор пропускается, в противном случае оператор заносится в двоичный журнал.

Если требуется реплицировать все базы данных, кроме немногих, используйте параметр binlog-ignore-db. Если необходимо реплицировать только несколько баз данных, применяйте binlog-do-db. Не рекомендуется сочетать эти параметры, иначе чересчур усложнится логика отбора баз данных для репликации (см. рис. 3-3). Список баз данных в качестве значения этих параметров задавать нельзя, — придется повторить параметр несколько раз, чтобы приложить его более чем к одной базе.

К примеру, после добавления в файл параметров приведенных ниже строк, репликация будет выполняться для всех баз данных, кроме *top* and *secret*:

```
[mysqld]
...
binlog-ignore-db = top
binlog-ignore-db = secret
```

Применение для фильтрации событий параметров binlog-*-db означает, что несколько баз данных (в примере — две) не будут отражаться в двоичном журнале, и следовательно, в случае сбоя их восстановление в соответствии с процедурой восстановления состояния на определенный момент времени (PITR) окажется невозможно. По этой причине настоятельно рекомендуется для фильтрации потока репликации использовать не фильтры главного сервера, а фильтры подчиненных серверов. Фильтры главного сервера можно накладывать только на нестабильные, часто меняющиеся данные, которые не страшно утратить.

Фильтры подчиненных серверов

Для настройки фильтрации на подчиненных серверах предлагается длинный список параметров. Фильтры подчиненных серверов накладываются не только на события уровня базы данных, но и на события уровня отдельных таблиц, и даже на события, относящиеся к группам таблиц (группы определяются при помощи подстановочных символов).

В перечисленных ниже правилах параметры с подстановочными знаками (параметры, начинающиеся с replicate-wild) работают с полными именами таблиц, то есть с именами, состоящими из имени базы данных и имени таблицы. Эти параметры используют такие же шаблоны, как и функция для сравнения строк LIKE, — а именно, подчеркивание (_) заменяет одиночный символ, в то время как символ процента (%) заменяет строку любой длины. Обратите внимание, — чтобы шаблон считался корректным, он должен содержать точку. Дело в том, что имя базы данных и имя таблицы проверяются на соответствие шаблону по отдельности, то есть каждый подстановочный знак относится либо к имени базы данных, либо к имени таблицы.

- *replicate-do-db=db* Если база данных является текущей для оператора, оператор выполняется.
- *replicate-ignore-db=db* Если база данных db является текущей для оператора, оператор пропускается.
- *replicate-do-table=table* и *replicate-wild-do-table=db_pattern.tbl_pattern* Если имя обновляемой таблицы равно table или соответствует шаблону, обновления применяются к таблице.
- *replicate-ignore-table=table* и *replicate-wild-ignore-table=db_pattern.tbl_pattern* Если имя обновляемой таблицы равно table или соответствует шаблону, обновления таблицы пропускаются.

Описанные правила фильтрации накладываются непосредственно перед тем, как сервер принимает решение о выполнении, поэтому все события без исключения посылаются подчиненному серверу, и лишь затем фильтруются.

Применение фильтрации для секционирования событий подчиненных серверов

Итак, какие преимущества и недостатки фильтрация на главном сервере имеет по сравнению с фильтрацией на подчиненном? На первый взгляд было бы удобно организовать базы данных таким образом, чтобы фильтровать события сразу на главном сервере, при помощи параметров типа binlog-*-db, а не параметров replicate-*-db. В этом случае сеть освобождается от передачи множества бесполезных событий, которые все равно будут отброшены подчиненным сервером. Однако, как уже говорилось ранее в главе, фильтрация на главном сервере несет свои проблемы.

- Поскольку события отфильтровываются из двоичного журнала, а двоичный журнал только один, невозможно разделить, «разбить» изменения и отправить отдельные части базы данных разным серверам.
- Двоичный журнал используется еще и для процедуры восстановления состояния на определенный момент времени (PITR), поэтому при возникновении на сервере проблем невозможно выполнить полное восстановление в соответствии с этой процедурой.
- Если, по определенным соображениям, потребуется разбить данные по-другому, эта задача окажется невыполнимой, поскольку двоичный журнал уже отфильтрован, а откатить этот процесс невозможно.

Представляется идеальным вариантом фильтрация, применяемая к событиям, отправляемым с главного сервера, но не к событиям, регистрируемым в двоичном журнале. Было бы также неплохо, если бы подчиненный сервер мог контролировать фильтрацию, решая, какие данные реплицировать. Но в MySQL версии 5.1 это невозможно, и напротив, для фильтрации событий на подчиненном сервере в этой версии необходимо задействовать параметры replicate-*.

В команде разработчиков, отвечающей за репликацию MySQL, ведутся нескончаемые дискуссии о реализации расширенных возможностей фильтрации, которые позволяли бы фильтровать события в различные моменты обработки событий, а также допускали бы более сложную логику фильтрации.

На момент работы над этой книгой еще не было решено, на какой версии сервера будет реализована подобная расширенная фильтрация.

К примеру, чтобы настроить подчиненный сервер на работу с пользовательскими данными, хранящимися в таблицах с именами *users* и *profiles* в базе данных *app*, следует остановить сервер и добавить в файл *my.cnf* следующие параметры фильтрации:

```
[mysqld]
...
replicate-wild-do-table=app.users
replicate-wild-do-table=app.profiles
```

Если вас беспокоит проблема сетевого трафика, — актуальная в первую очередь для репликации по протяженным, глобальным сетям, — имеет смысл настроить подчиненный сервер-ретранслятор на том же компьютере, на котором расположен главный сервер (рис. 5-6). Или хотя бы на компьютере, расположенном в том же сегменте сети, что и главный сервер, исключительно для генерации фильтрованной версии двоичного журнала главного сервера.

Шардинг

Мы уже обсудили масштабирование операций чтения путем добавления подчиненных серверов к главному и перенаправления чтения подчиненным серверам, в то время как операции записи выполняются главным сервером. По мере роста нагрузки не составит проблемы подключить к главному серверу еще больше подчиненных, чтобы обслужить большее количество запросов на чтение. До этого момента мы рассматривали только ситуации, когда операции записи производятся главным сервером, поэтому если количество таких операций достаточно возрастет, главный сервер окажется узким местом системы в процессе ее мере вертикального масштабирования. Можно ли масштабировать запись так же успешно, как и чтение?

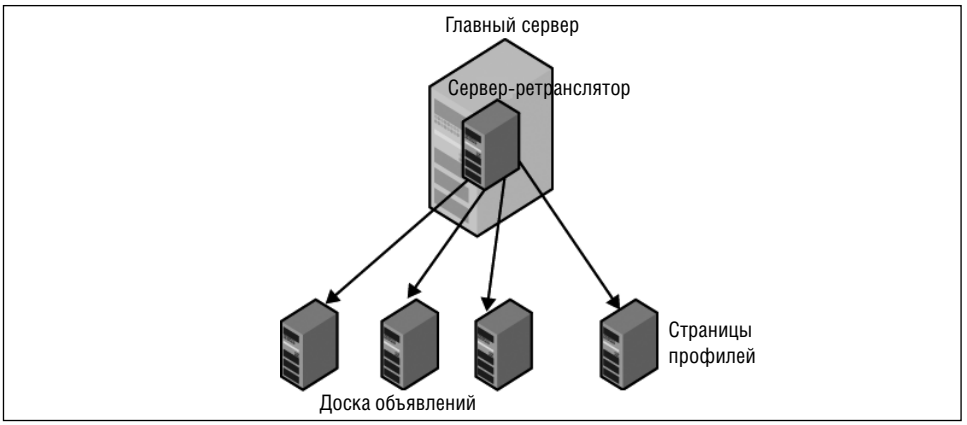


Рис. 5-6. Фильтрация, ведущаяся сервером-ретранслятором, расположенным на компьютере главного сервера

Перед тем, как приступить к поиску наилучшего способа масштабирования операций записи, рассмотрим конфигурацию (см. рис. 5–7), состоящую из двух главных серверов, поддерживающих двунаправленную репликацию и набор клиентов, которые обновляют информацию то на одном, то на другом сервере, в зависимости от текущей потребности. Хотя кажется, что по-

добная конфигурация удваивает возможности выполнения операций записи (главных серверов у нас два), она не помогает организовать масштабирование. Каждый оператор выполняется на обоих главных серверах, поэтому и количество операций записи удваивается. Таким образом, мы остаемся на прежних позициях. Короче говоря, установка пары главных серверов не ведет к масштабированию операций записи, поэтому необходимо прибегнуть к другим мерам.

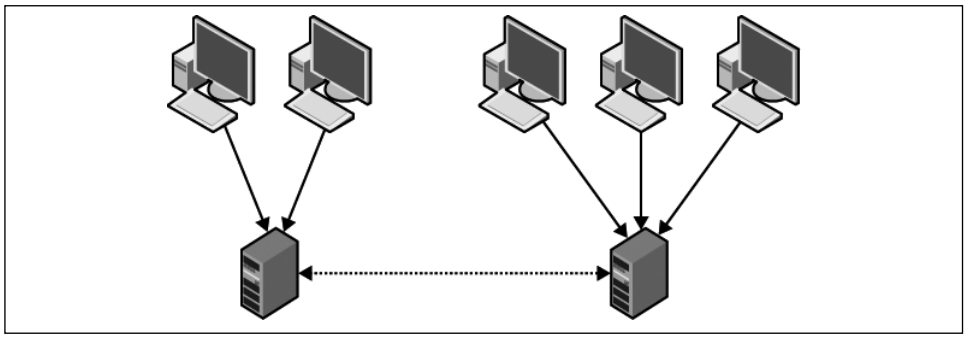


Рис. 5-7. Пара главных серверов, поддерживающих двунаправленную репликацию

Пара главных серверов, показанных на рис. 5–7 не обеспечивают масштабирование, поскольку удвоение производительности сводится на нет тем фактом, что все операторы выполняются дважды. Следовательно, масштабирование могло бы стать возможным, если бы не приходилось выполнять все операторы по два раза, — а для этого между серверами на рис. 5-7 не должно быть репликации, разные сервера должны быть полностью независимы.

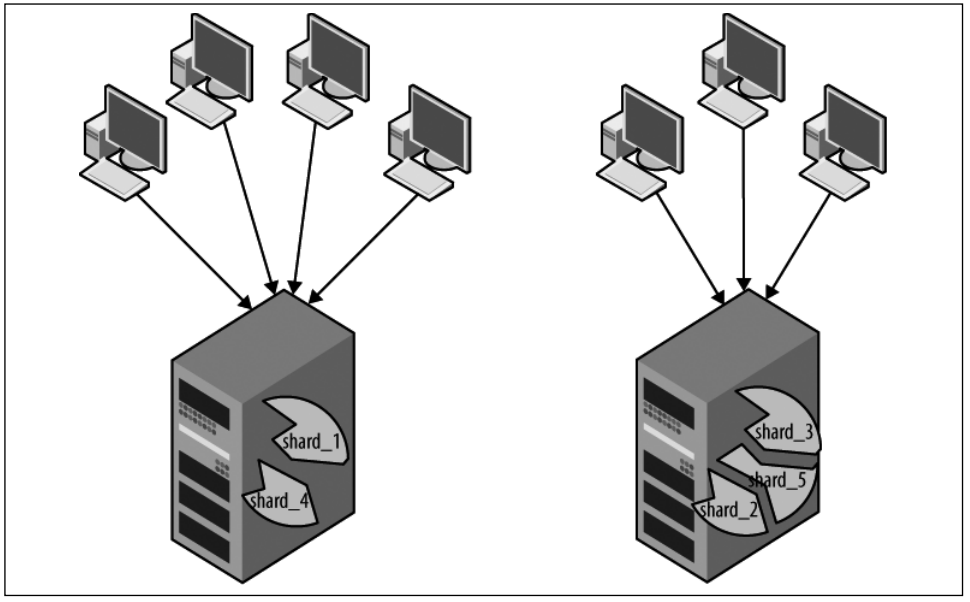


Рис. 5-8. Несколько узлов, поддерживающих шардинг

Такая архитектура позволит масштабировать операции записи, секционируя данные, разбивая их на два абсолютно независимых множества. Тогда клиент может обратиться к секции, отвечающей за обновляемые им данные. При этом не нужны ресурсы для обновления данных в других секциях.

Разделение данных в подобном стиле обычно называется шардингом (*sharding*), также часто используются термины расщепление (*splintering*) или горизонтальное секционирование [(*horizontal*) *partitioning*], и о разделе говорят — шард (*shard*). Принято использовать шардинг для объемных данных, таких как статьи в блогах, комментарии, изображения и видео, при этом каталоги и пользовательские данные размещаются в центральном хранилище, как проиллюстрировано на рис. 5-8.

Производительность системы улучшается не только потому, что связанные между собой данные размещаются в одном узле (что позволяет направлять запросы к единственному узлу), но и потому, что можно приблизить шарды к пользователю географически, и уменьшить таким образом задержку сети. Имеет смысл прибегнуть к шардингу, если требуется:

- *Приблизить данные к пользователю географически* Как уже говорилось, приближение объемных данных к пользователю позволяет уменьшить задержку сети.
- *Уменьшить размер рабочего набора* Поиск в небольшой таблице более эффективен, чем в объемной, к тому же появляется возможность вести поиск параллельно — в подобной конфигурации это достаточно просто.

  Этот подход эффективен, только для шардов примерно одинакового размера. Если вы обратитесь к этой стратегии, вам придется сбалансировать размер шардов.
- *Сбалансировать нагрузку* При использовании шардинга не только сокращается размер рабочего набора за счет уменьшения размеров таблиц, но и балансировка нагрузки обновления становится эффективнее. Если объем обновлений, поступающих на отдельные шарды, слишком велик и не пропорционален размерам шарда, такой шард можно разбить в свою очередь на меньшие шарды.

Применяя шардинг, вы можете разместить на сервере несколько шардов. Это имеет смысл, поскольку когда придется повторно балансировать систему, намного проще переместить шард на другой сервер, чем распределять данные по-новому. Небольшие секции не только удобны для администрирования, они еще позволяют сокращать размеры таблиц, что улучшает общую производительность системы.

При работе с описываемой архитектурой, положение шарда не фиксируется, поэтому необходимо иметь возможность по идентификатору шарда получить узел, на котором он хранится. Обычно для этой цели используется конфигурация, представленная на рис. 5-9, где информация о местоположении шардов хранится в центральном репозитории.

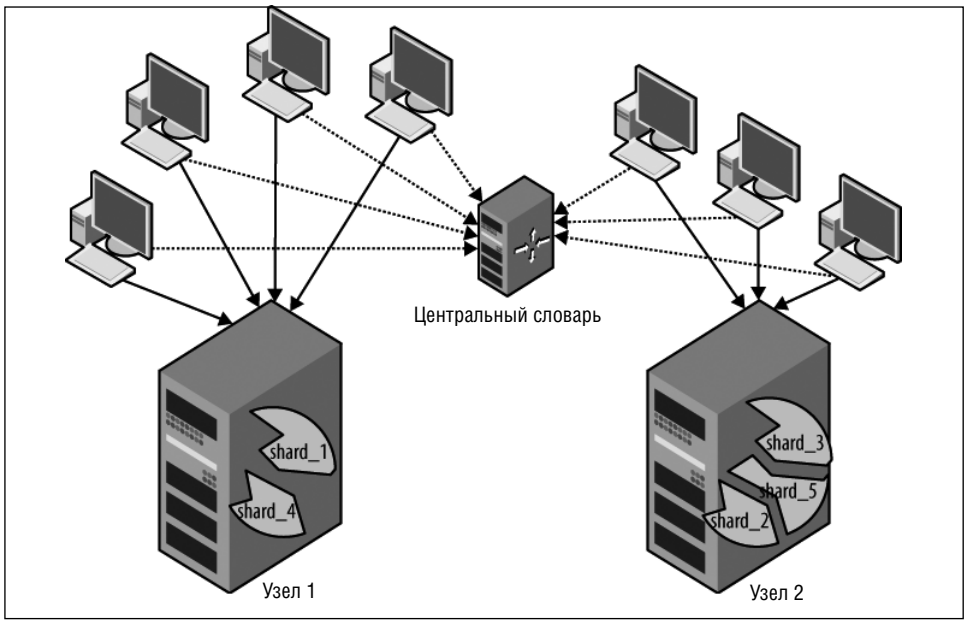


Рис. 5-9. Шарды с централизованным словарем

Хотя при переводе существующего приложения на схему шардинга проще размещать по одному шарду на узел, но более распространенный подход предполагает сосуществование на узле нескольких шардов. Так можно сократить рабочий набор, что, вероятно, приведет к ускорению выполнения запросов.

Обычно поддержку шардинга обеспечивает либо непосредственно приложение, либо промежуточный слой между базой данных и приложением. Даже если шардингом управляет слой приложения баз данных, очень сложно полностью скрыть шардинговую структуру, не жертвуя при этом производительностью. Лучше разработать прикладное приложение, знающее о структуре шардинга и использующее это знание в целях повышения эффективности.

Имеются несколько реализаций шардинга для MySQL. Две самых популярных — Hibernate Shards от команды Google, и HiveDB (*http://www.hivedb.org*), реализованы на уровне слоя приложения баз данных на языке Java. Можно обратиться и к MySQL Proxy, хотя шардинг там реализован только в начальной степени. Тем не менее, команда разработчиков MySQL считает разработку эффективных решений для масштабирования операций записи одним из своих главных приоритетов, поэтому не стоит отбрасывать этот вариант решения.

Представление шарда

Для эффективной работы со структурой шардинга важен правильный подход к представлению шардов. Необходимо учитывать следующие требования:

- Перемещение шардов не должно быть слишком сложным, поэтому процедуры создания резервной копии шарда и восстановления ее в другом месте также должны быть простыми.
- На сервере можно разместить несколько шардов, поэтому необходимо различать шарды, сосуществующие на одном узле.
- Должна существовать возможность репликации отдельного шарда сервера, для перемещения этого шарда.

Существует только один вариант, позволяющий обеспечить выполнение первого условия, — каждый шард должен быть представлен как отдельная база данных MySQL. Большинство способов создания резервных копий позволяют создавать копии отдельных баз данных, но при попытке скопировать отдельные таблицы могут возникнуть проблемы. Так произойдет не только, например, при непосредственном копировании каталога базы данных, но и при обращении ко многим другим методам резервного копирования. Если для шарда отводится отдельная база данных, то удовлетворяется и последнее требование из нашего списка, поскольку посредством параметра replicate-do-db можно отобрать для репликации отдельные шарды на сервере (это возможно, поскольку вы будете перемещать шарды на главные серверы, а главные серверы в общем случае не реплицируются откуда угодно).

После того как для шардов будут отведены отдельные базы данных, можно выполнить и второе условие, добавив уникальный номер к именам всех баз данных. То есть шарды на сервере — это базы данных с такими именами, как *shard_123*, а при этом каждая таблица имеет секцию в каждой из баз данных. Например, таблица *posts* состоит из секций *shard_1.posts*, *shard_2.posts*, ..., *shard_N.posts*.

Если вы выбираете вышеописанный подход, то назначаете своим таблицам либо «простые», «обычные» имена, либо имена, ссылающиеся на их шарды. В первой схеме к таблице *posts*, разбитой по базам данных, можно обращаться как *shard_123.posts*, *shard_124.posts* и т.п.. Во второй схеме имена будут выглядеть как *shard_123.posts_123*, *shard_124.posts_124* и т. д. Хоятя добавление номера шарда к таблице выглядит избыточной мерой, это поможет избежать проблем из-за попыток приложения послать запросы не тому шарду.

Секционирование данных

Если вы будете каждую порцию данных записывать на отдельный сервер, то добьетесь эффективного масштабирования операций записи. Но этого недостаточно, чтобы обеспечить масштабируемость — дело опять-таки в эффективности, и чтобы эффективно отбирать данные, необходимо хранить связанные между собой данные вместе. Поэтому ключевой вопрос эффективного шардинга — это задание такого *ключа секционирования* (*partition key)*, который позволит данным, обычно запрашиваемым совместно, располагаться в одном и том же шарде, или по крайней мере в минимально возможном числе шардов.

Чтобы создать хороший ключ секционирования, нужна изобретательность вкупе с определенной суммой знаний о структуре данных в базе. Например, приложение, предоставляющее профессиональным фотографам сервис по надежному хранению их творений, может распределить фотографии по районам проживания заказчиков и размещать все снимки каждого из фотографов в шарде, ближайшем к нему географически.

Выбрав ключ секционирования, вы должны будете решить, как использовать его для разбиения данных. На этом этапе следует определить *функцию секционирования* (*partition function*), принимающую в качестве параметра ключ секционирования и возвращающую номер шарда. Ниже описаны две общеупотребительные разновидности схемы:

- *Схемы статического шардинга* В этом случае ключ секционирования привязан к статическому элементу, обычно диапазону или хэш-функции. Например, если ваш ключ секционирования «Страна», вы размещаете всех шведов в одном шарде, а всех американцев — в другом (пренебрежем тем фактом, что население Соединенных штатов примерно в тридцать раз больше, чем население Швеции).

  Другой способ распределить шарды — это связать их с уникальным идентификатором (ID) пользователя. Можно выделить шарды диапазонам идентификаторов, а именно — первый шард отвечает за пользователей с 0 по 9999, второй — с 10000 по 19999 и т. п. Или же вы «разбрасываете» пользователей наполовину случайным образом, на основании хэша, вычисляемого по значению последних четырех цифр идентификатора.
- *Схемы динамического шардинга* В этом случае функция секционирования ищет ключ секционирования в словаре, указывающем на шарды с нужными данными. Эта схема — более гибкая по сравнению со статической, но требует центрального хранилища для словаря, что увеличивает время выполнения запроса. Центральное хранилище представляет также проблему и с точки зрения высокой доступности, поскольку становится узким местом при сбое.

Как вы уже поняли, основной проблемой схем статического шардинга является неравномерное распределение запросов, — как в нашем примере, где данные разбиваются по странам. В этом случае не исключено, что нагрузка на шард с данными по США будет в 30 раз больше, чем нагрузка на шард с данными по Швеции. Шведам это понравится (в предположении, что серверы имеют одинаковую мощность) — время отклика для их запросов будет совсем небольшим. Американских же посетителей ждут определенные неудобства. Поэтому правильный выбор ключа секционирования имеет исключительное значение.

Схемы динамического шардинга очень гибки. Они позволяют не только модифицировать шардинг, но и перемещать данные между шардами в случае необходимости. Как всегда и бывает, за эту гибкость приходится платить, — чтобы найти шард с нужными данными, необходимо выполнить

дополнительные запросы. Использование систем кэширования (например, Memcached) несколько облегчает ситуацию, но в конце концов, хорошей производительности можно добиться только тщательно продумывая проект и приводя его в соответствие с структурой пользовательских запросов.

После создания ключа и функции секционирования можно спроецировать номера шардов на узлы (где каждый сервер содержит один или более полных узлов). При работе со схемой статического шардинга функция секционирования может автоматически сопоставить номер шарда узлу — как часть алгоритма функции. Такая простота сулит эффективность, но в обмен на гибкость, поскольку при перемещении шарда придется переписать функцию секционирования. При работе со схемой динамического шардинга информация о сопоставлении номеров шардов и узлов может храниться в таблице.

Балансировка шардов

Чтобы при изменении загруженности системы отклик не пострадал, или же для потребностей администрирования, время от времени приходится перемещать данные — либо целые шарды на другие узлы, либо данные между шардами. Обе эти процедуры привносят собственные проблемы при повторной балансировке нагрузки, — ведь время отключения серверов должно быть минимизировано, а еще лучше было бы серверы не отключать вообще. Предпочтение отдается автоматизированным решениям.

Перенос шарда на другой узел

Проще всего переместить на другой узел шард целиком. Если вы последовали нашему совету и представили каждый шард как отдельную базу данных, перенос базы данных будет так же несложен, как и перенос каталога. Однако выполнить это действие, не останавливая на узле операции записи, уже далеко не так просто.

Перемещение шарда с одного узла (исходного) на другой узел (целевой) вообще без отключения невозможно, но можно свести отключение к минимуму. Этот метод аналогичен описанному нами в главе 2 методу создания подчиненного сервера. Ключевая идея состоит в том, что создается резервная копия шарда, которая восстанавливается на целевом узле, а затем все изменения, которые произошли в интервале времени от резервного копирования до восстановления, выполняются повторно посредством репликации.

1. Создайте резервную копию предназначенной для переноса базы данных на исходном узле. Для удобства предположим, что база данных называется *shard_123*. Резервное копирование можно выполнять как в оперативном, так и в автономном режимах.
2. Каждая резервная копия, как мы видели в предыдущих главах, содержит копируемые данные на момент до определенной точки в двоичном журнале. Запомните эту позицию двоичного журнала.

3. Отключите целевой узел, остановив сервер.
4. Пока сервер остановлен:
   а. Добавьте в файл параметров параметр replicate-do-db, чтобы реплицировать только нужный шард.

   ```
   [mysqld] replicate-do-db=shard_123
   ```

   б. Если надо, то именно сейчас, пока сервер остановлен, восстановите резервную копию исходного узла.
5. Снова включите сервер.
6. Настройте репликацию так, чтобы чтение данных началось с позиции, отмеченной вами на шаге 2, и запустите репликацию на целевом сервере. Тогда события будут считаны с исходного сервера и все изменения применятся к перемещаемому шарду.

   Запланируйте для целевого узла дополнительную мощность, — это позволит справиться с временным увеличением количества операций записи на нем.
7. Если целевой узел достаточно близок к исходному, заблокируйте базу данных шарда на исходном узле, чтобы в нее нельзя было вносить изменения. Не обязательно останавливать изменения шарда на целевом узле, поскольку запросы на операции записи туда еще не поступают.

   Проще всего наложить блокировку командой LOCK TABLES и заблокировать таким образом все таблицы шарда, но возможны и другие варианты, например, простое удаление таблиц (если приложение способно функционировать при исчезнувшей таблице, к примеру, как показано ниже, это неплохой вариант).
8. Проверьте позицию журнала на исходном сервере. Поскольку шард больше не обновлялся, это будет последняя позиция журнала, которую следует восстанавливать.
9. Подгоните целевой сервер к этой позиции, например, посредством команд START SLAVE UNTIL и MASTER_POS_WAIT.
10. Отключите репликацию на целевом сервере, выполнив команду RESET SLAVE. Вся информация о репликации будет удалена, в том числе файлы *master.info*, *relay-log.info* и все файлы журналов ретрансляции. Если вы добавлzли какие-либо параметры репликации в файл *my.cnf*, их следует удалить, лучше всего на следующем шаге.
11. По желанию остановите целевой сервер, удалите из файла my.cnf параметр replicate-do-db, относящийся к этому серверу и запустите сервер снова.

    Этот шаг не является строго обязательным, поскольку параметр replicate-do-db применяется только для перемещения шардов и не влияет на функционирование шарда, когда тот уже будет на новом месте. Когда шард потребуется переместить снова, тогда в любом случае придется этот параметр изменить.

12. Обновите информацию о шарде так, чтобы запросы на обновление перенаправлялись к новому местоположению шарда.
13. Разблокируйте базу данных, чтобы операции записи шарда снова начали выполняться.
14. Удалите базу данных шарда с исходного сервера. В зависимости от способа, которым вы заблокировали шард, возможно, к шарду еще поступают запросы на чтение данных, и вам следует это учесть.

Вот так! Мы уложились в несколько шагов. К счастью, эти шаги можно автоматизировать, воспользовавшись библиотекой MySQL Replicant. Детали реализации для каждого отдельного шага во многом зависят от того, как реализовано приложение, — пример мы рассмотрим позже, при обсуждении образца приложения.

Перемещение данных между шардами

Даже если вы уже выработали стратегию перемещения шардов, вам все равно время от времени придется также переносить данные *между* шардами. Вернемся к тому примеру, в котором речь шла о приложении для хранения фотографий, — пользователь может в конце концов переехать, или же статистика покажет, что размещение снимка в другом шарде повысит эффективность развертывания. Например, сам фотограф живет в Калифорнии, но многочисленная родня этого художника, находящаяся в Нью-Йорке, проводит уйму времени за просмотром его работ. В подобных случаях необходимо автоматически изменить шардинг системы и перенести пользователя на другой шард.

Способы для переноса данных между шардами в большой степени зависят от приложений, поэтому мы ограничимся примером. Как правило, перемещение данные из шарда в шард — дорогостоящая операция, поскольку необходимо на достаточно продолжительное время перевести шард в автономный режим. Только так можно предотвратить изменение перемещаемых данных.

Пример шардинга

Чтобы продемонстрировать один из способов настройки шардинга, создадим небольшой пример, — первоначально со схемой статического шардинга, которую мы впоследствии переработаем в схему шардинга динамического. Речь пойдет о простом сайте блогов, на котором пользователи регистрируются, размещают статьи и оставляют комментарии.

В этом примере мы разработаем приложение, при помощи которого одни пользователи размещают на сайте статьи, а другие пишут к ним комментарии. Структура нашей базы данных показана на рис. 5-10, и как нетрудно заметить, она почти примитивна.

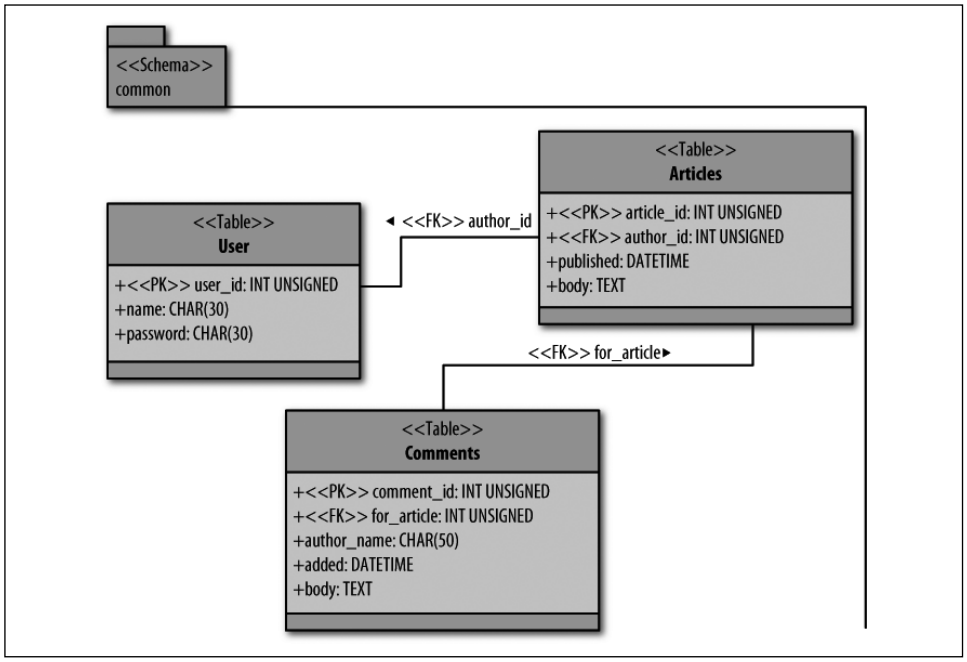


Рис. 5-10. Схема базы данных, представленная на унифицированном языке моделирования [UML (Unified Modeling Language)]

Лист. 5-6 содержит определение общих данных, которые будут для удобства размещены в таблице, так и названной *common*. Пока к общим данным относится только таблица *user* с перечнем всех пользователей, но далее в этой главе мы добавим туда и другие таблицы.

Лист. 5-6. Таблицы для общих данных

```
CREATE TABLE user (
   user_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   name CHAR(30),
   password CHAR(30)
);
```

Таблицы со статьями и комментариями мы поместим в другую базу данных, которая будет состоять из таблицы *articles* для статей и таблицы *comments* для комментариев — см. лист. 5-7.

Лист. 5-7. Таблицы для статей и комментариев к ним

```
CREATE TABLE articles(
   article_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   author_id INT,
   published DATETIME,
   body TEXT
);

CREATE TABLE comments (
```

```
    comment_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    for_article INT,
    author_name CHAR(50),
    added DATETIME,
    body TEXT
);
```

Как вы видите, наше приложение совсем несложное, но оно подходит для нашей цели — продемонстрировать приложение, поддерживающее шардинг. Чтобы упростить описание, мы сознательно не стали задавать внешние ключи. Для подобных централизованных баз данных, доступ к которым требуется часто и к тому же одновременно со стороны множества клиентов, лучшим механизмом хранения является тот, который допускает блокировку на уровне строк, например InnoDB.

Шардинг базы данных

Допустим, что количество пользователей значительно выросло, соответственно выросло и количество размещенных на сайте статей. Данными о пользователях проще управлять, если они находятся на единственном узле, поэтому не надо разбивать таблицу *user*, — данные о 100 миллионах пользователей занимают только 6 Гб, проблем с дисковым пространством не возникает, хотя есть другие причины, по которым лучше не проектировать таблицы с сотней миллионов строк. С другой стороны, статьи и сопровождающие их комментарии, напротив, становятся чересчур объемными, поэтому имеет смысл выполнить шардинг соответствующих таблиц.

В этом случае статьи и комментарии разбиваются на отдельные шарды, при этом комментарии к каждой статье хранятся в том же шарде, что и сама статья. В этом случае пользователи будут получать доступ к статьям и комментария эффективно, посредством единственного запроса к одному шарду. Для организации такой структуры добавим номер к имени каждого шарда, и разместим определения таблиц из лист. 5-7 на каждом шарде. Для удобства постоянной частью имени шарда будет просто *shard*, тогда примером имени шарда с номером послужит имя *shard_123*.

Ключи секционирования и функции секционирования

Посетитель, который просматривает пользователей, сочтет удобным, если ему одновременно предложат и заголовки всех статей, размещенных этим пользователей. При поиске статьи, пользователи часто предпочитают сразу получать и комментарии к ней.

Чтобы удовлетворить таким особенностям запросов, мы выполним шардинг данных так, чтобы все статьи одного пользователя располагались в одном и том же шарде, и все комментарии для статьи размещались в том же самом шарде, что и сама статья. Это означает, что в нашем случае требуется несколько ключей секционирования:

- Когда извлекается пользователь, ключом секционирования служит идентификатор пользователя user_id.
- Когда извлекается статья, ключом секционирования служит идентификатор статьи article_id.

По тому же принципу мы разработаем две функции секционирования, — одна принимает идентификатор пользователя и возвращает номер шарда, другая принимает идентификатор статьи и возвращает номер шарда.

- *От идентификатора пользователя к номеру шарда* Чтобы найти номер шарда по ID пользователя, мы просто возьмем остаток от деления этого идентификатора на количество шардов. То есть для 100 шардов и идентификатора пользователя 192 мы получим номер шарда, равный 92.
- *От идентификатора статьи к номеру шарда* Определить номер шарда по идентификатору статьи не так просто, поскольку сперва следует найти автора статьи. Проще всего — но этот способ, к сожалению, слишком привязан к конкретному приложению, — сделать идентификатор автора доступным в коде приложения к тому моменту, когда потребовалась статья. Это несложно реализовать, если пользователь выбирает статью на странице другого пользователя, — тогда идентификатор автора просто добавляется как скрытое поле к HTML-форме, — но в других ситуациях эта задача может оказаться непростой.

  Альтернативный вариант — поместить необходимую информацию о сопоставлении идентификаторов статей идентификаторам пользователей в отдельной таблице общей базы данных *common*.

Мы не будем расписывать код, специфичный для приложения, но вкратце обсудим таблицы, добавляемые в базу данных *common*. Этот подход более гибок, чем встраивание схемы шардинга в код приложения, даже когда выбрана схема статического шардинга, но позже, когда мы перейдем к схеме динамического шардинга, крайне важно будет иметь возможность обновлять информацию о шардинге динамически.

Необходимые изменения выделены жирным шрифтом в коде, представленном в лист. 5-8. Заметьте, что столбец шардов проиндексирован, поскольку поиск по нему будет вестись очень часто.

Лист. 5-8. Таблицы с информацией о местонахождении шардов и узлов

```
CREATE TABLE user (
   user_id INT UNSIGNED AUTO_INCREMENT,
   name CHAR(50), password CHAR(50),
   PRIMARY KEY (user_id)
);

CREATE TABLE node_for (
   shard INT UNSIGNED,
   host CHAR(28),
   port INT UNSIGNED,
```

```
   sock CHAR(64),
   KEY (shard)
);

CREATE TABLE user_for (
   article_id INT UNSIGNED,
   user_id INT UNSIGNED,
   PRIMARY KEY (article_id)
);
```

Воспользовавшись таблицами из лист. 5-8 вы легко определите функции для получения номера шарда и адреса узла по идентификатору статьи или по идентификатору пользователя. Соответствующие функции приведены в лист. 5-9. Обе функции выполняют SQL-запрос к серверу, на котором хранятся общие данные. Если нужно получить идентификатор пользователя по идентификатору статьи, объедините запросы, используя подзапрос, чтобы сократить время передачи и приема, если сервер расположен на большом расстоянии.

Обратите внимание на отдельную функцию shardNumber. На данном этапе эта функция реализует схему статического шардинга, сопоставляя идентификатору пользователя номер шарда при помощи простой арифметической операции «взятие по модулю». Далее в главе мы внесем в эту функцию изменения, позволяющие перейти к схеме динамического шардинга.

Лист. 5-9. PHP-функции для получения номера шарда и адреса узла

```
   function shardNumber($userId)
   {
      return $userId % 4;
   }

   $NODE = array();
   $NODE[] = array("localhost", 3307, "/var/run/mysqld/mysqld1.sock");
   $NODE[] = array("localhost", 3308, "/var/run/mysqld/mysqld2.sock");
   $NODE[] = array("localhost", 3309, "/var/run/mysqld/mysqld3.sock");
   $NODE[] = array("localhost", 3310, "/var/run/mysqld/mysqld4.sock");

   function getShardAndNodeFromUserId($userId, $common)
   {
      global $NODE;
1     $shardNo = shardNumber($userId);
2     $row = $NODE[$shardNo % count($NODE)];
      $db_server = $row[0] == "localhost" ? ":{$row[2]}" : "{$row[0]}:{$row[1]}";
      $conn = mysql_connect($db_server, 'query_user');
3     mysql_select_db("shard_$shardNo", $conn);
      return array($shardNo, $conn);
   }
```

```
function getShardAndNodeFromArticleId($articleId, $common) {
   $query = "SELECT user_id FROM article_author WHERE article_id = %d";
   mysql_select_db("common");
   $result = mysql_query(sprintf($query, $articleId), $link);
   $row = mysql_fetch_row($result);
   return getShardAndNodeFromUserId($row[0], $common);
}
```

Обновление шарда или чтение из него

Разобравшись с определением номера шарда и адреса узла, переходите к разработке функций для получения информации от шардов. Две такие функции показаны в лист. 5-10:

- *getArticlesForUser* Эта функция принимает идентификатор пользователя и возвращает массив всех статей, им написанных. Функция секционирования определена нами так, что все эти статьи находятся в одном шарде, поэтому функция получения статей в строке, помеченной 1, вычисляет номер шарда, общего для всех статей. Затем в строке 2 определяется узел этого шарда. И наконец, формируется правильное имя базы данных, представляющей шард (строка 3), а узлу направляется единственный запрос, возвращающий все статьи.
- *getCommentsForArticle* Эта функция принимает идентификатор пользователя и идентификатор статьи, а возвращает массив, включающий статью и все комментарии к ней. В нашем конкретном примере идентификатор пользователя является частью полного идентификатора статьи, поэтому вызывающая функция получает его, не прибегая к дополнительному поиску.

Описанные функции достаточно просты, — после определения нужного шарда достаточно послать запрос его узлу. Поскольку на одном узле могут располагаться несколько шардов, необходимо убедиться, что чтение обращено к правильной базе данных. Чтобы упростить демонстрацию, мы не включили в функции обработку ошибок.

Лист. 5-10. PHP-функции для получения статей и комментариев

```
function getArticlesForUser($userId, $common)
{
   $query = <<<END_OF_SQL
SELECT author_id, article_id, title, published, body
   FROM articles
   WHERE author_id = $userId
END_OF_SQL;

   list($shard, $node) = getShardAndNodeFromUserId($userId, $common);
   $articles = array();
   $result = mysql_query($query, $node);
   while ($obj = mysql_fetch_object($result))
```

```
      $articles[] = $obj;
   return $articles;
}

function getArticleAndComments($userId, $articleId, $common)
{
   list($shard, $node) = getShardAndNodeFromArticleId($articleId, $common);
   $article_query = <<<END_OF_SQL
SELECT author_id, article_id, title, published, body
   FROM articles
  WHERE article_id = $articleId
END_OF_SQL;

   $QUERIES[] = $article_query;
   $result = mysql_query($article_query, $node);
   $article = mysql_fetch_object($result);

   # Комментарии извлекаются из того же шарда
   $comment_query = <<<END_OF_SQL
SELECT author_name, body, published
   FROM comments
  WHERE article_ref = $articleId
END_OF_SQL;

   $result = mysql_query($comment_query, $node);
   $comments = array();
   while ($obj = mysql_fetch_object($result))
      $comments[] = $obj;
   return array($article, $comments);
}
```

В этом примере мы читаем данные непосредственно из шардов, но если требуется масштабировать операции чтения, следует направлять запросы на чтение подчиненным серверам. Разработать такую схему также несложно.

Реализация динамической схемы шардинга

Недостаток подхода, обсуждаемого нами до настоящего момента, заключается в том, что функция секционирования является статической. Следовательно, если отдельные узлы будут перегружены, для перемещения шардов на другие узлы придется вносить изменения в код приложения, что усложнит задачу.

В качестве примера возьмем то же несложное приложение по обслуживанию блога, с которым мы работали до сих пор. Если пользователь неожиданно размещает очень интересные статьи и тем самым обращает на себя внимание большого числа посетителей, соответствующий шард становится «горячим». Это ведет к разбалансировке шардов, некоторые шарды оказываются горячими, поскольку их пользователи обрели известность, другие,

напротив, выглядят холодными, — пользователи, размещенные на них, не проявляют активности. Если значительное число активных пользователей приписано к одному и тому же шарду, количество запросов к этому шарду может вырасти настолько, что невозможно будет соблюсти приемлемое время отклика. Решение состоит в перемещении пользователей с горячих шардов на холодные, но предложенная ранее схема для этого не подходит.

Динамический шардинг позволяет вашей программе переносить шарды между узлами в соответствии с трафиком. Чтобы реализовать подобную схему, нам придется внести некоторые изменения в общую базу данных, в частности, добавить таблицу с информацией о местоположении шардов. Таблица *user* подходит для указания шардов, соответствующих пользователю.

В лист. 5-11 приведено модифицированное описание базы данных, — добавлена таблица *shard_to_node*, устанавливающая соответствие между номерами шардов и узлами. В таблице *user* tпоявился новый столбец, содержащий информацию о шарде, к которому приписан пользователь.

Лист. 5-11. Модифицированное для поддержки динамического шардинга описание общей базы данных

```
CREATE TABLE user (
   user_id INT UNSIGNED AUTO_INCREMENT,
   name CHAR(50), password CHAR(50),
   shard INT UNSIGNED,
   PRIMARY KEY (user_id)
);

CREATE TABLE shard_to_node (
   shard INT UNSIGNED,
   host CHAR(28),
   port INT UNSIGNED,
   sock CHAR(64),
   KEY (shard)
);

CREATE TABLE article_author (
   article_id INT UNSIGNED,
   user_id INT UNSIGNED,
   PRIMARY KEY (article_id)
);
```

Чтобы определять адрес узла, содержащего шард, придется добавить в PHP-функцию запрос, который будет извлекать информацию об узле из таблицы *shard_to_node*. Исправленный код представлен в лист. 5-12. Обратите внимание — мы отказались от массива узлов, заменив его на запрос к таблице *shard_to_node* в базе данных *common*. К тому же функция теперь получает номер шарда, хранящего данные пользователя, выполняя запрос к таблице *user*.

Лист. 5-12. Изменения, обеспечивающие переход к схеме динамического шардинга

```
function shardNumber($userId, $common)
{
   $result = mysql_query("SELECT shard FROM user WHERE user_id = $userId", $common);
   $row = mysql_fetch_row($result);
   return $row[0];
}

function getShardAndNodeFromUserId($userId, $common)
{
   $shardNo = shardNumber($userId);
   $query = "SELECT host,port,sock FROM shard_to_node WHERE shard = %d";
   mysql_select_db("common", $common);
   $result = mysql_query(sprintf($query, $shardNo), $common);
   $row = mysql_fetch_row($result);
   $db_server = $row[0] == "localhost" ? ":{$row[2]}" : "{$row[0]}:{$row[1]}";
   $conn = mysql_connect($db_server, 'query_user');
   mysql_select_db("shard_$shardNo", $conn);
   return array($shardNo, $conn);
}
```

Вы уже поняли, как выполняется поиск шарда в динамической схеме, а далее мы предложим код для перемещения шардов на новые узлы и для добавления новых шардов. Это и станет предметом обсуждения в следующем разделе.

Повторная балансировка шардов

Перейдя от статического шардинга к динамическому, мы получили инструмент для балансировки системы, а именно, — возможность без труда перемещать шарды между узлами и данные между шардами. Вы можете применять эту методику для решения задачи реструктуризации шардинга, то есть для выполнения полной повторной балансировки данных, размещенных в шардах.

К счастью, перенести шард целиком с узла на узел несложно. Вначале создается резервная копия шарда, которая восстанавливается на другом узле. Если все шарды являются отдельными базами данных, и используемый вами механизм хранения представляет базы данных как каталоги файловой системы, для перемещения шарда имеется несколько способов.

Определения объектов баз данных обычно хранятся в файловой системе, но не все объекты содержатся в каталоге. Исключения составляют определения хранимых процедур и событий, хранящиеся непосредственно в базе данных *mysql*, а также в зависимости от механизма хранения, данные могут и не быть размещены в каталоге с информацией о базе данных.

Поэтому, если вы решите перемещать базу данных, перемещая ее каталог, убедитесь, что в этом случае действительно будут перенесены все объекты и все данные.

В главе 12 мы обсудим различные варианты резервного копирования, поэтому не будем останавливаться на них здесь. Заметьте, что при проектировании решения не следует привязывать процедуру к определенному методу резервного копирования, поскольку в дальнейшем может оказаться, что другие способы резервного копирования подходят вам больше.

Чтобы реализовать уже описанную процедуру резервного копирования, необходимо иметь механизм для перевода шарда в автономный режим, то есть возможность на время запретить обновления шарда. С этой целью можно либо заблокировать шард из приложения, либо заблокировать таблицы базы данных

Реализация блокировки из приложения требует такого координирования всех запросов, которое способно предотвратить возникновение конфликтов. А поскольку веб-приложения по своей сути являются распределенными, управление блокировкой может в скором времени стать весьма непростым.

В нашем примере мы упростим задачу, заблокировав единственную таблицу — *shard_to_node* — вместо того, чтобы распространить блокировку на различные таблицы, доступ к которым получает множество клиентов. В основном поиск местоположения шардов ведется с использованием таблицы *shard_to_node*, поэтому этой единственной блокировки должно быть достаточно, чтобы никакие обновления не поступали на шарды, пока мы их перемещаем и заново сопоставляем с узлами. Вероятно, некоторые обновления окажутся в процессе исполнения — либо они уже начали применяться к шарду, либо вот-вот начнут применяться. Когда шард блокируется, уже выполняющиеся обновления будут завершены, а обновления, которые еще не успели стартовать, будут ждать снятия блокировки. Когда блокировка снимется, шарда на прежнем месте уже не окажется, поэтому все операторы обновления потерпят неудачу и будут вынуждены выполняться заново уже на перемещенном шарде.

Для автоматизации описанной процедуры имеет смысл воспользоваться библиотекой Replicant (см. лист. 5-13).

Лист. 5-13. Процедура для перемещения шарда между узлами

```
_UPDATE_SHARD_MAP = """
UPDATE shard_to_node
   SET host = %s, port = %d, sock = %s
WHERE shard = %d
"""

_LOCK_SHARD_MAP = """
BEGIN;
SELECT host, port, sock
   FROM shard_to_node
WHERE shard = %d FOR UPDATE
"""
```

```
_UNLOCK_SHARD_MAP = "COMMIT"

def lock_shard(server, shard):
   server.use("common")
   server.sql(_LOCK_SHARD_MAP, (shard))

def unlock_shard(server):
   server.sql(_UNLOCK_SHARD_MAP)

def move_shard(common, shard, source, target, backup_method):
backup_pos = backup_method.backup_to()
config = target.fetch_config()
config.set('replicate-do-db', shard)
target.stop().replace_config(config).start()
replicant.change_master(target, source, backup_pos)
replicant.slave_start(target)

# Ожидает, пока запаздывание подчиненного сервера от
# главного не составит по меньшей мере 10 секунд
   replicant.slave_status_wait_until(target,
            'Seconds_Behind_Master', lambda x: x < 10)
   lock_shard(common, shard)
pos = replicant.fetch_master_pos(source)
replicant.slave_wait_for_pos(target, pos)
   lock_database(target, shard_name)
   common.sql(_UPDATE_SHARD_MAP,
            (target.host, target.port, target.socket, shard))
   unlock_shard(common, shard)
   source.sql("DROP DATABASE shard_%s", (shard))
```

Как отмечалось ранее, необходимо помнить, что хотя таблица заблокирована, некоторые клиентские сеансы могут работать с ней, если они уже получили местоположение узла, но еще не подключились к нему, или только начали обновлять шард.

Код приложения учитывает этот момент. Проще всего заставить приложение заново вычислить узел, если запрос к шарду провалится. Допустим, что запрос к шарду оказывается неудачным именно тогда, когда узел перенесен и, следовательно, поиск его местоположения следует повторить. В лист. 5-14 показано, как следует скорректировать функцию getArticlesForUser.

Лист. 5-14. Изменения кода приложения, необходимые для поддержки переноса шарда

```
function getArticlesForUser($userId, $common)
{
   global $QUERIES;
   $query = <<<END_OF_SQL
SELECT author_id, article_id, title, published, body
   FROM articles
  WHERE author_id = %d
END_OF_SQL;
```

```
    do {
        list($shard, $node) = getShardAndNodeFromUserId($userId, $common);
        $articles = array();
        $QUERIES[] = sprintf($query, $userId);
        $result = mysql_query(sprintf($query, $userId), $node);
    } while (!$result && mysql_errno($node) == 1146);
    while ($obj = mysql_fetch_object($result))
        $articles[] = $obj;
    return $articles;
}
```

Как мы видели в предыдущем разделе, иногда, если пользователь внезапно обретает популярность, приходится также переносить между шардами отдельные блоки данных.

Переместить пользователя намного сложнее, чем переместить шард, поскольку при этом необходимо извлечь пользователя и все связанные с ним статьи и комментарии из одного шарда и добавить эти данные в другой шард. Детали реализации этой процедуры во многом зависят от конкретного приложения, поэтому мы предложим вам только общие рекомендации.

Рассмотрим методологию перемещения пользователя из исходного шарда в целевой. Эта процедура разработана для таблицы, поддерживающей блокировку на уровне строк (например, под управлением InnoDB), поэтому при перемещении пользователя между таблицами MyISAM блокировку придется обрабатывать по-другому. Соответствующий код на языке Python несложен, поэтому здесь приводится только SQL-листинг.

Если исходный и целевой шарды расположены на одном и том же узле, перенос пользователя без труда выполняется при помощи процедуры, представленной ниже. Предположим, что базы данных содержат номера соответствующих шардов. Используем для имен старого и нового шардов поля подстановки *old* и *new*, а для пользователя — подстановку *UserID*.

1. Чтобы заблокировать сеансы, стремящиеся получить доступ к нашему пользователю, заблокируйте строку с информацией о пользователе в базе данных *common*.

```
common> BEGIN;
common> SELECT shard INTO @old_shard
     -> FROM common.user
     -> WHERE user_id = UserID FOR UPDATE;
```

2. Переместите статьи и комментарии пользователя из старого шарда в новый.

```
shard> BEGIN;
shard> INSERT INTO shard_new.articles
    ->   SELECT * FROM shard_old.articles
    ->   WHERE author_id = UserID
```

```
    ->  FOR UPDATE;
shard> INSERT INTO  shard_new.comments(comment_id, article_ref, author_name,
    ->                                 body, published)
    ->   SELECT comment_id, article_ref, author_name, body, published
    ->      FROM shard_old.comments, shard_old.articles
    ->      WHERE article_id = article_ref AND user_id = UserID;
```

3. Обновите информацию о пользователе так, чтобы она указывала на новый шард.

```
common> UPDATE common.user SET shard = new WHERE user_id = UserID;
common> COMMIT;
```

4. Удалите статьи пользователя и комментарии к ним из старого шарда.

```
shard> DELETE FROM shard_old.comments
    ->      USING shard_old.articles, shard_old.comments
    ->   WHERE article_ref = articles_id AND author_id = UserID;
shard> DELETE FROM shard_old.articles WHERE author_id = UserID;
shard> COMMIT;
```

В этом случае необходимо поддерживать два подключения открытыми: одно подключение для узла, содержащего общую базу данных, а второе — для узла, содержащего шарды. Если шарды и общая база данных находятся на одном узле, задача существенно упрощается, но не следует на это рассчитывать.

Если шарды находятся в разных базах данных, приведенный ниже алгоритм решает проблему относительно просто.

1. Создайте резервную копию статей и комментариев исходного узла, и одновременно получите позицию двоичного журнала, соответствующую резервной копии.

   Для этого заблокируйте строки пользователя в обеих таблицах *articles* и *comments*. Обратите внимание на то, что необходимо открыть транзакцию, аналогичную той, в которой мы при перемещении шарда обновляли таблицу *shard_to_node*, но в данном случае достаточно заблокировать только операции записи, не блокируя чтение.

```
shard_old> BEGIN;
shard_old> SELECT * FROM articles, comments
        -> WHERE article_ref = article_id AND author_id = UserID
        ->  FOR UPDATE;
```

2. Создайте резервную копию статей и комментариев.

```
shard_old> SELECT * INTO OUTFILE 'UserID-articles.txt' FROM articles
        -> WHERE author_id = UserID;
shard_old> SELECT * INTO OUTFILE 'UserID-comments.txt' FROM comments
        -> WHERE article_ref = article_id AND author_id = UserID;
```

3. Скопируйте сохраненные статьи и комментарии на новый узел и перепишите их в новый шард при помощи команды LOAD DATA INFILE.

```
shard_new> LOAD DATA INFILE 'UserID-articles.txt' INTO articles;
shard_new> LOAD DATA INFILE 'UserID-comments.txt' INTO comments;
```

4. Обновите местоположение шарда, соответствующего пользователю, в общей базе данных.

```
common> UPDATE user SET shard = new WHERE user_id = UserID;
```

5. Удалите статьи пользователя и комментарии к ним в старом шарде, также как в предыдущей процедуре.

```
shard_old> DELETE FROM comments USING articles, comments
        -> WHERE article_ref = articles_id AND author_id = UserID;
shard_old> DELETE FROM articles WHERE author_id = UserID;
shard_old> COMMIT;
```

Управление согласованностью данных

Как обсуждалось ранее в главе, одной из проблем, связанных с асинхронной репликацией, является обеспечение согласованности данных. Проиллюстрируем эту проблему следующим примером. Представьте, что вы обслуживаете сайт веб-магазина, при работе с которым пользователи просматривают товары, которые они хотели бы приобрести, и добавляют понравившиеся товары в корзину. Ваше серверы настроены так, что при добавлении товара в корзину запрос на изменение поступает на главный сервер, но когда веб-сервер запрашивает информацию о содержимом корзины, запрос посылается одному из подчиненных серверов, предназначенных для обработки подобных запросов. Поскольку главный сервер опережает подчиненный, не исключено, что изменение еще не поступило на подчиненный сервер, а тогда запрос к подчиненному серверу обнаружит пустую корзину, что, несомненно, сильно удивит покупателя, который в таком случае сразу же добавит товар в корзину снова. В результате покупатель обнаружит, что его корзина содержит *две* единицы товара, поскольку за прошедший интервал времени подчиненный сервер успеет принять и реплицировать оба изменения информации о корзине. Если вы не хотите иметь дело с толпой раздраженных пользователей, такой ситуации следует избегать.

Чтобы не выбирать устаревшие данные, необходимо тем или иным образом убедиться, что данные, пришедшие от подчиненного сервера, достаточно свежие и подходят для работы. Как вы вскоре убедитесь, проблема оказывается еще более изощренной, когда в системе задействован и сервер-ретранслятор. Основная идея решения заключается в том, чтобы некоторым образом помечать все транзакции, зафиксированные на главном сервере, и перед выполнением запроса на подчиненном сервере дождаться, пока он повторит эту транзакцию (можно ждать и дольше).

Конкретный метод решения зависит от того, участвуют ли в системе в качестве промежуточного слоя подчиненные серверы-ретрансляторы.

Согласованность данных в неиерархическом развертывании

Когда все подчиненные серверы подключены непосредственно к главному, согласованность данных несложно проверить. В этом случае достаточно записать позицию двоичного журнала после фиксации транзакции, а затем подождать, пока подчиненный сервер достигнет этой позиции, воспользовавшись описанной ранее функцией MASTER_POS_WAIT. Однако получить точную позицию транзакции, записанной в двоичный журнал, невозможно. Почему? Потому что за время между фиксацией транзакции и выполнением команды SHOW MASTER STATUS, в двоичный журнал будут записаны еще несколько событий.

На самом деле это не имеет значения, поскольку у нас нет необходимости получать точную позицию транзакции в двоичном журнале. Достаточно получить позицию, которая *совпадет* с точной позицией транзакции или будет располагаться *после нее*. Поскольку команда SHOW MASTER STATUS показывает позицию, на которой репликация пишет события в текущий момент, достаточно выполнить эту команду после того, как интересующая нас транзакция будет зафиксирована, — мы получим такую позицию журнала, которую уже можно использовать для проверки согласованности данных.

В лист. 5-15 приведен PHP-код, помогающий выполнить обновления таким образом, чтобы извлекаемые данные наверняка были свежими.

Лист. 5-15. Код на языке PHP, позволяющий избежать чтения устаревших данных

```
function fetch_master_pos($server) {
   $result = $server->query('SHOW MASTER STATUS');
   if ($result == NULL)
      return NULL;                          // Выполнение потерпело неудачу
   $row = $result->fetch_assoc();
   if ($row == NULL)
      return NULL;                          // Двоичный журнал недоступен
   $pos = array($row['File'], $row['Position']);
   $result->close();
   return $pos;
}

function sync_with_master($master, $slave) {
   $pos = fetch_master_pos($master);
   if ($pos == NULL)
      return FALSE;
   if (!wait_for_pos($slave, $pos[0], $pos[1]))
      return FALSE;
   return TRUE;
}
```

```
function wait_for_pos($server, $file, $pos) {
   $result = $server->query("SELECT MASTER_POS_WAIT('$file', $pos)");
   if ($result == NULL)
      return FALSE;                  // Выполнение потерпело неудачу
   $row = $result->fetch_row();
   if ($row == NULL)
      return FALSE;                  // Выходной результат пуст ?!
   if ($row[0] == NULL || $row[0] < 0)
      return FALSE;                  // Синхронизация потерпела неудачу
   $result->close();
   return TRUE;
}

function commit_and_sync($master, $slave) {
   if ($master->commit()) {
      if (!sync_with_master($master, $slave))
         return NULL;                // Синхронизация потерпела неудачу
      return TRUE;                   // Фиксация и синхронизация выполнены
                                     // успешно
   }
   return FALSE;                     // Фиксация потерпела неудачу
                                     // (синхронизация не выполнялась)
}

function start_trans($server) {
   $server->autocommit(FALSE);
}
```

В лист. 5-15 представлены функции commit_and_sync и start_trans, а также три вспомогательные функции fetch_master_pos, wait_for_pos и sync_with_master. Функция commit_and_sync фиксирует транзакцию и ожидает, пока транзакция достигнет предназначенный ей подчиненный сервер. Эта функция принимает два параметра — объект, представляющий собой подключение к главному серверу, и объект — подключение к подчиненному серверу. Она возвращает значение TRUE, если фиксация и синхронизация были выполнены успешно, значение FALSE, если фиксация потерпела неудачу, и значение NULL, если фиксация выполнилась успешно, но синхронизация не удалась (либо потому, что на подчиненном сервере произошла ошибка, либо потому, что подчиненный сервер потерял связь с главным).

Функция фиксирует текущую транзакцию, и затем, если эта операция закончилась успешно, получает текущую позицию двоичного журнала главного сервера при помощи команды SHOW MASTER STATUS. Поскольку другие потоки могли выполнить обновления базы данных между фиксацией и вызовом команды SHOW MASTER STATUS, возвращаемая позиция, скорее всего, соответствует уже не окончанию транзакции, а находится в журнале несколько дальше. Как уже говорилось, бóльшая точность и не требуется, когда мы достигнем этой позиции, транзакция будет уже выполнена.

После того, как позиция двоичного журнала главного сервера получена, функция подключается к подчиненному серверу и при помощи функции MASTER_POS_WAIT ожидает поступления позиции главного сервера. Если подчиненный сервер работает, в момент обращения к этой функции выполнение останавливается и программа переходит в состояние ожидания указанной позиции, если же подчиненный сервер не функционирует, немедленно возвращается значение NULL. То же самое произойдет, если подчиненный сервер будет остановлен в процессе ожидания, к примеру, если при выполнении некого оператора произойдет ошибка. В любом случае значение NULL указывает на то, что транзакция не достигла подчиненного сервера, поэтому необходимо проверять возвращаемый результат. Если функция MASTER_POS_WAIT возвращает 0, значит подчиненный сервер получил транзакцию, следовательно синхронизация фактически уже завершилась успешно.

Чтобы работать по описанной схеме, достаточно подключиться к серверу как обычно, а затем открывать, фиксировать или отменять транзакцию при помощи предложенных функций. Применение этих функций продемонстрировано в лист. 5-16, но проверка ошибок опущена, поскольку она зависит от реализации поддержки ошибок.

Лист. 5-16. Обращение к функциям start_trans и commit_and_sync

```
require_once './database.inc';

start_trans($master);
$master->query('INSERT INTO t1 SELECT 2*a FROM t1');
commit_and_sync($master, $slave);
```

Согласованность данных в иерархическом развертывании

Поддержка согласованности данных в иерархическом развертывании существенно отличается от поддержки согласованности данных в простой топологии репликации, где все подчиненные серверы подключены непосредственно к главному. В данном случае нельзя опираться на позицию двоичного журнала главного сервера, поскольку каждый промежуточный сервер-ретранслятор эту позицию изменяет. Нам нужен другой способ, помогающий дождаться выполнения транзакции. Функция MASTER_POS_WAIT довольно удобна для этой цели, поэтому если ее удастся использовать, проблем станет намного меньше. По сути, имеются только два варианта, позволяющие избежать чтения устаревших данных.

В первом решении при помощи глобального идентификатора транзакции, описанного в главе 4, отслеживается продвижение подчиненного сервера, а также периодически производится его опрос, что позволяет установить факт выполнения им транзакции.

Во втором решении, представленном на рис. 5-11, выполняется подключение ко всем серверам-ретрансляторам, расположенным между главным и конечным подчиненным серверами, и проверяется, дошло ли изменение до подчиненного сервера. Необходимо подключиться к каждому подчиненному серверу-ретранслятору, расположенному по пути от главного сервера до интересующего нас подчиненного сервера, поскольку невозможно узнать, какая позиция двоичного журнала будет использована тем или иным промежуточным сервером-ретранслятором.

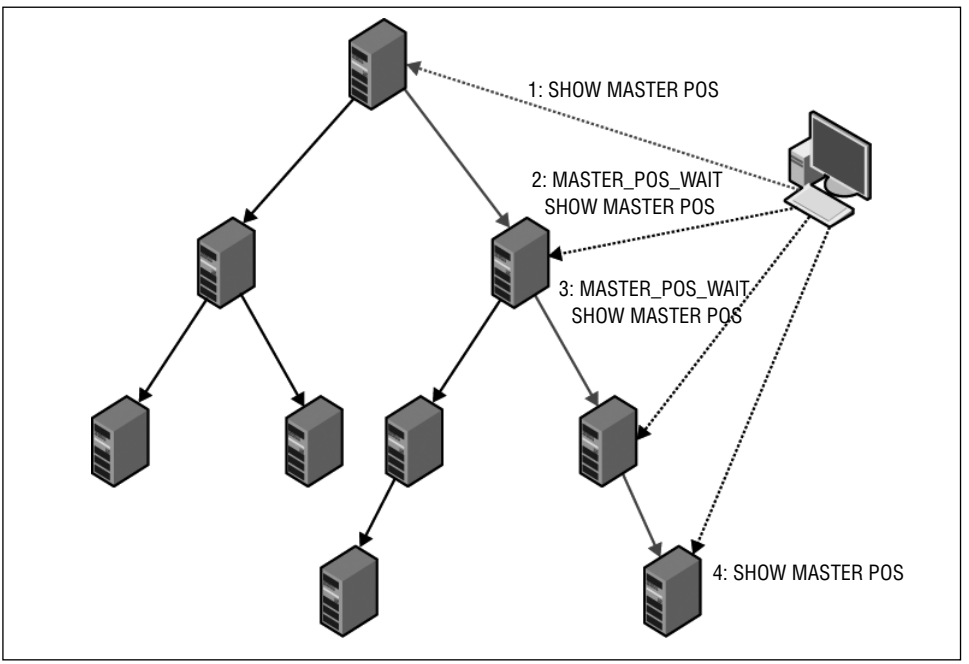


Рис. 5-11. Синхронизация серверов по цепочке ретрансляторов

Оба решения имеют свои достоинства, поэтому мы рассмотрим преимущества и недостатки каждого.

Если подчиненные серверы системы относительно главного сервера обновляются без задержки, первая же проверка конечного подчиненного сервера, выполняемая в первом алгоритме, скорее всего, покажет, что транзакция уже реплицирована на подчиненный сервер и обработку можно продолжить. Если транзакция еще не выполнялась, вероятно, она будет выполнена перед следующей проверкой, поэтому во время второй проверки конечного подчиненного сервера будет показано, что транзакция его достигла. Если интервал проверки достаточно мал, задержка незаметна для пользователя, и стандартная проверка согласованности потребует всего лишь одно или два дополнительных сообщения при опросе конечного подчиненного сервера. При этом подходе требуется опрос только конечного подчиненного сервера, а не промежуточных серверов. Эта особенность может рассматриваться как

преимущество с точки зрения администратора, поскольку не требуется хранить маршруты промежуточных подчиненных серверов и информацию об их подключении.

С другой стороны, если подчиненные сервера системы отстают, или если запаздывание репликации существенно варьируется, возможно, больше подходит второй алгоритм. Если к подобной системе применить первый алгоритм и периодически выполнять опрос подчиненного сервера, большую часть времени опрос будет показывать, что транзакция на подчиненном сервере *не* зафиксирована. Для устранения этой проблемы можно увеличить интервал опроса, но если интервал опроса вырастает до такой степени, что время отклика оказывается неприемлемым, ясно, что решение не годится. В этом случае лучше обратиться ко второму алгоритму — ждать, пока изменения спустятся по дереву репликации, а затем выполнять запрос.

Для дерева размера *N* количество дополнительных запросов пропорционально *log N*. Например, если ваша система включает 50 серверов-ретрансляторов и каждый сервер-ретранслятор обслуживает 50 конечных серверов, вы можете разобраться со всеми 2 500 подчиненными серверами при помощи всего двух дополнительных запросов — одного, обращенного к подчиненному серверу-ретранслятору, и другого — к конечному серверу.

Второй подход имеет свои недостатки:

- Код приложения должен иметь доступ ко всем подчиненным серверам-ретрансляторам, чтобы подключаться к ним по очереди и выжидать, пока нужная позиция репликации будет достигнута.
- Код приложения должен хранить конфигурацию репликации, чтобы опрашивать подчиненные серверы-ретрансляторы.

Запросы, направленные подчиненным серверам-ретрансляторам, замедлят их работу, поскольку увеличится загруженность, но на практике это не обязательно выльется в проблему. Если добавить слой кэширования подключений к базе данных, лишнего трафика можно избежать. Каждый раз при получении запроса позиция двоичного журнала будет запоминаться в слое кэширования, и ретранслятор будет опрашиваться только тогда, когда позиция двоичного журнала будет больше, чем все кэшированные. Общий план функции кэширования может выглядеть так:

```
function wait_for_pos($server, $wait_for_pos) {
   if (кэшированная позиция для $server > $wait_for_pos)
      return TRUE;
   else {
      код ожидания позиции и обновления кэша
   }
}
```

Поскольку позиции двоичного журнала всегда возрастают, — пройденная позиция двоичного журнала навсегда останется пройденной, — некорректный результат вернуть невозможно. Чтобы выяснить, какой из методов более эф-

фективен, необходимо настроить мониторинг и профилирование развертывания, — тогда станет ясно, достаточно ли быстро выполняются запросы.

В лист. 5-17 приведен код для реализации первого решения, — чтобы выяснить, выполнена ли транзакция, подчиненный сервер периодически опрашивается. В коде выполняется обращение к таблице *Last_Exec_Trans*, описанной в главе 4, — сначала ее содержимое проверяется на главном сервере, а затем, до тех пор, пока нужная транзакция не будет обнаружена, обращение к этой таблица периодически выполняется на подчиненном сервере.

Лист. 5-17. PHP-код, позволяющий избежать чтения устаревших данных при помощи регулярного опроса

```
function fetch_trans_id($server) {
   $result = $server->query('SELECT server_id, trans_id FROM Last_Exec_Trans');
   if ($result == NULL)
      return NULL;                // Выполнение потерпело неудачу
   $row = $result->fetch_assoc();
   if ($row == NULL)
      return NULL;                // Пустая таблица !?
   $gid = array($row['server_id'], $row['trans_id']);
   $result->close();
   return $gid;
}

function wait_for_trans_id($server, $server_id, $trans_id) {
   if ($server_id == NULL || $trans_id == NULL)
      return TRUE;    // Нет выполненных транзакций, синхронизация тривиальна
   $server->autocommit(TRUE);
   $gid = fetch_trans_id($server);
   if ($gid == NULL)
      return FALSE;
   list($current_server_id, $current_trans_id) = $gid;
   while ($current_server_id != $server_id || $current_trans_id < $trans_id) {
      usleep(500000);                          // Ожидание 0,5 с
      $gid = fetch_trans_id($server);
      if ($gid == NULL)
         return FALSE;
      list($current_server_id, $current_trans_id) = $gid;
   }
   return TRUE;
}

function commit_and_sync($master, $slave) {
   if ($master->commit()) { $gid = fetch_trans_id($master);
      if ($gid == NULL)
         return NULL;
      if (!wait_for_trans_id($slave, $gid[0], $gid[1]))
         return NULL;
```

```
      return TRUE;
   }
   return FALSE;
}

function start_trans($server) {
   $server->autocommit(FALSE);
}
```

Функции commit_and_sync и start_trans сохранили логику, показанную еще в лист. 5-15, соответственно и работать с ними можно, как предлагается в лист. 5-16. Разница заключается в том, что здесь эти функции обращаются к функциям fetch_trans_id и wait_for_trans_id, а не к fetch_master_pos и wait_for_pos. Некоторые нюансы кода стоит обсудить:

- Мы отключили автофиксацию до начала опроса подчиненного сервера в функции wait_for_trans_id. Это неоходимо, поскольку при уровне изоляции, начиная с repeatable read, SELECT каждый раз будет обнаруживать один и тот же глобальный идентификатор транзакции.
- Чтобы предотвратить такую ситуацию, мы фиксируем каждый SELECT как отдельную транзакцию, посредством включения автофиксации. Вместо этого можно установить уровень изоляции read committed
- Чтобы в функции wait_for_trans_id избежать ненужного ожидания, мы еще до цикла в первый раз получаем глобальный идентификатор транзакции и проверяем его.
- Код должен иметь доступ только к главному и подчиненному серверам, а не к промежуточным серверам-ретрансляторам.

В лист. 5-18 представлен код, позволяющий убедиться, что данные выборки не будут устаревшими. Здесь опрашиваются все сервера между главным и конечным подчиненным сервером. Сначала определяется полная цепочка серверов между конечным подчиненным сервером и главным сервером, а затем выполняется синхронизация серверов по очереди, прохождением всего пути вниз по цепочке, пока транзакция не достигнет конечного подчиненного сервера. В коде используются обращения к функциям fetch_master_pos и wait_for_pos из лист. 5-13, здесь их код не приводится. Кэширующий слой в этом коде не реализован.

Лист. 5-18. PHP-код, позволяющий избежать чтения устаревших данных при помощи ожидания

```
function fetch_relay_chain($master, $final) {
   $servers = array();
   $server = $final;
   while ($server !== $master) {
      $server = get_master_for($server);
      $servers[] = $server;
   }
```

```
   $servers[] = $master;
   return $servers;
}

function commit_and_sync($master, $slave) {
   if ($master->commit()) {
      $server = fetch_relay_chain($master, $slave);
      for ($i = sizeof($server) - 1; $i > 1 ; --$i) {
      if (!sync_with_master($server[$i], $server[$i-1]))
         return NULL;                    // Синхронизация потерпела неудачу
      }
   }
}

function start_trans($server) {
   $server->autocommit(FALSE);
}
```

Чтобы собрать все сервера между главным и подчиненным сервером, воспользуемся функцией fetch_relay_chain. В качестве исходной точки эта функция берет интересующий нас подчиненный сервер, и последовательно получает следующие главные при помощи функции get_master_for. Мы сознательно опустили код этой функции, поскольку его обсуждение не внесет в нашу дискуссию ничего нового. Однако чтобы код мог выполняться, функцию надо определить.

Когда цепочка ретрансляторов построена, проходом вниз выполняется синхронизация главного и подчиненного сервера. Эту задачу решает функция sync_with_master, показанная в лист. 5-15.

Чтобы по подчиненному серверу определить его главный сервер, можно вызвать команду SHOW SLAVE STATUS, а затем воспользоваться информацией из полей Master_Host и Master_Port. Но выполнение этой операции для каждой фиксируемой транзакции серьезно повлияет на производительность системы.

Поскольку топология меняется редко, имеет смысл кэшировать такую информацию в приложении или где-либо в другом месте во избежание чрезмерного трафика к серверам базы данных.

В главе 4 было показано, как следует действовать в случае отказа главного сервера, — например, передать его функции другому главному серверу или перевести подчиненный сервер в ранг главного. Мы также отметили, что следует вернуть главный сервер в развертывание, как только он будет отремонтирован. Главный сервер является важнейшим звеном развертывания, и скорее всего он размещен на более мощном компьютере, чем подчиненные сервера, поэтому вернуть этот компьютер в развертывание следует тоже в качестве главного сервера. Поскольку остановка главного сервера не была запланирована, нельзя исключить, что теперь он рассинхронизирован с остальной частью развертывания. Рассинхронизация могла произойти по двум причинам:

- Если главный сервер оставался в автономном режиме достаточно долго, другие серверы выполнили множество транзакций, о которых этот главный сервер не знает. Можно сказать, что главный сервер оказался в *альтернативном будущем* по сравнению с остальной частью системы. Эта ситуация проиллюстрирована на рис. 5-12.
- Если главный сервер зафиксировал транзакцию и занес ее в двоичный журнал, а сражу же после этого вышел из строя, возможно, транзакция к подчиненным серверам не попала. Это означает, что несколько транзакций, прошедших на главном сервере, не видны ни подчиненным серверам, ни где-либо еще в системе.

Если исходный главный сервер не слишком отстал от главного сервера, выполняющего его обязанности на текущий момент, из первой ситуации можно выйти следующим образом — подключить исходный главный сервер к текущему в качестве подчиненного, а как только прежний главный сервер догонит систему, переключить все подчиненные сервера снова на него. Однако, если первоначальный главный сервер находился в автономном режиме слишком долго, имеет смысл клонировать его из какого-либо подчиненного сервера — это будет быстрее, — и затем вернуть ему подчиненные сервера.

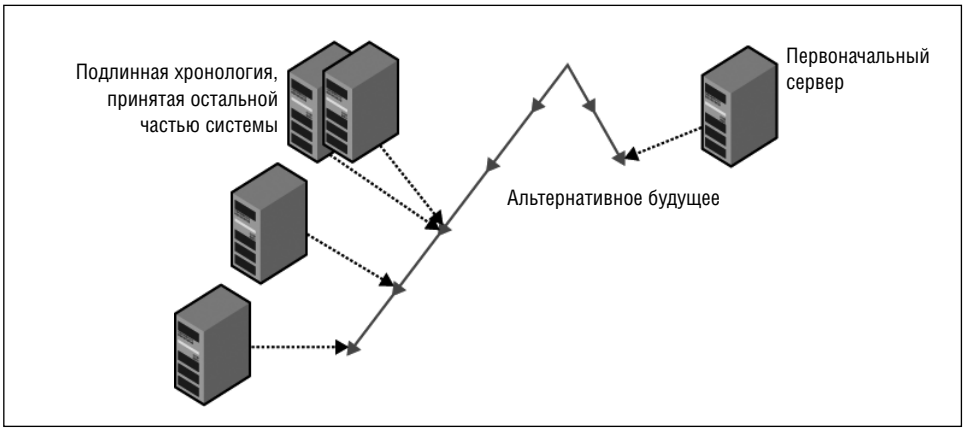


Рис. 5-12. Первоначальный главный сервер, оказавшийся в альтернативном будущем

Если главный сервер попал в альтернативное будущее, вряд ли развертывание сможет принять его дополнительные транзакции. Почему? Дело в том, что новая транзакция, неожиданно введенная в систему, с большой степенью вероятности будет конфликтовать с существующими транзакциями, причем самым неожиданным образом. Например, если транзакция — объявление на доску объявлений, не исключено, что пользователь уже отправил объявление повторно. Если ранее созданное объявление сначала пропало, поскольку главный сервер вышел из строя, не успев отправить его подчиненным серверам, а затем повторно появилось, это удивит пользователей и не понравится им. Также пользователи не придут в восторг, если в их корзинах внезапно появятся товары, которые вернул в систему восстановленный главный сервер.

Короче говоря, из обеих ситуаций рассинхронизации — главный сервер в альтернативном будущем или же главный сервер, который должен догнать систему, — можно выйти, просто клонировав подчиненный сервер в исходный главный, а затем переключить на него все текущие подчиненные сервера.

Вышесказанное призвано подчеркнуть, как важно с точки зрения поддержки аварийных ситуаций обеспечивать согласованность данных, проверяя, что изменения главного сервера переданы хотя бы какой-то части системы перед тем, как главный сервер признает транзакцию завершенной. Код, который мы обсуждали в этой главе, разрабатывался в предположении, что пользователь попытается прочитать данные сразу. В этом случае необходимо убедиться, что изменения попали на подчиненный сервер до получения им запроса на чтение данных. С точки зрения восстановления системы такой алгоритм представляется избыточным, — достаточно знать, что транзакция доступна хотя бы на одном из оставшихся компьютеров, например, на одном из подчиненных серверов или подчиненных серверов-ретрансляторов, подключенных к главному серверу. В общем случае можно перенести n-1 сбой, если изменение попало на n серверов.

Заключение

В этой главе мы обсуждали повышение пропускной способности приложений путем горизонтального масштабирования — когда для обработки большего количества запросов в систему добавляются дополнительные серверы. Перечислялись способы реализации масштабирования при помощи репликации MySQL, а некоторые концепции сопровождались примерами из жизни. В следующей главе мы рассмотрим более сложные варианты репликации.

Джоэл услышал легкий стук, и в дверях показался г-н Саммерсон.

— Мне понравился ваш отчет по горизонтальному масштабированию наших серверов, Джоэл. Я бы хотел, чтобы вы приступили к реализации прямо сейчас. Воспользуйтесь нашими свободными серверами из компьютерного зала.

Джоэл был настолько доволен, что решился задать шефу вопрос.

— Да, сэр. Когда их следует подключить?

Г-н Саммерсон улыбнулся и взглянул на часы.

— Рабочий день еще не кончился, — бросил он, удаляясь.

Джоэл не понял, шутка ли это, поэтому решил приступить к делу, не мешкая. Прихватив уже изрядно потертый экземпляр книги «*Высокая доступность MySQL*» и свои записи, он направился в компьютерный зал. «Надеюсь, я настроил свой медиа-центр на запись», — пробормотал он, предчувствуя, что застрянет на работе до поздней ночи.

ГЛАВА 6

Дополнительные возможности репликации

Джоэл просматривал почту, когда в дверь постучали. Появление на пороге г-на Саммерсона уже не вызывало у него ни малейшего удивления.

— Да, сэр?

— Джоэл, все-таки меня беспокоит наша хитроумная репликация. Пожалуй, тебе следует понять, чего мы о ней еще не знаем из того, что стоило бы знать. Хотелось бы иметь единый пакет, включающий не только описание текущей конфигурации, но и детальный анализ возможных проблем и путей их решения. Надо же знать, что делать, если вдруг что-то пойдет не так...

Джоэл ожидал чего-то подобного. Последнее время он и сам думал, что неплохо было бы узнать о репликации побольше.

— Приступаю прямо сейчас, сэр.

— Отлично. Это твоя основная задача. Я жду результата.

Джоэл кивнул, глядя вслед удаляющемуся боссу. Вздохнув, он сгреб в охапку свои любимые книги по MySQL. Чтобы узнать о репликации все, ему еще читать и читать...

В предыдущих главах мы обсудили основы настройки и развертывания репликации — эти знания помогут вам поддерживать свой сайт в рабочем и доступном состоянии, но чтобы справиться с подводными камнями репликации и применять ее максимально эффективно, вы должны получить как можно больше самых разнообразных сведений об этом предмете. Настоящая глава охватывает обширный материал, в том числе:

- Как максимально надежно повысить подчиненный сервер до главного.
- Как избежать повреждений баз данных после сбоев.
- Репликация с множеством источников.
- Построчная репликация.

Основы архитектуры репликации

В главе 3 мы обсудили двоичный журнал и некоторые инструменты для исследования записанных в нем событий. Но мы не говорили о том, как со-

бытия передаются подчиненному серверу и повторно выполняются уже на нем. Разобравшись в деталях этой процедуры, вы обретете дополнительный контроль над репликацией, сможете предотвращать повреждение данных после сбоев и выявлять истоки проблем, анализируя журналы.

На рис. 6-1 приведена схема внутренней архитектуры репликации, включающая клиентов, подключенных к главному серверу, сам главный сервер и несколько подчиненных серверов. Для каждого подключенного клиента главный сервер запускает сеанс, который отвечает за выполнение всех SQL-операторов и возращение результатов клиенту.

События перетекают от главного сервера к подчиненным следующим образом:

1. Сеанс принимает от клиента оператор, выполняет его и синхронизируется с другими сеансами, чтобы гарантировать, что все транзакции выполнены без конфликтов с изменениями, внесенными другими сеансами.
2. Непосредственно перед завершением выполнения оператора в двоичный журнал записывается одно или несколько событий. Эта процедура подробно разобрана в главе 2, здесь мы не будем к ней возвращаться.
3. После того, как события записаны в двоичный журнал, принимает управление *поток дампа* на главном сервере, который считывает события из двоичного журнала и посылает их потоку ввода-вывода подчиненного сервера.
4. Получив событие, поток ввода-вывода подчиненного сервера заносит его в конец журнала ретрансляции.
5. После того как событие оказалось в журнале ретрансляции, его оттуда считывает поток SQL подчиненного сервера, который затем выполняет код, содержащийся в событии, чтобы применить изменения к базе данных на подчиненном сервере.

Если подключение к главному серверу разрывается, поток ввода-вывода пытается заново подключиться к серверу, аналогично любому клиентскому потоку MySQL. Некоторые параметры, описываемые в этой главе, относятся именно к попыткам повторного подключения.

Структура журнала ретрансляции

Как было показано в предыдущем разделе, журнал ретрансляции хранит информацию, связывающую главный и подчиненный серверы, — информацию, ключевую для репликации. Следует четко представлять, как этот журнал используется, в частности, каким образом он участвует в координации потоков подчиненного сервера. Поэтому мы в подробностях обсудим структуру журнала ретрансляции и выясним, как работают с ним потоки подчиненного сервера при репликации.

Как вы уже знаете из предыдущего раздела, события, посылаемые главным сервером, заносятся в журнал ретрансляции потоком ввода-вывода. Журнал ретрансляции играет роль буфера, поэтому главный сервер может послать следующее событие, не дожидаясь завершения операции подчиненным сервером.

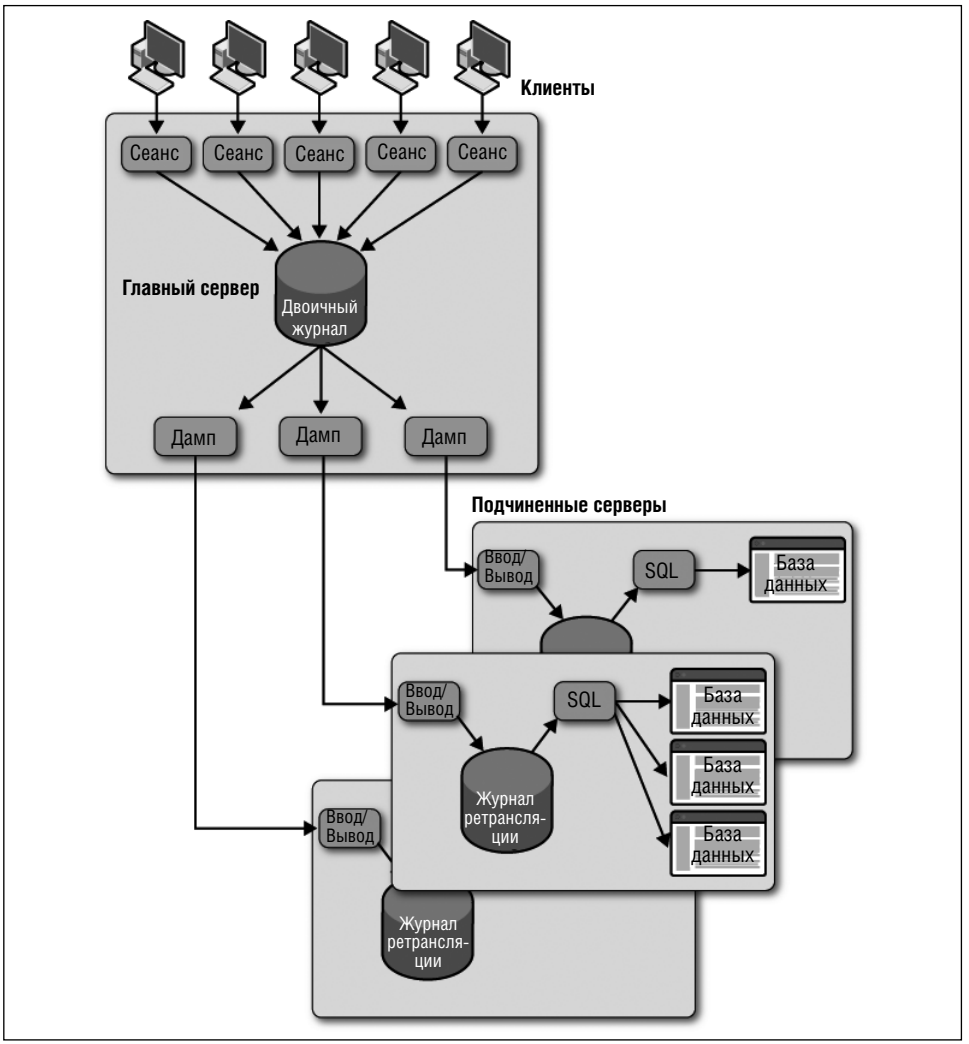


Рис. 6-1. Внутренняя структура системы, состоящей из главного и подчиненных серверов

На рис. 6-2 журнал ретрансляции представлен в схематическом виде. Его структура аналогична структуре двоичного журнала главного сервера, но журнал ретрансляции включает несколько дополнительных файлов.

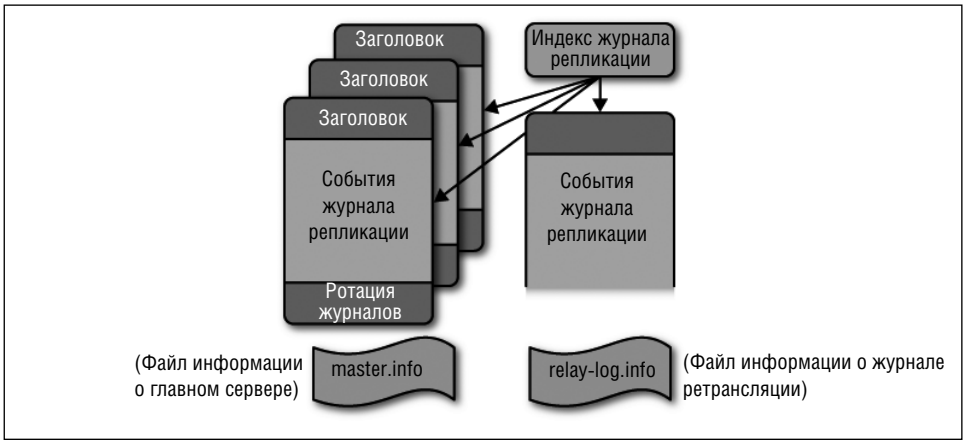


Рис. 6-2. Структура журнала репликации

Помимо собственно файлов с содержимым и индексных файлов (присутствующих и в двоичном журнале), журнал ретрансляции содержит еще два файла, предназначенные для отслеживания хода ретрансляции, — *файл информации о журнале ретрансляции* (*relay log information file*) и *файл информации о главном сервере* (*master log information file*). Имена этих файлов задаются в файле конфигурации *my.cnf* двумя параметрами:

- *relay-log-info-file=имя_файла* Задает имя файла информации о журнале ретрансляции. С этим именем можно работать и через серверную неизменяемую переменную relay_log_info_file. Если не задано абсолютное (полное) имя файла, подразумевается, что имя указано относительно каталога данных сервера. Значение по умолчанию — *relay-log.info*.
- *master-info-file= имя_файл* Задает имя файла информации о главном сервере. Значение по умолчанию — *master.info*.

Информация из файла master.info имеет приоритет перед информацией из файла *my.cnf*. То есть, если изменить значения в *my.cnf* и перезапустить сервер, соответствующие настройки будут все равно браться из файла *master.info*, а не из *my.cnf*.

Поэтому не рекомендуется заносить в файл *my.cnf* те настройки, которые можно установить командой CHANGE MASTER TO, а настраивать репликацию именно этой командой. Если же вы по какой-то причине предпочитаете поместить отдельные параметры репликации в файл *my.cnf*, то перед редактированием *my.cnf* выполните команду RESET SLAVE — это обеспечит считывание значений параметров при запуске подчиненного сервера.

Но будьте осторожны с командой RESET SLAVE! Она удаляет файлы *master.info*, *relay-log.info* и *все файлы журнала ретрансляции*.

Для удобства будем ссылаться на файлы информации, используя их стандартные имена.

Файл *master.info* содержит позицию в файле двоичного журнала главного сервера, а также информацию, необходимую для подключения к главному серверу и запуску репликации. Если файл *master.info* доступен, то поток ввода-вывода подчиненного сервера, стартуя, берет данные из него.

В лист. 6-1 приведен пример файла *master.info*. Перед каждой строкой стоит ее номер, а в конце строки — ее описание, выделенное курсивом (сам файл комментариев не содержит). Если сервер MySQL скомпилирован без поддержки протокола SSL, строки с 9 по 15, содержащие параметры SSL, будут опущены. В лист. 6-1 эти параметры приведены в предположении, что сервер поддерживает SSL. Поля SSL обсуждаются далее в этой главе.

В файле *master.info* пароль присутствует в незашифрованном виде. Поэтому крайне важно защитить этот файл, чтобы его мог читать только сервер MySQL. Обычно это делается так: создается специальная запись пользователя для запуска сервера, ему передаются все файлы, отвечающие за репликации и поддержку базы данных, а все разрешения на доступ к ним, помимо разрешений на чтение и запись именно этим пользователем, отзываются.

Лист. 6-1. Содержимое файла master.info (MySQL версии 5.1.16 с поддержкой SSL)

```
1  15                      Количество строк файла
2  master1-bin.000032      Текущий двоичный журнал, читаемый в настоящее
                           время (Master_Log_File)
3  475774                  Последняя прочитанная позиция двоичного журнала
                           (Read_Master_Log_Pos)
4  master1.example.com     Главный сервер, к которому производится
                           подключение (Master_Host)
5  repl_user               Имя пользователя репликации (Master_User)
6  xyzzy                   Пароль этой записи
7  3306                    Порт главного сервера, используемый
                           при репликации (Master_Port)
8  1                       Количество попыток подключения подчиненного
                           сервера к главному (Connect_Retry)
9  1                       1, если SSL поддерживается, в противном случае 0
10                         Центр сертификации SSL [SSL Certification
                           Authority (CA)]
11 /etc/ssl/certs          Путь к центру сертификации SSL
12 /etc/ssl/certs/slave.pem Сертификат SSL
13                         Шифрование SSL
14 /etc/ssl/private/slave.key Ключ SSL
15 0                       Проверка сертификата SSL (версия 5.1.16 и последующие)
```

В старых версиях сервера формат может быть несколько иным.

В версиях MySQL до 4.1 отсутствует первая строка. Разработчики включили счетчик строк файла в версию 4.11, — это позволило им добавлять в файл новые поля и, проверяя счетчик строк, определять, какие поля поддерживаются в той или иной версии сервера.

В версии 5.1.16 появилась последняя строчка — *проверка сертификата SSL (SSL Verify Server Certificate)*.

В файле *relay-log.info* отслеживается ход репликации, этот файл обновляется потоком SQL. В лист. 6-2 представлен примерный фрагмент файла *relay-log.info* (начало следующего события).

Лист. 6-2. Содержимое файла relay-log.info

```
./slave-relay-bin.000003  Файл журнала ретрансляции (Relay_Log_File)
380                       Позиция журнала ретрансляции (Relay_Log_Pos)
master1-bin.000001        Файл журнала главного сервера(Relay_Master_Log_File)
234                       Позиция журнала главного сервера(Exec_Master_Log_Pos)
```

Если любой из описанных выше файлов недоступен, он будет создан в *при запуске подчиненного сервера* на основе данных из файла my.cnf и параметров команды CHANGE MASTER TO.

Но для настройки репликации на подчиненном сервере недостаточно внести параметры в *my.cnf* и выполнить команду CHANGE MASTER TO. Файлы журнала ретрансляции, а также *master.info* и *relay-log.info* будут созданы лишь после выполнения команды START SLAVE.

Потоки репликации

Как было сказано выше, для репликации требуются несколько специализированных потоков, как на главном сервере, так и на подчиненном. Поток дампа на главном сервере поддерживает репликацию со стороны главного сервера. Еще два потока — поток ввода-вывода и поток SQL — обеспечивают выполнение репликации на подчиненном сервере.

- *Поток дампа главного сервера* Создается на главном сервере при подключении потока ввода-вывода подчиненного сервера. Поток дампа отвечает за чтение записей двоичного журнала на главном сервере и отправку их на подчиненный сервер. Для каждого подключенного подчиненного сервера создается один поток дампа.
- *Поток ввода-вывода подчиненного сервера* Подключается к главному серверу, чтобы запросить дамп всех произведенных изменений, затем заносит эти изменения в журнал ретрансляции, откуда их возьмет на обработку поток SQL. На каждом подчиненном сервере выполняется один поток ввода-вывода. После установки подключение остается открытым, чтобы все изменения, сделанные на главном сервере, немедленно передавались подчиненному.
- *Поток SQL подчиненного сервера* Считывает изменения из журнала ретрансляции и применяет их к базе данных подчиненного сервера. Поток SQL отвечает за координацию с другими потоками MySQL, — внесение в данные изменений не должно пересекаться с другими операциями сервера.

С точки зрения главного сервера поток ввода-вывода представляется всего лишь еще одним клиентским потоком, способным выполнять на главном сервере как запросы дампа, так и операторы SQL. Это означает, что клиент может подключиться к главному серверу, представившись (или, скажем,

прикинувшись) подчиненным сервером, и получить дамп изменений двоичного журнала. Именно так функционирует программа mysqlbinlog (подробно описанная в главе 3).

Поток SQL, работая с базой данных, действует как сеанс. Это означает, что поток SQL поддерживает ту же информацию о состоянии, что и сеанс, но с некоторыми отличиями. Поскольку поток SQL должен внести изменения, сделанные разными потоками главного сервера. События всех потоков главного сервера пишутся в двоичный журнал в порядке фиксации соответствующих транзакций, а поток SQL хранит дополнительную информацию, позволяющую различать события. К примеру, временные таблицы привязаны к сеансам, и чтобы различать временные таблицы разных сеансов, к событиям добавляется идентификатор сеанса. Поток SQL различает действия различных сеансов главного сервера идентификатору сеанса. Подробнее выполнение событий потоком SQL — ниже в этой главе.

Поток ввода-вывода заметно быстрее потока SQL, поскольку первый просто пишет события в журнал, а второй модифицирует БД. Поэтому во время репликации события обычно накапливаются в буфере (журнале ретрансляции). Если на главном сервере произойдет сбой, вам придется разобраться с буфером событий до подключения к новому главному серверу.

Чтобы не потерять накопленные события, перед подключением подчиненного сервера к новому главному следует подождать, пока SQL поток выберет все события, догнав тем самым поток ввода-вывода.

Существует несколько способов определить, пуст ли журнал ретрансляции, их мы рассмотрим ниже.

Запуск и остановка потоков подчиненного сервера

В главе 2 было показано, как запустить подчиненный сервер командой START SLAVE, но многие нюансы были опущены. Теперь вы готовы воспринять более детальное описание запуска и остановки потоков подчиненного сервера.

При наличии файла *master.info* подчиненный сервер, стартуя, запускает свои потоки. Как упоминалось ранее в этой главе, файл *master.info* создается, если были заданы параметры конфигурации и выполнена команда START SLAVE, запускающая потоки подчиненного сервера. Поэтому, если предыдущий сеанс выполнял репликацию, репликация возобновится с последней позиции, сохраненной в файлах *master.info* и *relay-log.info*, при несколько рознящемся поведении двух потоков подчиненного сервера.

- *Поток ввода-вывода подчиненного сервера* Поток ввода-вывода подчиненного сервера возобновит работу, начав с чтения последней прочитанной позиции, указанной в файле *master.info*.

  Для записи событий поток ввода-вывода произведет ротацию файлов журнала ретрансляции и начнет запись в новый файл, исправляя позиции соответствующим образом.

- *Поток SQL подчиненного сервера* Поток SQL подчиненного сервера возобновит свою работу, начав с чтения позиции журнала ретрансляции, указанной в файле *relay-log.info*.

Потоки подчиненного сервера можно явно запускать (командой START SLAVE) и останавливать (командой STOP SLAVE). Эти команды управляют потоками подчиненного сервера, ими можно запускать и останавливать потоки ввода-вывода и SQL независимо друг от друга.

- START SLAVE и STOP SLAVE В этой форме команды запускают и останавливают оба потока подчиненного сервера.
- START SLAVE IO_THREAD и STOP SLAVE IO_THREAD В этой форме команды запускают и останавливают только поток ввода-вывода.
- START SLAVE SQL_THREAD и STOP SLAVE SQL_THREAD В этой форме команды запускают и останавливают только поток SQL.

При остановке потоков подчиненного сервера текущее состояние репликации сохраняется в файлах *master.info* и *relay-log.info*. Эта информация будет использована при последующем запуске потоков.

Если вы зададите хост главного сервера (имя узла или IP-адрес) в параметре master-host (параметр можно поместить в файл *my.cnf* или же передать при запуске программы mysqld), подчиненный сервер также будет запущен.

Поскольку вместо этого параметра рекомендуется использовать параметр MASTER_HOST команды CHANGE MASTER, параметр master-host в нашей книге не рассматривается.

Репликация через Интернет

Существует множество поводов для организации репликации между географически удаленными центрами обработки данных (ЦОД). Во-первых, это позволит восстановить систему в случае крупной катастрофы, такой как землетрясение или отключение электроэнергии. Во-вторых, так можно приблизить сайт к пользователям, как делается в сетях доставки контента [Content Delivery Network, (CDN)], чтобы уменьшить время отклика. Хотя достаточно крупные организации могут позволить себе выделенные оптоволоконные линии, предположим, что вы решили воспользоваться обычным публичным интернет-каналом.

Передача событий от главного сервера к подчиненному не может считаться безопасной. В сущности, расшифровать передаваемые данные и получить реплицируемую информацию совсем несложно. Если ваша сеть защищена брандмауэром и не ведет репликацию (к примеру, между двумя ЦОД) через Интернет, дает некоторую защиту. Если же вы решите реплицировать данные в ЦОД, расположенный в другом городе или на другом континенте, крайне важно зашифровать информацию, чтобы скрыть ее от посторонних.

Стандартным способом шифрования данных для передачи по Интернету является использование протокола SSL. Имеются несколько вариантов защиты данных, и во всех участвует SSL:

- Чтобы шифровать репликацию от главного сервера к подчиненному, применяйте поддержку SSL, встроенную непосредственно в сервер.
- Если ваше программное обеспечение не поддерживает SSL, обратитесь к программе Stunnel, устанавливающей туннель SSL, по существу — виртуальную частную сеть [Virtual Private Network (VPN)].
- Используйте туннельный режим протокола ssh [Secure Shell (безопасная оболочка)].

Последний вариант не дает существенных преимуществ по сравнению с применением Stunnel, но окажется полезным, если установка новых программ запрещена и можно включить на серверах протокол ssh, который и установит туннель. В дальнейшем мы не будет обсуждать эту альтернативу.

Выберете ли вы для создания защищенного подключения встроенную поддержку SSL или stunnel, вам понадобится:

- Сертификат от центра сертификации (CA).
- Сертификат открытого ключа для сервера.
- Закрытый ключ для сервера.

В задачи этой книги не входит детальное обсуждение генерации, управления и применения сертификатов SSL, но в качестве иллюстрации мы приводим лист. 6-3, демонстрирующий генерацию самоподписанного сертификата открытого ключа и связанного с ним закрытого ключа. Подразумевается, что используется OpenSSL и соответствующий файл конфигурации */etc/ssl/openssl.cnf*.

Лист. 6-3. Генерация самоподписанного сертификата открытого ключа и закрытого ключа

```
$ sudo openssl req -new -x509 -days 365 -nodes -config /etc/ssl/openssl.cnf \
>   -out /etc/ssl/certs/master.pem -keyout /etc/ssl/private/master.key
Генерация 1024-битового закрытого ключа RSA
.....++++++
.++++++
writing new private key to '/etc/ssl/private/master.key'
Запись нового закрытого ключа в файл '/etc/ssl/private/master.key'
-----
Сейчас вы введете информацию, которая будет встроена в ваш запрос сертификата.
Следует ввести DN-имя (Distinguished Name)
Имеется еще несколько полей, но вы можете их не заполнять
Для некоторых полей будет установлено значение по умолчанию
Если вы введете '.', поле будет оставлено пустым
-----
Страна (Country Name) (Двухбуквенный код) [AU]:SE
Штат или область [Штат]:Uppland
Населенный пункт (к примеру, город) []:Storvreta
Организация [Internet Widgits Pty Ltd]:Big Inc.
Подразделение []:Database Management
Общее имя []:master-1.example.com
E-mail адрес []:mats@example.com
```

Самоподписанный сертификат помещается в файл */etc/ssl/certs/master.pem*, а закрытый ключ — в файл */etc/ssl/private/master.key* (он используется и для подписания сертификата открытого ключа).

Необходимо аналогичным образом создать ключ сервера и сертификат сервера на подчиненном сервере. Для удобства будем считать, что имя сертификата открытого ключа подчиненного сервера — */etc/ssl/certs/slave.pem*, а имя его закрытого ключа— */etc/ssl/private/slave.key*.

Настройка защищенной репликации с использованием встроенной поддержки SSL

Проще всего зашифровать подключение между главным и подчиненным сервером, если сервер MySQL поддерживает SSL. Обсуждение способов компиляции сервера с поддержкой SSL не входит в эту книгу, обращайтесь к руководствам в Интернете.

Чтобы воспользоваться встроенной поддержкой SSL, необходимо:

- Настроить главный сервер, сделав доступными ключи главного сервера.
- Настроить подчиненный сервер для шифрования канала репликации.

Чтобы настроить главный сервер на работу с SSL, добавьте в файл *my.cnf* следующие параметры:

```
[mysqld]
ssl-capath=/etc/ssl/certs
ssl-cert=/etc/ssl/certs/master.pem
ssl-key=/etc/ssl/private/master.key
```

Параметр ssl-capath содержит имя каталога, хранящего сертификаты доверенных центров сертификации, параметр ssl-cert — имя файла с сертификатом сервера, а параметр ssl_key — имя файла с закрытым ключом сервера. Как обычно, после внесения изменений в файл *my.cnf* следует перезапустить сервер.

Теперь главный сервер настроен на предоставление поддержки SSL любому клиенту, а поскольку подчиненный сервер использует обычный клиентский протокол, он также сможет применять SSL.

Чтобы настроить подчиненный сервер на работу с подключением по протоколу SSL, выполните команду CHANGE MASTER TO, передав ей параметр MASTER_SSL (со значением 1) — так вы инициируете SSL для подключения, — а затем параметры MASTER_SSL_CAPATH, MASTER_SSL_CERT и MASTER_SSL_KEY (они выполняют те же функции, что и параметры файла конфигурации ssl-capath, ssl-cert и ssl-key, упомянутые выше, но на стороне подчиненного сервера).

```
slave> CHANGE MASTER TO
    ->  MASTER_HOST = 'master-1',
    ->  MASTER_USER = 'repl_user',
    ->  MASTER_PASSWORD = 'xyzzy',
```

```
    ->  MASTER_SSL_CAPATH = '/etc/ssl/certs',
    ->  MASTER_SSL_CERT = '/etc/ssl/certs/slave.pem',
    ->  MASTER_SSL_KEY = '/etc/ssl/private/slave.key';
Query OK, 0 rows affected (0.00 sec)

slave> START SLAVE;
Query OK, 0 rows affected (0.15 sec)
```

Теперь подчиненный сервер соединен с главным защищенным каналом.

Настройка защищенной репликации при помощи Stunnel

Stunnel — простое в работе приложение для организации туннельного режима SSL. Эта программа настраивается и как сервер SSL, и как клиент SSL.

Применение Stunnel для организации безопасной репликации не намного сложнее, чем создание защищенного SSL подключения при наличии встроенной поддержки этого протокола, но требует некоторой дополнительной настройки. Этот подход полезен, если сервер скомпилирован без поддержки SSL или же, по каким-то причинам, вы хотите разгрузить сервер MySQL, освободив его от дополнительных операций шифрования и расшифровки данных (что имеет смысл только при использовании многоядерных процессоров).

Как и в случае встроенной поддержки SSL, необходимо иметь сертификат от центра сертификации (ЦС), а также сертификат открытого ключа и закрытый ключ для каждого сервера. Но использоваться они будут не для сервера, а для stunnel.

На рис. 6-3 показаны главный сервер, подчиненный сервер и два экземпляра программы Stunnel, обменивающиеся данными по незащищенной сети. Экземпляр Stunnel со стороны подчиненного сервера принимает данные от подчиненного сервера по стандартному клиентскому подключению MySQL, шифрует их и передает экземпляру Stunnel, работающему на стороне главного сервера. Этот экземпляр Stunnel, в свою очередь, прослушивает выделенный порт SSL для получения шифрованных данных, дешифрует их и посылает по клиентскому подключению в порт главного сервера, не поддерживающий SSL.

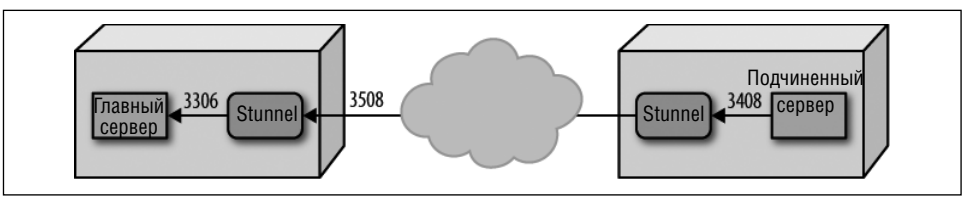


Рис. 6-3. Репликация по незащищенному каналу при участии Stunnel

В соответствии с настройками файла конфигурации в лист. 6-4, Stunnel прослушивает сокет SSL с номером 3508, в то время как главный сервер про-

слушивает сокет MySQL по умолчанию — 3306. В примере используются предложенные нами ранее имена для файлов сертификата и ключа.

Лист. 6-4. Файл конфигурации Stunnel на стороне главного сервера /etc/stunnel/master.conf

```
cert=/etc/ssl/certs/master.pem
key=/etc/ssl/private/master.key
CApath=/etc/ssl/certs
[mysqlrepl]
accept = 3508
connect = 3306
```

Лист. 6-5 представляет собой файл конфигурации для Stunnel на стороне клиента. Здесь в качестве промежуточного порта, не поддерживающего SSL, используется порт с номером 3408, — подчиненный сервер подключается через него локально. Также видно, что Stunnel будет подключаться к порту главного сервера SSL с номером 3508, как мы уже видели в лист. 6-4.

Лист. 6-5. Файл конфигурации Stunnel на стороне подчиненного сервера /etc/stunnel/slave.conf

```
cert=/etc/ssl/certs/slave.pem
key=/etc/ssl/private/slave.key
CApath=/etc/ssl/certs
[mysqlrepl]
accept = 3408
connect = master-1:3508
```

Теперь можно запустить оба экземпляра программы Stunnel и настроить подчиненный сервер так, чтобы он подключался к «своему» экземпляру. Поскольку этот экземпляр Stunnel выполняется на том же сервере, что и подчиненный сервер MySQL, параметру, определяющему хост главного сервера, следует присваивать значение localhost, а в качестве соответствующего порта указывать порт, прослушиваемый Stunnel (3408). Тогда Stunnel обеспечит туннелирование подключения до главного сервера.

```
slave> CHANGE MASTER TO
    ->  MASTER_HOST = 'localhost',
    ->  MASTER_PORT = 3408,
    ->  MASTER_USER = 'repl_user',
    ->  MASTER_PASSWORD = 'xyzzy';
Query OK, 0 rows affected (0.00 sec)

slave> START SLAVE;
Query OK, 0 rows affected (0.15 sec)
```

Теперь мы имеем защищенное подключение, установленное поверх незащищенной сети.

Если вы используете операционную систему Linux на основе Debian (например, Debian или Ubuntu), то можете для каждого файла конфигурации в каталоге */etc/stunnel* запустить по экземпляру Stunnel, указав параметр ENABLED=1 в файле */etc/default/stunnel4*.

Таким образом. чтобы при запуске машины автоматически запускались один экземпляр Stunnel на стороне подчиненного сервера и один экземпляр Stunnel на стороне главного сервера, достаточно создать файлы конфигурации Stunnel по рецептам из этого раздела.

Тонкая настройка репликации

Разобравшись в тонкостях механизмов репликации и данных, необходимых для репликации, вы сможете управлять репликацией как эксперт и избежите многих проблем. В этом вам поможет текущий раздел.

Информация о состоянии репликации

Большая часть информации о состоянии репликации находится на подчиненном сервере, хотя определенная ее часть присутствует и на главном. В основном информация главного сервера относится к двоичному журналу (см. главу 3), но доступны также сведения о подключенных подчиненных серверах.

Команда SHOW SLAVE HOSTS отображает информацию только о тех подчиненных серверах, в файлах конфигурации которых присутствует параметр report-host, — в этом параметре главному серверу передается информация (имя хоста или IP-адрес) о подключаемом подчиненном сервере. Поскольку между главным и подчиненным сервером могут присутствовать маршрутизаторы, использующие NAT, главный сервер не может доверять такой информации о подключенных серверах. В дополнение к имени хоста предлагаются еще несколько параметров, содержащих данные о подчиненном сервере.

- *report-host* Имя подключенного подчиненного сервера. Обычно используется доменное имя подчиненного сервера или какой-либо его аналог, но в действительности здесь можно указать любую строку. В лист. 6-6. мы выбрали имя «Magic Slave».
- *report-port* Порт, который прослушивает подчиненный сервер, подключившись к главному. По умолчанию — 3306.
- *report-user* Учетная запись (пользователь) для подключения подчиненного сервера к главному. Может отличаться от значения, переданного команде CHANGE MASTER TO. Будет отображаться, только если включен параметр show-slave-auth-info.
- *report-password* Пароль для подключения подчиненного сервера к главному. Может отличаться от значения, переданного команде CHANGE MASTER TO.

- *show-slave-auth-info* Если параметр включен, главный сервер допишет имя и пароль заданного пользователя в результат команды SHOW SLAVE HOSTS.

В лист. 6-6 приводится пример выходной информации команды SHOW SLAVE HOSTS — к главному серверу подключены три подчиненных.

Лист. 6-6. Пример вывода команды SHOW SLAVE HOSTS

```
master> SHOW SLAVE HOSTS;
+-----------+-------------+------+------------------+-----------+
| Server_id | Host        | Port | Rpl_recovery_rank | Master_id |
+-----------+-------------+------+------------------+-----------+
|         2 | slave-1     | 3306 |                0 |         1 |
|         3 | slave-2     | 3306 |                0 |         1 |
|         4 | Magic Slave | 3306 |                0 |         1 |
+-----------+-------------+------+------------------+-----------+
1 row in set (0.00 sec)
```

Команда перечислила подчиненные серверы, подключенные к главному, и добавила некоторую информацию о них. Обратите внимание — показаны и те подчиненные серверы, которые подключены к главному не напрямую, а через ретрансляторы. Если параметр show-slave-auth-info включен, добавляются еще два поля (здесь они опущены).

Выводимые значения — справочные, они могут и не отображать настоящее имя хоста, порт подчиненного сервера, учетную запись и пароль, заданные в момент настройки подчиненного сервера командой CHANGE MASTER TO:

- *Server_id* Идентификатор подключенного подчиненного сервера.
- *Host* Имя хоста, заданное параметром report-host.
- *User* Имя пользователя (учетная запись), заданная параметром report-user для отображения в отчетах о подчиненном сервере.
- *Password* Пароль, заданный параметром report-password для отображения в отчетах о подчиненном сервере.
- *Port* Порт.
- *Master_id* Идентификатор сервера, с которого ведется репликация.
- *Rpl_recovery_rank* Этот столбец никогда не использовался и был удален в MySQL версии 5.5.

Нельзя полностью доверять информации о подчиненных серверах, подключенных не напрямую, поскольку существует вероятность подмены истинных данных. Поэтому разработчики хотят удалить эту информацию из выходных данных, оставив только подключенные напрямую подчиненные серверы, сведения о которых гарантированны достоверны.

Чтобы получить список всех журналов главного сервера, составляющих двоичный журнал, достаточно выполнить команду SHOW MASTER LOGS. Примерный результат выполнения этой команды представлен в лист. 6-7.

Лист. 6-7. Типичный вывод команды SHOW MASTER LOGS

```
master> SHOW MASTER LOGS;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| master-bin.000011 |    469768 |
| master-bin.000012 |   1254768 |
| master-bin.000013 |    474768 |
| master-bin.000014 |      4768 |
+------------------+-----------+
1 row in set (0.00 sec)
```

Команда SHOW MASTER STATUS (см. лист. 6-8) показывает позицию двоичного журнала, в которой будет записываться следующее событие. Поскольку главный сервер имеет единственный двоичный журнал, таблица состоит из одной строки, а значения в этой строке соответствует значению последней строки вывода команды SHOW MASTER LOGS, но с другими заголовками. Поэтому если вы вызвали SHOW MASTER LOGS, то SHOW MASTER STATUS уже не требуется — достаточно взять последнюю строку результата SHOW MASTER LOGS.

Лист. 6-8. Типичный вывод команды SHOW MASTER STATUS

```
master> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB  | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| master-bin.000014 |     4768 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Для определения состояния потоков подчиненного сервера применяется команда `SHOW SLAVE STATUS`. Выходные данные этой команды включают практически всю нужную информацию о состоянии репликации. Обсудим вывод этой команды на примере (лист. 6-9).

Лист. 6-9. Типичный пример вывода команды SHOW SLAVE STATUS

```
               Slave_IO_State: Waiting for master to send event
                  Master_Host: master1.example.com
                  Master_User: repl_user
                  Master_Port: 3306
                Connect_Retry: 1
              Master_Log_File: master-bin.000001
          Read_Master_Log_Pos: 192
               Relay_Log_File: slave-relay-bin.000006
                Relay_Log_Pos: 252
        Relay_Master_Log_File: master-bin.000001
             Slave_IO_Running: Yes
```

```
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 192
              Relay_Log_Space: 553
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
            M  aster_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
```

Состояние потока ввода-вывода и потока SQL

Два поля — Slave_IO_Running и Slave_SQL_Running — показывают, выполняются ли потоки ввода-вывода и SQL, соответственно. Потоки подчиненного сервера не выполняются, если они остановлены или в ходе репликации произошла ошибка.

Если поток ввода-вывода не выполняется, поля Last_IO_Errno и Last_IO_Error помогут определить причину его остановки. Аналогично, в полях Last_SQL_Errno и Last_SQL_Error содержится информация о причинах остановки потока SQL. Если какой-либо из потоков остановился не из-за ошибки, а был остановлен явно или по условию (until), сообщения об ошибке отсутствует, а поле Errno равно 0, как в лист. 6-9. Поля Last_Errno и Last_Error — псевдонимы полей Last_SQL_Errno и Last_SQL_Error, соответственно.

Поле Slave_IO_State содержит описание текущего состояния потока ввода-вывода. На рис. 6-4 приведена диаграмма состояний, показывающая динамику состояния (и изменение соответствующего сообщения) потока ввода-вывода.

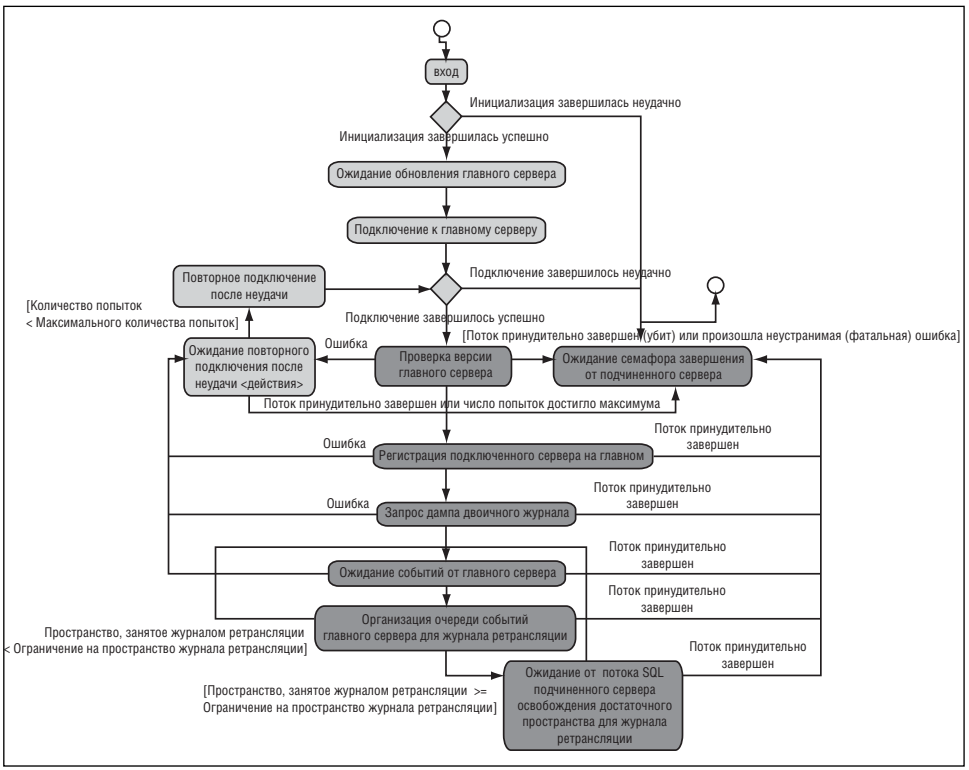


Рис. 6-4. Состояния потока ввода-вывода подчиненного сервера

Ниже перечислены сообщения и их значения:

- *Waiting for master update (Ожидание обновления главного сервера)* Выводится на короткий период, когда поток ввода-вывода уже инициализирован, но еще не начал подключаться к главному серверу.
- *Connecting to master (Подключение к главному серверу)* Выводится на том этапе, когда подчиненный сервер пытается подключиться к главному серверу, но подключение еще не установлено.
- *Checking master version (Проверка версии главного сервера)* Выводится тогда, когда подчиненный сервер подключился к главному и выполняет с ним процедуру «рукопожатия» (handshake).
- *Registering slave on master (Регистрация подчиненного сервера на главном)* Выводится, когда подчиненный сервер пытается зарегистрировать себя на главном сервере. При регистрации подчиненный сервер посылает главному серверу значение описанного выше параметра report-host. Значением этого параметра обычно является имя хоста или IP-адрес подчиненного сервера, но разрешена и произвольная строка. Главный сервер не может довериться простой проверке IP-адреса подключения по протоколу TCP, поскольку между подчиненным и главным сервером иногда попадаются маршрутизаторы с NAT.

- *Requesting binlog dump (Запрос дампа двоичного журнала)* Выводится, когда подчиненный сервер для получения от главного сервера дампа двоичного журнала отправляет ему имя файла и позицию двоичного журнала, а также свой идентификатор.
- *Waiting for master to send event (Ожидание событий от главного сервера)* Выводится, когда подчиненный сервер установил подключение к главному серверу и ожидает отправки главным сервером события.
- *Queueing master event to the relay log (Организация очереди событий главного сервера для журнала ретрансляции)* Выводится тогда, когда главный сервер послал событие и поток ввода-вывода подчиненного сервера собирается занести его в журнал ретрансляции. Сообщение отображается независимо от того, пишется ли событие на самом деле в журнал ретрансляции, или же отбрасывается в соответствии с правилами фильтрации (см. главу 5).

Обратите внимание на написание в оригинале слова «Queueing», а не «Queuing». Это важно, когда приходится искать отдельные сообщения при помощи сценариев или другого инструментария.

- *Waiting to reconnect after **action** (Ожидание повторного подключения после **действия**)* Выводится тогда, когда предыдущее действие завершилось со временной ошибкой и подчиненный сервер собирается подключаться повторно. ***Действие*** может быть одним из следующих:
  - registration on master (регистрация на главном сервере) Попытка зарегистрироваться на главном сервере;
  - binlog dump request (запрос дампа двоичного журнала) Запрос дампа двоичного журнала от главного сервера;
  - master event read (чтение события главного сервера) Ожидание или чтение события главного сервера.
- *Reconnecting after **failed action** (Повторное подключение после неудачи **действия**)* Выводится в тот момент, когда подчиненный сервер после неудачно завершенного действия пытается повторно подключиться к главному серверу, но подключение еще не установлено. Список действий тот же, что и для сообщения «Waiting to reconnect after ***action***» («Ожидание повторного подключения после ***действия***»)
- *Waiting for slave mutex on exit (Ожидание семафора завершения от подчиненного сервера)* Выводится на этапе остановки потока ввода-вывода.
- *Waiting for the slave SQL thread to free enough relay log space (Ожидание от потока SQL подчиненного сервера освобождения достаточного места для журнала ретрансляции)* Выводится, если ограничение на пространство для журнала ретрансляции (заданное параметром relay-log-space-limit) достигнуто, и потоку SQL требуется обработать журнал ретрансляции, чтобы занести в него новые событий.

Позиции двоичного журнала и журнала ретрансляции

Обрабатывая события на подчиненном сервере, механизм репликации отслеживает три позиции.

Каждой позиции соответствует пара полей в выходном результате команды SHOW SLAVE STATUS (см. лист. 6-9).

- *Master_Log_File, Read_Master_Log_Pos Позиция чтения от главного сервера* — позиция очередного события, предназначенного для чтения потоком ввода-вывода, в двоичном журнале главного сервера. Значения этих полей берутся из второй и третьей строки файла *master.info* (см. лист. 6-1).
- *Relay_Master_Log_File, Exec_Master_Log_Pos Позиция выполнения от главного сервера* — позиция очередного события, предназначенного для выполнения потоком SQL, в двоичном журнале главного сервера. Значения этих полей берутся из третьей и четвертой строки файла *relay-log.info* (см. лист. 6-2)
- *Relay_Log_File, Relay_Log_Pos Позиция выполнения из журнала ретрансляции* — позиция очередного события, предназначенного для выполнения потоком SQL, в журнале ретрансляции подчиненного сервера. Значения этих полей берутся из первой и второй строки файла *relay-log.info* (см. лист. 6-2)

Пользуясь информацией о позициях, вы можете следить за ходом репликации, а также оптимизировать некоторые алгоритмы, предложенные в главе 4.

Например, чтобы определить, имеются ли события, ожидающие выполнения, достаточно сравнить позиции чтения и позиции выполнения от главного сервера. Это особенно интересно наблюдать в том случае, если поток ввода-вывода остановлен, — так легче дождаться момента, когда журнал ретрансляции станет пуст. Как только позиции сравнялись, в журнале ретрансляции не осталось никаких событий в состоянии ожидания, и подчиненный сервер можно спокойно останавливать и переводить на другой главный сервер.

В лист. 6-10 приведен пример кода, отслеживающего момент опустошения журнала ретрансляции на подчиненном сервере. MySQL предлагает подходящую функцию MASTER_POS_WAIT, ожидающая обработки всех событий из журнала ретрансляции подчиненного сервера.

Лист. 6-10. Сценарий на языке Python для ожидания опустошения журнала ретрансляции

```
class SlaveNotRunning(Error):
   "Exception raised when slave is not running but were expected to run"
   pass

def slave_wait_for_empty_relay_log(slave):
   result = server.sql("SHOW SLAVE STATUS");
   file = result["Master_Log_File"]
   pos = result["Read_Master_Log_Pos"]
```

```
   if server.sql(_MASTER_POS_WAIT, (file, pos)) is None:
      raise SlaveNotRunning
```

Используя знания о позициях, вы может также оптимизировать сценарии из главы 4. Например, после выполнения кода, повышающего подчиненный сервер до главного (см. лист. 4-15), вам, вероятно, придется обработать множество событий из всех журналов ретрансляции оставшихся подчиненных серверов перед тем, как окончательно утвердить выбранный подчиненный сервер в роли главного. К тому же, если вы добьетесь, чтобы повышенный сервер обработал все события до подключения к нему остальных подчиненных серверов, то минимизируете потери данных.

Преобразуем лист. 4-15 в лист. 6-11 так, чтобы перед сменой ролей прежние подчиненные серверы выполнили все события, оставшиеся у них в журналах ретрансляции.

Лист. 6-11. Минимизация потерь при повышении подчиненного сервера

```
def fetch_global_trans_id(slave):
   result = slave.sql(«SELECT server_id, trans_id FROM Last_Exec_Trans»)
   return (int(result[«server_id»]), int(result[«trans_id»]))

def wait_for_empty_relay_log(slave):
result = slave.sql("SHOW SLAVE STATUS")
   slave.sql("SELECT MASTER_POS_WAIT(%s,%s)",
   (result["Master_Log_File"], result["Read_Master_Log_Pos"]))

def promote_slave(slaves):
   slave_info = {}

   # Соберем идентификаторы глобальных транзакций всех подчиненных серверов
   for slave in slaves:
      slave.connect()
      wait_for_empty_relay_log(slave)
      server_id, trans_id = fetch_global_trans_id(slave)
      slave_info.setdefault(trans_id, []).append((server_id, trans_id, slave))
      slave.disconnect()

   # В качестве подчиненного сервера – кандидата на повышение возьмем
   подчиненный сервер с максимальным идентификатором глобальной транзакции
   new_master = slave_info[max(slave_info)].pop()[2]

def maybe_change_master(server_id, trans_id, position):
   from mysqlrep.utility import change_master
   try:
      for sid, tid, slave in slave_info[trans_id]:
         if slave is not new_master:
            change_master(slave, new_master, position)
   except KeyError:
      pass
```

```
# Получаем файлы журнала нового главного сервера
new_master.connect()
logs = [row["Log_name"] for row in new_master.sql("SHOW MASTER LOGS")]
new_master.disconnect()

# Просматриваем файлы журнала главного сервера по одному в обратном порядке
– последний файл двоичного журнала берем первым
logs.reverse()
for log in logs:
   scan_logfile(new_master, log, maybe_change_master)
```

В дополнение к описанной здесь методике, в литературе упоминается и другой способ проверки состояния потока SQL — при помощи команды SHOW PROCESSLIST. Если в поле State указано «Has read all relay log; waiting for the slave I/O thread to update it" («Весь журнал репликации прочитан, поток ожидает обновления журнала репликации потоком ввода-вывода подчиненного сервера»), это означает, что поток SQL прочитал весь журнал репликации. Это значение поля State генерируется исключительно для потока SQL, поэтому искать этот текст можно в выходной информации по всем потокам.

Параметры для обработки разорванных подключений

Поток ввода-вывода отвечает за обслуживание подключений к главному серверу, и, как вы поняли из рис. 6-4, использует для этой цели довольно сложную логику.

Если поток ввода-вывода теряет подключение к главному серверу, то пытается подключиться повторно, имея при этом ограниченное число попыток подключения. Период бездействия, после которого поток ввода-вывода приступает к повторному подключению, период между попытками и количество попыток задаются тремя параметрами:

- *--slave-net-timeout* Количество секунд бездействия до того момента, когда подчиненный сервер решает, что подключение к главному серверу утрачено и приступает к повторному подключению. Этот параметр не влияет на ситуации, когда обрыв подключения определяется явно. В этих случаях подключенный сервер реагирует немедленно, переводя поток ввода-вывода в фазу повторного подключения, и попытается подключиться заново (возможно, после паузы, заданной параметром master-connect-retry, и с ограничением на число попыток из параметра master-retry-count). Значение по умолчанию — 3 600 секунд.
- *--master-connect-retry* Количество секунд между попытками переподключения. Задать эту настройку можно в параметре CONNECT_RETRY команды CHANGE MASTER TO. Соответствующий параметр файла

конфигурации *my.cnf* на данный момент не используется как устаревший. Значение по умолчанию — 60 секунд.

- *--master-retry-count* Максимальное количество попыток подключения. Значение по умолчанию — 86 400.

Возможно, значения по умолчанию вас не устроят, — задайте собственные значения этих параметров.

Как подчиненный сервер обрабатывает события

Ключевыми элементами репликации являются события из журналов. Они передают информацию о репликации по системе, а также содержат все метаданные, необходимые для того, чтобы изменения, внесенные на главном сервере, воспроизвелись бы и на репликах. Поскольку двоичный журнал главного сервера хранит все транзакции, выполненные на главном сервере, в порядке фиксации, транзакции должны выполняться на подчиненном сервере в том же порядке, в каком они занесены в двоичный журнал, — тогда результат их выполнения на подчиненном сервере будет идентичен результату их выполнения на главном сервере.

Поток SQL подчиненного сервера выполняет события всех сеансов главного сервера по порядку. Ниже приведены возможные последствия этого решения:

- *На подчиненном сервере события журнала обрабатываются одним потоком, а на главный сервер многопоточен* В результате подчиненный сервер может отстать от главного, если главный сервер фиксирует множество транзакций.
- *Некоторые операторы зависят от сеанса* Выполнение некоторых операторов на главном сервере зависит от сеанса, при их выполнении в сеансе подчиненного сервера результаты могут оказаться иными, чем в сеансе главного сервера. Специфичны для сеанса:
  - пользовательские переменные;
  - временные таблицы;
  - некоторые функции, к примеру, CONNECTION_ID.
- *Порядок выполнения определяется двоичным журналом* Не исключено, что две независимые на первый взгляд транзакции из двоичного журнала (транзакции, которые теоретически могли бы выполняться параллельно) на практике окажутся неким образом связанными. Следовательно, подчиненный сервер обязан выполнять транзакции строго последовательно, чтобы гарантировать согласованность данных главного и подчиненного сервера.

Роль потока ввода-вывода

Перед тем как события попадают к потоку SQL, несущему основную нагрузку по их обработке, их подготавливает поток ввода-вывода. Поэтому сначала мы рассмотрим обслуживание событий потоком ввода-вывода, а потом перейдем к «подлинному» выполнению кода потоком SQL. Чтобы не страдала производительность, поток ввода-вывода проверяет только определенные байты, чтобы выяснить тип события, а затем проделывает соответствующие манипуляции с журналом ретрансляции:

- *События останова (Stop events)* Указывают на то, что следующий подчиненный сервер в цепочке должен быть остановлен в обязательном порядке. Эти события игнорируются потоком ввода-вывода и даже не заносятся в журнал ретрансляции.
- *Событие ротации (Rotate event)* Если производится ротация двоичного журнала главного сервера, то производится ротация и журнала ретрансляции. Журнал ретрансляции можно ротировать чаще, чем двоичный журнал главного сервера, но ротация журнала ретрансляции непременно влечет за собой ротацию двоичного журнала главного сервера.
- *События описания формата (Format description events)* Запись этих событий откладывается на момент ротации журнала ретрансляции. Поскольку при переходе от одного файла двоичного журнала к другому формат может измениться, поток ввода-вывода должен помнить об этом событии, чтобы обработать файлы корректно.

Если используется круговая или двунаправленная (круговая схема из двух серверов) репликация, события двигаются по кругу, пока не попадут на сервер, который изначально их послал. Чтобы избежать бесконечного цикла репликации, необходимо удалять события по мере их исполнения.

Для этого каждый сервер проверяет идентификатор события. Если он совпадает с идентификатором проверяющего сервера, это означает, что данное событие отправлено именно этим сервером, и репликация совершила полный круг. Чтобы событие не ходило по кругу бесконечно (и не модифицировало к данным бесконечно), оно в этом случае не пишется в журнал ретрансляции, а просто игнорируется. Это можно отменить, задав параметр сервера replicate-same-server-id. Если этот параметр включен, сервер не сравнивает идентификатор сервера в событии со своим собственным идентификатором, и событие заносится в журнал ретрансляции независимо от значения своего идентификатора сервера.

Работа потока SQL

Поток SQL подчиненного сервера читает журнал ретрансляции и выполняет на подчиненном сервере операторы, ранее применявшиеся к базе данных главного сервера. Некоторым их этих событий требуется дополнительная информация, не содержащаяся в операторе SQL. Ниже приведен список подобных ситуаций:

- *Передача контекста главного сервера на подчиненный* Иногда, чтобы операторы выполнялись корректно, необходимо передать информацию о состоянии на подчиненный сервер. Как упоминалось в главе 3, главный сервер записывает одно или более событий контекста для передачи этой дополнительной информации.
- *Обработка событий из различных потоков* Поскольку главный сервер обрабатывает транзакции из разных сеансов, поток SQL подчиненного сервера должен определить, каким потоком сгенерированы отдельные события. Главный сервер обладает максимальной информацией об операторе, поэтому он помечает те события, которые считаются зависимыми от потока. Например, главный сервер обычно помечает как поточно-зависимые события, работающие с временными таблицами.
- *Фильтрация событий и таблиц* Поток SQL отвечает за фильтрацию на подчиненном сервере. MySQL поддерживает как фильтры уровня баз данных (задаваемые параметрами replicate-do-db и replicate-ignore-db), так и фильтры уровня таблиц (параметры replicate-do-table, replicate-ignore-table, replicate-wild-do-table и replicate-wild-ignore-table).
- *Пропуск событий* Чтобы возобновление остановленной репликации проходило нормально, часть событий при перезапуске репликации нужно пропускать. Эта задача решается потоком SQL.

События контекста

Некоторые события можно корректно обрабатываются только в исходном контексте. Под контекстом обычно понимают зависящую от потока среду, например, пользовательские переменные, а также информацию о состоянии, необходимую для правильного исполнения операторов: значения автоматического счетчика в полях с таким счетчиком. Для передачи подобного контекста с главного сервера на подчиненный в распоряжении главного сервера имеется набор событий контекста, которые можно заносить в двоичный журнал.

Главный сервер записывает событие контекста перед событием, несущим изменение данных. В настоящее время события контекста ассоциируются только с событиями запросов Query и добавляются в двоичный журнал перед ними.

События контекста делятся на следующие категории:

- *События пользовательских переменных* Событие содержит имя и значение пользовательской переменной.

  Событие генерируется всякий раз, когда оператор ссылается на пользовательскую переменную.

```
SET @foo = 'SmoothNoodleMaps';
INSERT INTO my_albums(artist, album) VALUES ('Devo', @foo);
```

- *События целочисленных переменных* Событие содержит целое значение либо для сеансовой переменной INSERT_ID, либо для сеансовой переменной LAST_INSERT_ID.

Событие целочисленной переменной INSERT_ID применяется для операторов вставки в таблицы, включающие столбец с функцией автоприращения AUTO_INCREMENT, — так передается следующее значение для этого столбца. Подобная операция потребуется, например, для приведенной ниже таблицы и оператора вставки в нее:

```
CREATE TABLE Artist (id INT AUTO_INCREMENT PRIMARY KEY, artist TEXT);
INSERT INTO Artist VALUES (DEFAULT, 'The The');
```

Событие целочисленной переменной LAST_INSERT_ID генерируется, если оператор использует функцию LAST_INSERT_ID, например:

```
INSERT INTO Album VALUES (LAST_INSERT_ID(), 'Mind Bomb');
```

- *События функции Rand (генератора псевдослучайных чисел)* Если в операторе вызывается функция `RAND`, событие содержит начальные значения, использованные этой функцией для генерации (random seeds), что позволяет подчиненному серверу воспроизвести «случайное» значение, порожденное на главном.

```
INSERT INTO my_table VALUES (RAND());
```

Мы перечислили те случаи, когда для корректной репликации необходимы события контекста, но существуют и ситуации, в которых события контекста не помогают. Например, репликация может работать с пользовательской функцией [user-defined function (UDF)] только в том случае, когда функция является детерминированной и присутствует на подчиненном сервере. В подобных случаях проблема решается при помощи событий пользовательских переменных.

События пользовательских переменных удобны, когда требуется избежать проблем с репликацией недетерминированных функций, также они позволяют улучшить производительность и помогают контролировать целостность данных.

Представьте, что вы вводите документы в таблицу базы данных. Каждому документу автоматически присваивается номер при помощи поля со свойством AUTO_INCREMENT. Чтобы обеспечить целостность документов, в ту же таблицу добавляется контрольная сумма документа, рассчитанная на основе 128-битного алгоритма хеширования MD5. В лист. 6-12 приведено определение такой таблицы:

Лист. 6-12. Определение таблицы документов с контрольной суммой на основе MD5

```
CREATE TABLE document(
   id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   doc BLOB,
   checksum CHAR(32)
);
```

Как показано в лист. 6-13, мы можем добавлять документы в таблицу, устанавливать их контрольные суммы, а также проверять целостность документов, чтобы убедиться, что они не изменены случайно либо преднамеренно. Хотя алгоритм MD5 на данный момент не считается криптографически надежным, он тем не менее дает защиту от случайных ошибок из-за повреждения данных на диске или в ОЗУ.

Лист. 6-13. Вставка в таблицу и проверка целостности документов

```
master> INSERT INTO document(doc) VALUES (document);
Query OK, 1 row affected (0.02 sec)

master> UPDATE document SET checksum = MD5(doc) WHERE id = LAST_INSERT_ID();
Query OK, 1 row affected (0.04 sec)

master> SELECT id,
    ->     IF(MD5(doc) = checksum, 'OK', 'CORRUPT!') AS Status
    ->  FROM document;
+----+----------+
| id |  Status  |
+----+----------+
|  1 | OK       |
|  2 | OK       |
|  3 | OK       |
|  4 | OK       |
|  5 | OK       |
|  6 | OK       |
|  7 | CORRUPT! |
|  8 | OK       |
|  9 | OK       |
| 10 | OK       |
| 11 | OK       |
+----+----------+
11 row in set (5.75 sec)
```

Но насколько совместима подобная технология с репликацией? Скажем так: это зависит от того, как вы ее применяете. Оператор INSERT (из лист. 6-13) при выполнении заносится в двоичный журнал в обычном виде, поэтому контрольная сумма MD5 на подчиненном сервере будет рассчитана заново. И что будет, если документ будет поврежден по пути на подчиненный сервер? Тогда контрольная сумма будет пересчитана уже для испорченного документа, и ошибку выявить не удастся. Значит, оператор из лист. 6-13 не безопасен для репликации. Но решение существует.

Заменим код из лист. 6-13 на код, представленный в лист. 6-14. Здесь контрольная сумма предварительно заносится в пользовательскую переменную, и лишь затем используется в операторе вставки INSERT. Поскольку пользовательская переменная содержит первоначальное значение, рассчитанной функцией MD5, контрольная сумма на подчиненном сервере совпадет

с контрольной суммой, рассчитанной на главном сервере, даже если документ при передаче будет испорчен (конечно, и сама контрольная сумма может исказиться при передаче данных). В любом случае повреждение документа не останется незамеченным.

Лист. 6-14. Ориентированный на репликацию способ вставки документа в таблицу

```
master> INSERT INTO document(doc) VALUES (document);
Query OK, 1 row affected (0.02 sec)

master> SELECT MD5(doc) INTO @checksum FROM document WHERE id = LAST_INSERT_ID();
Query OK, 0 rows affected (0.00 sec)

master> UPDATE document SET checksum = @checksum WHERE id = LAST_INSERT_ID();
Query OK, 1 row affected (0.04 sec)
```

События, зависимые от потока

Как уже говорилось ранее, некоторые операторы зависят от потока, — выполняясь в другом потоке, они дают иной результат. Подобные ситуации возникают при:

- *Чтении и записи объектов, локальных для потока* Объекты, локальные для потока, могут конфликтовать с объектами другого потока, имеющими те же имена. Типичными примерами таких объектов являются временные таблицы или пользовательские переменные. Мы уже знаем, как репликация обрабатывает пользовательские переменные, поэтому здесь сконцентрируемся на поддержке репликацией временных таблиц.
- *Применении переменных или функций, значение которых зависит от потока* Значение некоторых переменных и функций зависит от потока, в котором они выполняются. Типичный пример — серверная переменная connection_id.

В обработке сервером приведенных выше ситуаций присутствуют нюансы. К тому же иногда репликация и не старается учесть различия между сервером и клиентом, поэтому результаты слегка расходятся.

Обработка объектов, локальных для потока, требует присутствия разновидности локального хранилища потока [thread-local store (TLS)]. Поскольку репликация на подчиненном сервере однопоточна, он должен поддерживать локальные хранилища потоков и управлять ими. При работе с временными таблицами подчиненный сервер создает для таблицы уникальное (видоизмененное) имя из идентификатора серверного процесса, идентификатора потока и серийного номера потока. Это означает, что два оператора из лист. 6-15 могу вызываться разными клиентами главного сервера, и для представления их временных таблиц на подчиненном сервере будут сгенерированы разные имена.

Лист. 6-15. Два потока, каждый из которых создает временную таблицу

```
master-1> CREATE TEMPORARY TABLE cache (a INT, b INT);
Query OK, 0 rows affected (0.01 sec)

master-2> CREATE TEMPORARY TABLE cache (a INT, b INT);
Query OK, 0 rows affected (0.01 sec)
```

Поскольку в двоичный журнал все операторы из всех потоков главного сервера записываются друг за другом, необходимо различать операторы из лист. 6-15. В противном случае при выполнении на подчиненном сервере произойдет ошибка.

Чтобы избегать таких конфликтов, сервер помечает события запросов `Query`, содержащие подобные операторы, как зависимые от потока и добавляет к событию идентификатор потока. (На самом деле, идентификатор потока добавляется ко всем событиям Query, но в действительности это необходимо только для операторов, зависимых от потока.)

Когда подчиненный сервер получает событие, зависящее от потока, он присваивает специальной переменной потока репликации подчиненного сервера, так называемому *идентификатору псевдопотока*, значение идентификатора потока, переданное вместе с событием. Идентификатор псевдопотока в дальнейшем будет использован при создании временных таблиц. Идентификатор процесса подчиненного сервера — одинаковый для всех потоков главного сервера — также используется при построении имени таблицы, но это не имеет значения, если таблицы из различных потоков различаются.

Мы также говорили о том, что для правильного выполнения репликации необходим особый подход к функциям и переменным, зависящим от потока. Однако сервер такой подход не обеспечивает. Значение используемой в операторе серверной переменной будет получено на подчиненном сервере заново. Если по какой-то причине требуется реплицировать точное значение, вам придется сохранить значение в пользовательской переменной, как было показано в лист. 6-14, или использовать построчную репликацию, которую мы обсудим позже.

Фильтрация и пропуск событий

Иногда события пропускаются — они не проходят фильтры репликации или подчиненный сервер получает специальное указание о пропуске определенного количества событий.

Переменная SQL_SLAVE_SKIP_COUNTER указывает, сколько событий должен проигнорировать подчиненный сервер. Эту переменную можно задавать только тогда, когда поток SQL остановлен. Обычно это условие легко выполнимо, поскольку, скорее всего, эта переменная понадобится именно тогда, когда придется пропускать события, остановившие выполнение репликации.

Конечно, необходимо разобраться в причинах остановки репликации и устранить их, но если вы уже исправили ошибку вручную, следует проигнорировать событие, остановившее репликацию, и запустить репликацию снова, уже со следующих событий. Описываемая переменная предлагается для удобства, чтобы обойтись без команды CHANGE MASTER TO. В лист. 6-16 показано, как воспользоваться переменной после того, как сбойный оператор остановил репликацию.

Лист. 6-16. Применение переменной SQL_SLAVE_SKIP_COUNTER

```
slave> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 3;
Query OK, 0 rows affected (0.02 sec)

slave> START SLAVE;
Query OK, 0 rows affected (0.02 sec)
```

При запуске подчиненного сервера перед возобновлением репликации будут пропущены три события. Если таким образом подчиненный сервер окажется в середине транзакции, он проигнорирует и события, оставшиеся до конца транзакции.

Когда заданы фильтры репликации, подчиненный сервер фильтрует события. Как говорилось в главе 3, фильтрация может выполняться на главном сервере, но если заданы фильтры подчиненного сервера, то события, не подходящие условиям фильтра, отсеиваются потоком SQL, до того они, как и прочие, посылаются главным сервером и заносятся в журнал репликации.

Фильтрация зависит от уровня фильтров (фильтры уровня баз данных или уровня таблиц). В главе 3 обсуждалась логика, по которой оператор для определенной базы данных отсеивается из двоичного журнала. Такая же логика применима и к фильтрам подчиненного сервера, но здесь фильтр может быть наложен и на набор таблиц.

Важно отметить, что фильтр, наложенный на отдельную таблицу, изымает из репликации целый оператор, ссылающийся на нее. Логика фильтрации операторов на подчиненном сервере показана на рис. 6-5.

Логика фильтрации участвующих в операторах таблиц может показаться достаточно сложной. Во избежание нежелательных последствий придерживайтесь следующих правил:

- Не дополняйте имена таблиц именами баз данных, в которые они входят. Предваряйте оператор оператором USE, а не вводите новую базу данных по умолчанию.
- Не обновляйте таблицы из разных баз данных одним оператором.
- Избегайте обновления нескольких таблиц в одном операторе, если вы не уверены, что все эти таблицы отфильтровываются, или же не отфильтровывается ни одна из них. Обратите внимание: оператор будет отброшен целиком, если отбрасывается хотя бы одна из входящих в него таблиц (см. рис. 6-5).

Обеспечение надежности и восстановление подчиненного сервера

Подчиненные серверы также выходят из строя, и когда такое случается, приходится их восстанавливать. Для начала следует выяснить, отчего произошел сбой. Этот этап восстановления не поддается автоматизации, поскольку существует слишком много причин сбоев, которые трудно предвидеть. Подчиненному серверу может не хватить дискового пространства, или же он считал запорченную информацию о событии, не исключено также, что повторное выполнение некого оператора привело к ошибке из-за дублирования ключа. Однако некоторые процедуры восстановления и диагностики автоматизировать все же можно.

Синхронизация, транзакции и проблемы при сбоях баз данных

Чтобы обеспечить корректное возобновление репликации подчиненными серверами после сбоя на главном или на подчиненном сервере, вы должны:

- Убедиться, что на подчиненном сервере присутствуют все данные, необходимые для восстановления в случае сбоя.
- Выполнить восстановление подчиненного сервера.

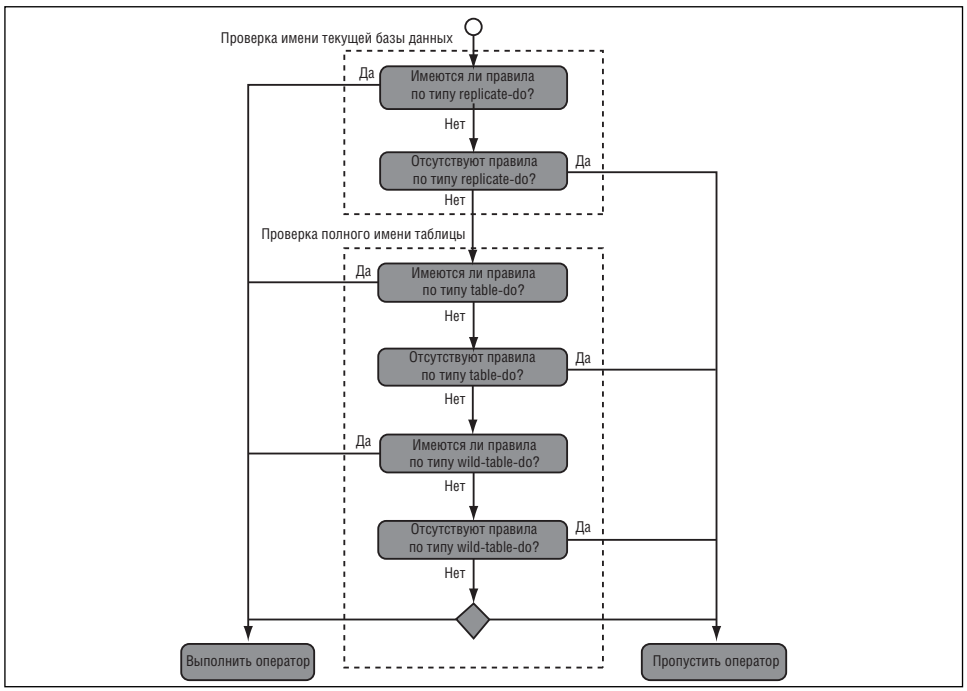


Рис. 6-5. Правила фильтрации при репликации

Подчиненные серверы синхронизируются с диском для максимального соответствия первому условию. Чтобы скорость выполнения была приемлемой, операционные системы работают с файлами, считанными в оперативную память, и сбрасывают данные на диск периодически или принудительно. Таким образом, запись данных в файл не всегда надежна. Если происходит сбой, данные в оперативной памяти, еще не перенесенные на диск, оказываются утраченными.

Чтобы заставить подчиненный сервер сбросить файлы на диск, сервер баз данных вызывает функцию fsync, которая переносит данные из оперативной памяти на диск. Чтобы защитить информацию репликации, сервер MySQL периодически вызывает fsync для журнала ретрансляции и файлов *master.info*, *relay-log.info*.

Синхронизация в потоке ввода-вывода

Если мы обратимся к потоку ввода-вывода, то увидим, что за обработкой любого события следуют два вызова fsync, — один, чтобы скинуть на диск журнал ретрансляции, и один, чтобы скинуть на диск файл *master.info*. Благодаря этому ни одно событие не пропадет, если подчиненный сервер «упадет» между сбросом журнала ретрансляции и сбросом файла *master.info*. Однако событие может дублироваться в следующей ситуации:

1. Сервер сбрасывает журнал ретрансляции и собирается приступить к обновлению позиции чтения от главного сервера в файле *master.info*.
2. Происходит сбой, и текущей становится позиция перед событием, сброшенным в журнал ретрансляции.
3. Сервер возобновляет выполнение, получает позицию на главном сервере из файла *master.info* и позиция перед последним событием пишется в журнал ретрансляции.
4. Репликация возобновляется с указанной позиции, и событие дублируется.

Если сбрасывать файлы на диск в обратном порядке, — сперва файл *master.info* и затем журнал ретрансляции, — аналогичный сценарий может привести к потере события, поскольку подчиненный сервер возобновит репликацию за событием, которое только должно было заноситься в журнал ретрансляции. Потеря события считается большей неприятностью, чем дублирование, поэтому на диск сначала переносится журнал ретрансляции.

Синхронизация в потоке SQL

Поток SQL обрабатывает события журнала ретрансляции группами, выполняя все события по очереди. Когда все события группы выполнены, поток SQL фиксирует транзакцию следующим образом:

1. Фиксирует транзакцию в механизме хранения (в предположении, что механизм хранения поддерживает фиксацию).

2. Обновляет файл *relay-log.info*, записав в него позицию следующего события, — этим событием откроется новая группа, предназначенная для обработки.
3. Сбрасывает файл *relay-log.info* при помощи вызова fsync(2).

При обработке событий из группы поток увеличивает позицию события согласно журналу ретрансляции. Если произойдет сбой, выполнение возобновится с последней позиции, занесенной в файл *relay-log.info*.

При таком алгоритме поток SQL сталкивается со своим вариантом проблемы атомарного обновления, о которой мы упоминали при обсуждении потока ввода-вывода, — база данных подчиненного сервера и файл *relay-log.info* могут рассинхронизироваться в следующей ситуации:

1. Событие применяется к базе данных и транзакция фиксируется. Теперь должен быть обновлен файл *relay-log.info*.
2. Подчиненный сервер выходит из строя, следовательно файл *relay-log.info* на данный момент указывает на начало только завершенной транзакции.
3. При восстановлении поток SQL считывает информацию из файла *relay-log.info* и запускает репликацию, начиная с сохраненной позиции.
4. Последняя выполненная транзакция оказывается повторенной.

Сказанное сводится к тому, что фиксация транзакции на подчиненном сервере плюс обновление информации о репликации нельзя считать атомарной операцией, — может оказаться, что файл *relay-log.info* отражает зафиксированную в базе данных ситуацию неадекватно.

Как еще следует из вышеизложенного, репликация выполняется в предположении об отказоустойчивости таблиц, даже не защищенных транзакциями в том смысле, что каждый оператор выполняется атомарно. То есть, либо оператор выполняется целиком, либо оператор не выполняется вообще. Чтобы гарантировать такую схему, механизм БД должен откатить оператор при возобновлении работы сервера, если сбой произошел в течение исполнения оператора.

Правила для защиты нетранзакционных операторов

Невозможно отследить и защитить от повторного выполнения после сбоя операторы, выполняемые вне транзакции. Эта проблема присуща как главным, так и подчиненным серверам. Оператор, изменяющий таблицу MyISAM и прерванный из-за сбоя на главном сервере, не попадает в журнал вообще, поскольку запись в двоичный журнал производится, когда оператор уже выполнен. После перезапуска (и успешного восстановления) таблица MyISAM окажется частично обновленой, но в двоичном журнале это оператор отражен не будет.

Похожим образом обстоят дела и на подчиненном сервере: если сбой происходит в процессе выполнения оператора (или транзакции, изменяющей

нетранзакционную таблицу), не исключено, что изменения останутся в таблице, но позиция группы не изменится. При возобновлении репликации подчиненным сервером нетранзакционный оператор выполнится повторно.

Проблемы, связанные со сбоями в момент обновления нетранзакционной таблицы, невозможно отлавливать автоматически, но существуют определенные правила, следование которым гарантирует в подобной ситуации уведомление об ошибке.

- *Операторы INSERT* При работе с этими операторами необходимо в каждой из реплицируемых таблиц иметь первичный ключ. В этом случае оператор INSERT, выполняемый повторно, сгенерирует ошибку из-за дублирования ключа, подчиненный сервер будет остановлен, а вы сможете определить причину несогласованности главного и подчиненного серверов.
- *Операторы DELETE* При работе с этими операторами необходимо избегать предложения LIMIT. В случае отсутствия предложения LIMIT при повторном выполнении оператор попытается удалить те же самые строки, что и в первый раз (строки, отвечающие предложению WHERE). Он либо удалит требуемые строки, если это не удалось в первый раз, либо не удалит ничего, если все подходящие строки уже удалены, — и то и то нас устраивает. Если же в операторе присутствует предложение LIMIT, удаляется только часть строк, соответствующих условию WHERE, и при повторном выполнении оператора будет удалено другое подмножество строк.
- *Операторы UPDATE* Это наиболее проблематичные операторы. Чтобы сбой не отражался на выполнении, оператору UPDATE следует быть *идемпотентным*, то есть результаты двукратного и однократного исполнения оператора должны совпадать, или же случающееся время от времени повторение оператора должно быть приемлемо для системы, к примеру, если этот оператор всего лишь обновляет статистику, скажем, обращения к странице.

Многоисточниковая репликация

Как вы уже могли заметить, невозможно подключить подчиненный сервер к нескольким главным серверам и получать обновления от всех них. Подобная топология называется *многоисточниковой* (*multisource*), и ее не следует путать с множественной (multi-master) топологией, представленной в главе 5. В многоисточниковой топологии изменения приходят от нескольких главных серверов, а во множественной — сервера образуют группу, которая функционирует как общий главный сервер, и изменения реплицируются между всеми серверами этой группы.

Планы организации поддержки многоисточниковой репликации в MySQL вынашиваются уже давно, но на пути реализации этого проекта стоит следующая проблема — как быть с конфликтующими обновлениями. Конфликтующие обновления могут поступить от различных источников,

также не исключено, что два промежуточных сервера-ретранслятора перенаправят изменение, выполненное общим главным сервером. На рис. 6-6 проиллюстрированы оба варианта конфликтов. В первом — два главных сервера (источника) изменяют одни и те же данные, и подчиненный сервер не знает, какое из изменений является окончательным. Во втором варианте делается единственное изменение, но для подчиненного сервера оно выглядит как два, от двух отдельных источников. В обоих случаях подчиненный сервер не может сделать выбор между событиями, поступившими с двух серверов-ретрансляторов. Таким образом, единичное событие, отправленное главным сервером, прибыв на подчиненный, выглядит как два различных события.

Ромбовидную (diamond) конфигурацию не обязательно настраивать явно, она может образоваться спонтанно при переключении ретрансляторов, если в момент переключения поток репликации окажется дублирован. Поэтому важно удостовериться, что все события из очереди — на подчиненном сервере и на всех ретрансляторах между главным и подчиненным сервером — были реплицированы на подчиненном сервере перед переводом его на другой главный.

Чтобы избежать конфликтов, следует убедиться, что переключения выполняются корректно, и (в случае нескольких источников данных) обновления выполнены так, что шансов для возникновения конфликта не осталось. Обычно для реализации такого подхода обновляются различные базы данных, также можно применять обновления к различным строкам одной и той же таблицы на различных серверах.

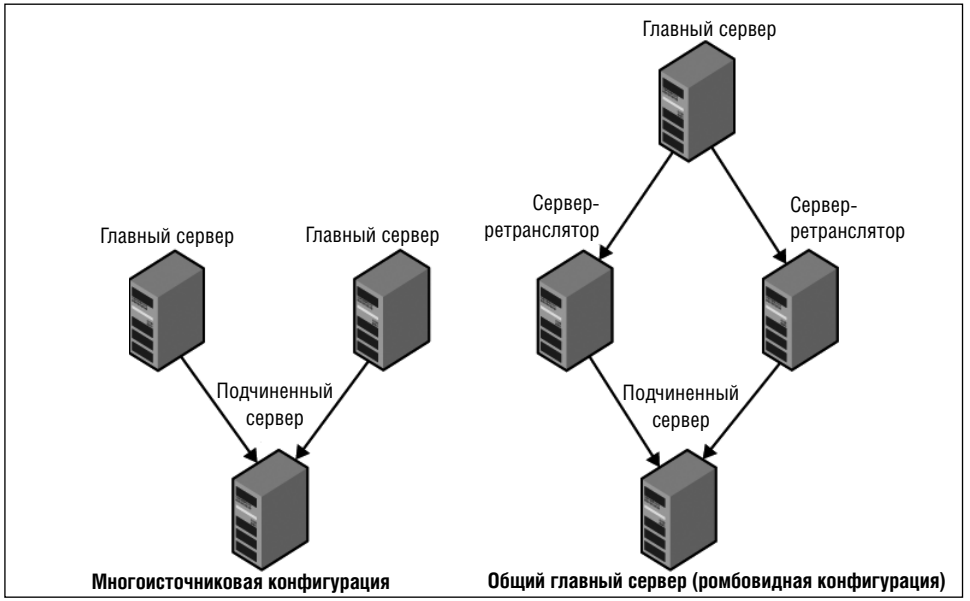


Рис. 6-6. Истинная многоисточниковая и ромбовидная конфигурации

Хотя в настоящее время MySQL не поддерживает одновременную репликацию из нескольких источников, вы можете реализовать близкую конфи-

гурацию, переключая подчиненный сервер между несколькими главными серверами, и реплицируя на подчиненном сервере изменения с нескольких главных по очереди. Такая репликация называется *циклической многоисточниковой репликацией (round-robin multisource replication)*, она хорошо подходит приложениям определенного типа, например, приложениям, которые агрегируют для отчета данные, поступающие из различных источников. В этих ситуациях данные можно разделить естественным образом, помещая записи от каждого из главных серверов в отдельную базу данных, таблицу или секцию. Это исключает риск конфликта и позволяет прибегнуть к многоисточниковой репликации.

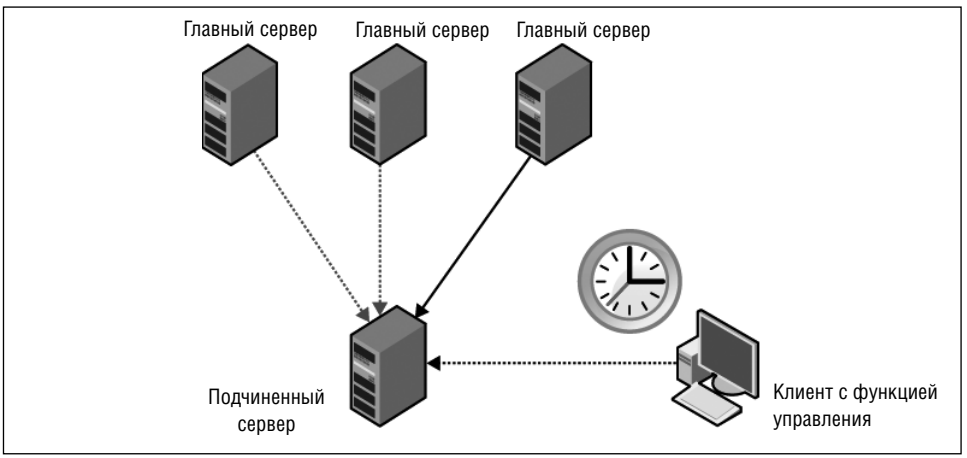


Рис. 6-7. Циклическая многоисточниковая репликация, использующая клиента для переключения между главными серверами

На рис. 6-7 показан подчиненный сервер, который реплицируется по циклическому типу от трех главных серверов. Репликацией управляет специальный клиент, контролирующий переключение между главными серверами. Для настройки циклической многоисточниковой репликации выполните следующие действия:

1. Настройте подчиненный сервер на репликацию от одного главного, который мы назовем *текущим главным сервером* (*current master*).
2. Запустите репликацию подчиненного сервера на фиксированный отрезок времени. Подчиненный сервер будет принимать изменения от текущего главного сервера и применять их, в том время как клиент, отвечающий за переключение серверов, находится в состоянии ожидания.
3. Остановите поток ввода-вывода подчиненного сервера командой STOP SLAVE IO_THREAD.
4. Дождитесь, пока журнал ретрансляции исчерпается.
5. Остановите поток SQL командой STOP SLAVE SQL_THREAD. Для выполнения команды CHANGE MASTER необходимо, чтобы оба потока подчиненного сервера были остановлены.

6. Запомните позицию подчиненного сервера в репликации от текущего главного сервера, сохранив значения столбцов Exec_Master_Log_Pos и Relay_Master_Log_File из вывода команды SHOW SLAVE STATUS.
7. Настройте подчиненный сервер на репликацию от следующего по порядку главного сервера, установив ранее сохраненную позицию командой настройки репликации CHANGE MASTER.
8. Перезапустите потоки подчиненного сервера командой START SLAVE.
9. Повторите шаги, начиная с шага 2.

Заметьте, что на этапах от 3 до 5 мы остановили сперва поток ввода-вывода и затем поток SQL. Действовать таким образом, вместо того чтобы сразу остановить репликацию на подчиненном сервере, необходимо потому, что поток SQL может отставать (и обычно отстает), поэтому остановив оба потока одновременно, мы потеряем часть необработанных событий из журнала ретрансляции — они просто будут отброшены. Если же вас больше волнует выполнение, допустим, некой незначительной группы транзакций от каждого главного сервера и потеря дополнительных событий не играет роли, не выполняйте шаги с 3 по 5, а остановите репликацию обычным образом. Процедура в целом будет работать корректно, поскольку отброшенные события придут с главного сервера в следующем цикле.

Конечно, имеет смысл автоматизировать описанную процедуру, воспользовавшись подключением клиента и библиотекой MySQL Replicant, как показано в лист. 6-17. Чтобы считывать по очереди главные сервера из списка, достаточно обратиться к функции cycle из модуля itertools.

Лист. 6-17. Реализация циклической многоисточниковой репликации на языке Python

```
import itertools

position = {}
def round_robin_multi_master(slave, masters):
   current = masters[0]
   for master in itertools.cycle(masters):
      slave.sql("STOP SLAVE IO_THREAD");
      mysqlrep.wait_for_empty_relay_log(slave)
      slave.sql("STOP SLAVE SQL_THREAD");
      position[current.name] = mysqlrep.fetch_slave_position(slave)
      slave.change_master(position[current.name])
      master.sql("START SLAVE")
      current = master
      sleep(60)   # Ожидание 1 минута
```

Построчная репликация

Основной целью репликации является синхронизации главного и подчиненного сервера — данных на них должны совпадать. Как вы уже видели, репликация снабжена дополнительным функционалом, позволяющим обе-

спечивать максимально возможное приближение к этой цели — события контекста, зависимые от сеанса идентификаторы и т. п.

Несмотря на это, в некоторых случаях на данный момент невозможно организовать корректную логическую репликацию:

- Как говорилось в этой главе выше, операторы UPDATE, DELETE или INSERT, содержащие предложение LIMIT, могут создать проблемы в случае сбоя базы данных, произошедшего в момент их исполнения.
- Если при выполнении нетранзакционного оператора происходит ошибка, результаты на главном и подчиненном серверах могут не совпасть.
- Если оператор обращается к пользовательской функции (UDF), невозможно обеспечить идентичность результата на подчиненном сервере.
- Если оператор включает любую недетерминированную функцию, — например, USER, CURRENT_USER, CONNECTION_ID, — опять-таки результаты, полученные на главном и на подчиненном серверах, могут разниться.
- Если оператор обновляет две таблицы, включающие столбцы со свойством автоувеличения (счетчики), он не отработает корректно, поскольку реплицировать можно только единственное значение последнего вставленного идентификатора, которое и будет использовано для обеих таблиц подчиненного сервера, в то время как на главном сервере идентификатор вставки вычисляется для каждой таблицы индивидуально.

В подобных случаях имеет смысл реплицировать сами данные, вставляемые в таблицы, — как это и делает построчная репликация.

Вместо репликации посредством операторов, выполняющих изменения, репликация на уровне строк реплицирует каждую вставляемую, удаляемую или обновляемую строку отдельно, с теми значениями, которые участвовали в операции. Поскольку строка, отправляемая на подчиненный сервер, — та же строка, что передана механизму БД, она содержит реальные данные, вставляемые в таблицу. Таким образом, не приходится ни разбираться с UDF, ни отслеживать столбцы-счетчики, ни принимать во внимание возможность частичного исполнения операторов, — только данные в чистом виде.

Построчная репликация позволяет реализовать совершенно новые варианты сценариев, невозможные при логической репликации. Однако необходимо учитывать нюансы поведения этих видов репликации.

Выбирая между логической и построчной репликацией, обдумайте следующие моменты:

- Имеются ли операторы, обновляющие множество строк, или основная часть операторов изменяет и вставляет ограниченное количество строк?
- Если оператор изменяет множество строк, логическая репликация, работающая с компактными операторами, может выполняться быстрее. Но поскольку оператор приходится выполнять и на сервере, это не всегда так. Если оператор требует сложной оптимизации и сложного плана выполнения, не исключено, что построчная репликация окажется быстрее.

- Если оператор изменяет или вставляет только несколько строк, построчная репликация потенциально быстрее, так как из процесса исключен синтаксический анализ и все выполняется непосредственно в механизме БД.
- Хотите ли вы видеть выполняющиеся операторы? События при построчной репликации туманны, если не сказать большего. В логической репликации операторы заносятся в двоичный журнал и прямо из него могут быть прочитаны.
- Логическая репликация проста — на подчиненном сервере выполняется тот же оператор, что и на главном. Эта концепция родилась довольно давно и знакома большинству администраторов баз данных. С другой стороны, построчная репликация относительно молода и может доставить неприятности в нештатной ситуации — при сбое.
- Если данные на главном и подчиненном серверах отличаются, то, вероятно, операторы при выполнении дадут различные результаты. Иногда это отвечает намерениям разработчиков, тогда можно и нужно использовать логическую репликацию, а иногда нет, тогда следует использовать построчную репликацию, исключающую подобную ситуацию.

Приемы для успешной работы с этими разновидностями репликации различаются. Некоторые моменты тонкой настройки логической репликации показаны в предыдущих главах, а здесь мы обсудим способы, позволяющие максимально эффективно применять построчную репликацию.

Настройка построчной репликации

Для настройки построчной репликации применяются следующие параметры:

- *binlog-format* Параметр binlog-format может принимать следующие значения:
- *STATEMENT* Для всех операторов будет использоваться традиционная логическая репликация.
- *ROW* Для операторов, вставляющих или изменяющих данные, то есть для операторов языка манипулирования данными [data manipulation language (DML)], применяется новейшая разновидность репликации — построчная репликация. Однако логическая репликации тоже работает, — она используется для операторов, создающих таблицы или иным образом изменяющих схему базы данных, то есть для операторов языка определения данных [data definition language (DDL)].
- *MIXED* Этот режим считается самым надежным для логической репликации. Начиная с MySQL версии 5.1, рекомендуется применять именно его. В смешанном режиме репликации сервер пишет операторы в двоичный журнал именно как операторы, но переключается в режим построчной репликации, если оператор ненадежен по одному из критериев, приведенных выше.

Режим репликации можно задать и посредством одноименной глобальной серверной переменной, и посредством сеансовой переменной. При запуске нового сеанса сеансовая переменная принимает глобальное значение, а затем используется для принятия решений о записи операторов в двоичный журнал.

- *binlog-max-row-event-size* Воспользуйтесь этим параметром, чтобы указать, когда следует начать новое событие, содержащее строки. Поскольку события полностью считываются в оперативную память при обработке, это позволяет с некоторой степенью точности контролировать размер событий со строками, чтобы не перерасходовать оперативную память при обработке строк.

Смешанный режим репликации

Хотя, начиная с MySQL версии 5.1, рекомендуется смешанный режим репликации, значением по умолчанию параметра binlog-format все еще остается STATEMENT. Это решение может показаться странным, но оно позволяет избежать проблем с обслуживанием серверов версии 5.0 или более ранней. В этих версиях построчной репликации еще нет, пользователи вынуждены применять логическую репликацию, поэтому переход к новому режиму репликации оказался бы достаточно резким, чего хотят избежать разработчики MySQL. Если сервера после обновления внезапно станут посылать события построчной репликации, начнется беспорядок. Чтобы администраторам, отвечающим за переход на новые версии, не прибавилось главной боли, значением по умолчанию для этого параметра остается STATEMENT.

Но если вы используете в работе один из шаблонов, поставляемых в дистрибутиве MySQL версии 5.1, то увидите, что параметр binlog-format имеет значение MIXED, в соответствии с рекомендацией.

Принцип, по которому работает репликация в смешанном режиме, несложен, — большую часть времени применяется логическая репликация, которая сменяется построчной репликацией только для небезопасных операторов. Мы уже анализировали типы операторов, репликация которых может повлечь проблемы. Здесь скажем кратко: если установлен смешанный режим, то переключение на построчную происходит, если:

- Оператор вызывает:
  - Функцию UUID, то есть функцию, генерирующую универсальный уникальный идентификатор [Universally Unique Identifier (UUID)].
  - Пользовательскую функцию.
  - Функцию CURRENT_USER или USER.
  - Функцию LOAD_FILE.
- Две или более таблицы, содержащие столбцы со свойством AUTO_INCREMENT (счетчики), обновляются в одном операторе.
- В операторе используется серверная переменная.

- Механизм БД не поддерживает логическую репликацию. Это справедливо, к примеру, для механизма MySQL Cluster.

Список этот по определению не может считаться исчерпывающим, он расширяется по мере обнаружения новых небезопасных конструкций. Более полный и точный список вы найдете в справочном руководстве MySQL по адресу http://dev.mysql.com/doc/refman/5.1/en/binary-log-mixed.html.

События построчной репликации

В логической репликации операторов каждый оператор в процессе обработки порождает единственное событие запроса Query. Однако, поскольку один оператор может изменить значительное количество строк, построчная репликация решает этот вопрос по-другому, используя несколько событий для одного оператора.

Построчная репликация вводит в обращение четыре новых события:

- *Событие Table_map* Событие сопоставления таблиц Table_map задает соответствие между идентификатором таблицы и ее именем (вместе с именем базы данных), а также некоторой базовой информацией о столбцах таблицы на главном сервере.

  Информация о таблице не включает имена столбцов, только типы. Дело в том, что построчная репликация ориентируется на позиции, — каждый столбец главного сервера реплицируется в столбец с той же позицией в таблице подчиненного сервера.
- *События Write_rows, Delete_rows и Update_rows* Эти события генерируются тогда, когда строки, соответственно, вставляются, удаляются или обновляются. Таким образом, единственный оператор может породить несколько событий.

  Помимо строк, каждое событие содержит идентификатор таблицы, ссылающийся на идентификатор таблицы из предшествующего события Table_map, и одну или две *битовых маски*, указывающие столбцы, на которое повлияло событие. Так удается сберечь место в журнале, поскольку в журнал пишутся только модифицированные столбцы или столбцы, необходимые для поиска вставляемых, удаляемых или обновляемых строк.

  В настоящее время только механизм БД MySQL Cluster способен ограничивать перечень столбцов, записываемых в журнал.

Каждый выполненный оператор представляется в двоичном журнале следующим образом — последовательность событий Table_map, за которой следует последовательность соответствующих событий строк. Последнее в операторе событие строки помечается специальным флажком.

В лист. 6-18. показано выполнение оператора и результирующие события. Для простоты мы опустили событие описания формата, поскольку вы уже видели его раньше.

Лист. 6-18. Выполнение оператора INSERT и результирующие события

```
master> BEGIN;
Query OK, 0 rows affected (0.00 sec)

master> INSERT INTO t1 VALUES (1),(2),(3),(4);
Query OK, 4 rows affected (0.01 sec)
Records: 4 Duplicates: 0 Warnings: 0

master> INSERT INTO t1 VALUES (5),(6),(7),(8);
Query OK, 4 rows affected (0.01 sec)
Records: 4 Duplicates: 0 Warnings: 0

master> COMMIT;
Query OK, 0 rows affected (0.00 sec)

master> SHOW BINLOG EVENTS IN 'master-bin.000053' FROM 106\G
*************************** 1. row ***************************
   Log_name: master-bin.000054
        Pos: 106
 Event_type:Query
  Server_id: 1
End_log_pos: 174
       Info: BEGIN
*************************** 2. row ***************************
   Log_name: master-bin.000054
        Pos: 174
 Event_type: Table_map
  Server_id: 1
End_log_pos: 215
       Info: table_id: 18 (test.t1)
*************************** 3. row ***************************
   Log_name: master-bin.000054
        Pos: 215
 Event_type: Write_rows
  Server_id: 1
End_log_pos: 264
       Info: table_id: 18 flags: STMT_END_F
*************************** 4. row ***************************
   Log_name: master-bin.000054
        Pos: 264
 Event_type: Table_map
  Server_id: 1
End_log_pos: 305
       Info: table_id: 18 (test.t1)
*************************** 5. row ***************************
   Log_name: master-bin.000054
        Pos: 305
```

```
 Event_type: Write_rows
  Server_id: 1
End_log_pos: 354
       Info: table_id: 18 flags: STMT_END_F
*************************** 6. row ***************************
   Log_name: master-bin.000054
        Pos: 354
 Event_type: Xid
  Server_id: 1
End_log_pos: 381
       Info: COMMIT /* xid=23 */
6 rows in set (0.00 sec)
```

Здесь показано, как два оператора добавляются в двоичный журнал. Запись об операторе начинается с события Table_map, за которым следует единственное событие Write_Rows, содержащее четыре строки, — и так для каждого из операторов.

Вы видите, что окончание каждого оператора отмечено в событии строки соответствующим флагом. Поскольку операторы заключены в транзакцию, они находятся между событиями Query с операторами BEGIN и COMMIT.

Размер событий строки регулируется параметром binlog-row-event-max-size, задающим предельное значение количества байтов в двоичном журнале. Параметр не устанавливает максимальный размер события строки, — не исключено, что событие строки в журнале транзакции будет иметь больший размер, если строка содержит больше байтов, чем задано параметром binlog-row-event-max-size.

События сопоставления таблиц Table_map

Как уже говорилось, событие Table_map сопоставляет имя таблицы идентификатору таким образом, чтобы использовать идентификатор в событиях строк, но роль события Table_map этим не исчерпывается. Также событие содержит некоторую базовую информацию о полях идентифицируемой им таблицы главного сервера. Пользуясь этой информацией, подчиненный сервер проверяет структуру своей «парной» таблицы, — достаточно ли таблицы похожи, чтобы позволить репликацию?

Базовая структура события сопоставления таблиц показана на рис. 6-8. Общий заголовок, — заголовок, который присутствует во всех событиях репликации, — содержит основную информацию о событии. За общим заголовком следует информация, специфическая для события сопоставления таблиц. Смысл большинства полей, представленных на рис. 6-8, очевиден, но представление типов полей мы рассмотрим подробно.

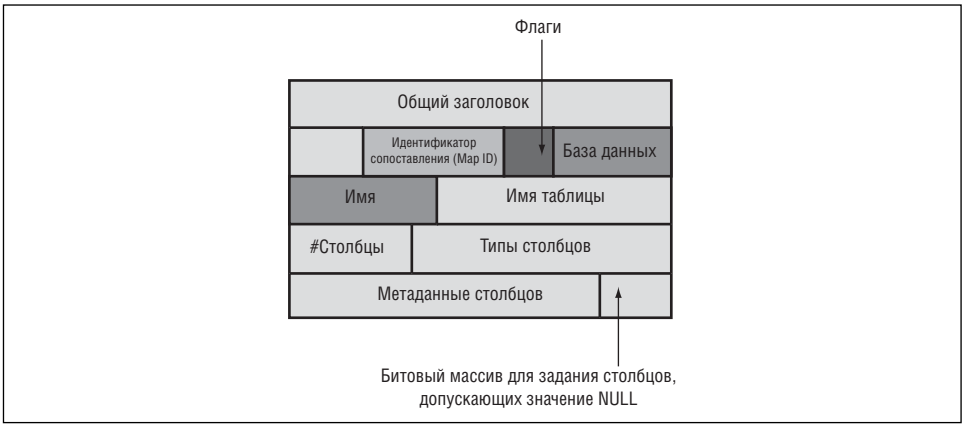


Рис. 6-8. Структура события сопоставления таблиц

Все следующие поля представляют тип столбца:

- *Массив типов столбцов (Column type array)* Массив, перечисляющий базовые типы всех столбцов. Там указывается, является ли столбец целочисленным, строковым, десятичным или относится к любому другому допустимому типу, но уточняющие тип столбца параметры там не приводятся. К примеру, если тип столбца CHAR(5), массив будет содержать значение 254 (константа, соответствующая строке), но сама длина строки (в нашем случае 5) хранится в метаданных столбца, о которых будет сказано ниже.
- *Битовая маска, определяющая столбцы, допускающие значение NULL (Null bit array)* Массив битов, которые показывают, может ли значением поля быть NULL.
- *Метаданные столбцов (Column metadata)*. Массив метаданных для полей, включающий все детали, опущенные в массиве типов столбцов. От типа поля зависит, какие будут у него метаданные. Например, метаданные для десятичных полей DECIMAL задают точность и количество десятичных знаков после запятой, а для строковых полей VARCHAR — максимальную длину поля.

Полностью тип поля определяется комбинацией данных всех трех массивов.

Не все типы информации хранятся в этих массивах, в следующих случаях различия типов на главном и подчиненном серверах не выявляются:

- Нет информации о том, является ли целочисленное поле знаковым или беззнаковым. Это означает, что подчиненный сервер, сверяя структуру таблиц, не может отличить поле со знаком от беззнакового.
- Кодировка строковых типов не сохраняется. Это означает, что репликация между серверами, использующими разные кодировки, не поддерживается, а попытка выполнить подобную репликацию даст неожиданный результат, поскольку байты добавляются в столбец без проверки и без преобразования.

Структура событий строк

На рис. 6-9 приведена структура события строк. Эта структура слегка варьируется в зависимости от типа события (вставка, удаление или обновление).

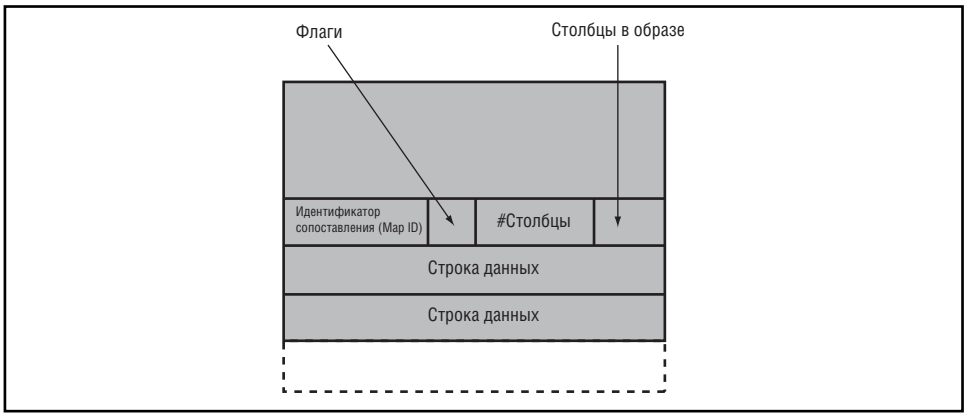


Рис. 6-9. Заголовок события строк

Помимо идентификатора таблицы, указывающего на идентификатор таблицы из предшествующего события сопоставления таблиц, событие строк содержит следующие поля:

- *Ширина таблицы* Ширина таблицы на главном сервере. Закодирована по алгоритму RLE, также, как и для клиентского протокола, поэтому может содержать один либо два байта, но чаще один байт.
- *Битовые массивы столбцов* Столбцы посылаются как часть полезной информации события. Структура информации позволяет главному серверу посылать с каждой строкой выбранный набор столбцов. Существуют два типа битовых массивов столбцов: один для образа строки перед выполнением операции, другой — для образа строки после выполнения операции. Образ строки перед выполнением операции требуется для удалений и обновлений, в том время как образ строки после выполнения операции требуется для вставок и опять-таки обновлений. Подробнее см. табл. 6-1.

Табл. 6-1. События строк и их образы

Образ до выполнения операции	Образ после выполнения операции	Событие
Нет	Вставляемая строка	Вставка строк
Удаляемая строка	Нет	Удаление строк
Значения столбцов до обновления	Значения столбцов после обновления	Обновление строк

Обработка событий

Поскольку несколько событий могут представлять единственный оператор, выполненный на главном сервере, подчиненный сервер должен хранить информацию о состоянии для корректной обработки событий строк при наличии конкурирующих потоков, обновляющих те же таблицы. Вспомним, что каждый оператор двоичного журнала начинается с одного или более событий сопоставления таблиц, за которыми следует одно или более событий строк одинакового типа. Выполнение оператора из двоичного журнала производится по следующей схеме:

1. Каждое событие считывается из журнала ретрансляции.
2. Если событие представляет собой событие сопоставления таблиц, поток SQL извлекает информацию о таблице и сохраняет представление определения таблицы на главном сервере.
3. Когда появляется первое событие строки, все таблицы в списке блокируются.
4. Для каждой таблицы из списка поток выясняет, совместимо ли ее определение на главном сервере с определением на подчиненном.
5. Если таблицы не совместимы, поток докладывает об ошибке и останавливает репликацию на подчиненном сервере.
6. События строк выполняются по схеме, предлагаемой в этом разделе ниже, пока поток не считает последнее событие оператора, — событие, у которого установлен флажок конца оператора.

Следование описанной схеме необходимо для того, чтобы корректно заблокировать таблицы на подчиненном сервере, по подобной схеме оператор выполняется на главном сервере. Все таблицы блокируются на шаге 3 и затем проверяются на шаге 4. Если не заблокировать таблицы перед сверкой определений, поток на подчиненном сервере может вклиниться между этапами и изменить определение, из-за чего в дальнейшем применение событий строк потерпит неудачу.

Каждое событие строки состоит из набора строк, используемых по-разному в зависимости от типа событий. В событиях Delete_rows и Write_rows каждая строка представляет собой изменение. В событии Update_rows необходимо иметь две строки, — одну для локализации строки, подлежащей обновлению, и другую с новыми значениями, — поэтому такое событие состоит из четного числа строк, где каждая пара представляет одно обновление.

Для событий, включающих образ до выполнения оператора, требуется организовать поиск, позволяющий правильно локализовать строку, над которой операция будет производиться, — для события Delete_rows ищется удаляемая строка, а для события Update_rows, соответственно, обновляемая. Поиск ведется в порядке убывания приоритета:

- *Отбор по первичному ключу* Если таблица на подчиненном сервере имеет первичный ключ, производится отбор по первичному ключу. Этот метод самый эффективный.

- *Сканирование индекса* Если первичный ключ для таблицы не определен, но имеется индекс, он и будет использован для поиска нужной строки. Все строки индекса сканируются, и значения в столбцах сравниваются со значениями строки, пришедшей от главного сервера.

  Если удается обнаружить совпадающую строку, она и будет использована для событий Delete_rows или Update_rows. Если подходящей строки найдено не будет, подчиненный сервер остановит репликацию и выдаст соответствующую ошибку.
- *Сканирование таблицы* Если в таблице нет ни первичного ключа, ни индекса, для поиска удаляемой или обновляемой строки будет выполнено полное сканирование таблицы.

  Как и при сканировании индекса, каждая строка таблицы сравнивается со строкой, полученной от главного сервера, и если для строки совпадение установлено, строка удаляется или обновляется в соответствии с требуемой операцией.

Поскольку для корректной локализации удаляемой или обновляемой строки используется индекс или первичный ключ подчиненного, а не главного сервера, следует помнить следующее:

- Если таблица на подчиненном сервере имеет первичный ключ, поиск подходящей строки будет выполняться быстро. Если у таблицы первичного ключа нет, для поиска требуемой строки подчиненному серверу придется выполнять либо полное сканирование таблицы, либо сканирование индекса, что скажется на скорости выполнения.
- Индексы на главном и подчиненном серверах могут быть различными.

Независимо от выбранного типа репликации имеет смысл создать для реплицируемой таблицы первичный ключ.

Поскольку при логической репликации каждый оператор реально выполняется, первичный ключ существенно повышает эффективность и этой репликации, ускоряя обновление и удаление строк.

События и триггеры

В логической и построчной репликации обработка событий и триггеров имеет свои нюансы. Единственное различие для событий состоит в том, что построчная репликация вместо событий запроса (Query) генерирует события строк.

А вот ситуация с триггерами оказывается несколько иной (и более интересной).

Как мы видели в главе 3, при репликации уровня операторов на подчиненный сервер реплицируются и определения триггеров, поэтому при выполнении оператора, затрагивающего таблицу с триггером, триггер на подчиненном сервере также срабатывает.

Для построчной репликации не важно, по какой причине изменилась строка: пришли ли изменения от триггера, хранимой процедуры, события

или непосредственно от оператора. Если строки, измененные триггером, реплицируются на подчиненный сервер, там уже нет необходимости запускать сам триггер. На самом деле запуск триггера на подчиненном сервере даст некорректный результат.

Обратимся к лист. 6-19, определяющему таблицу с триггером.

Лист. 6-19. Определение таблиц и связанных с ними триггеров

```
CREATE TABLE log(
   number INT AUTO_INCREMENT PRIMARY KEY,
   user CHAR(64),
   brief TEXT
);

CREATE TABLE user (
   id INT AUTO_INCREMENT PRIMARY KEY,
   email CHAR(64),
   password CHAR(64)
);

CREATE TRIGGER tr_update_user AFTER UPDATE ON user FOR EACH ROW
   INSERT INTO log SET
      user = NEW.email,
      brief = CONCAT("Пароль изменен с '",
                     OLD.password, "' на '",
                     NEW.password, "'");

CREATE TRIGGER tr_insert_user AFTER INSERT ON user FOR EACH ROW
   INSERT INTO log SET
      user = NEW.email,
      brief = CONCAT("Добавлен пользователь '", NEW.email, "'");
```

Для определенных таким образом таблиц и триггеров можно выполнить следующую последовательность операторов:

```
master> INSERT INTO user(email,password) VALUES ('mats@example.com', 'xyzzy');
Query OK, 1 row affected (0.05 sec)

master> UPDATE user SET password = 'secret' WHERE email = 'mats@example.com';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

master> SELECT * FROM log;
+--------+-------------+-----------------------------------------------------+
| number | user        | brief                                               |
+--------+-------------+-----------------------------------------------------+
|      1 | mats@sun.com | Добавлен пользователь 'mats@example.com'           |
|      2 | mats@sun.com | Пароль изменен с 'xyzzy' на 'secret'               |
+--------+-------------+-----------------------------------------------------+
2 rows in set (0.00 sec)
```

Это не самый надежный подход, но ситуация теперь вам ясна. Как же будут эти изменения выглядеть в двоичном журнале в случае построчной репликации?

```
master> SHOW BINLOG EVENTS IN 'mysqld1-bin.000054' FROM 2180;
+------------------+------+-------------+-----------+-------------+-------------------------+
| Log_name         | Pos  | Event_type  | Server_id | End_log_pos | Info                    |
+------------------+------+-------------+-----------+-------------+-------------------------+
| master-bin.000054 | 2180 | Query       |         1 |        2248 | BEGIN                   |
| master-bin.000054 | 2248 | Table_map   |         1 |        2297 | table_id: 24 (test.user) |
| master-bin.000054 | 2297 | Table_map   |         1 |        2344 | table_id: 26 (test.log)  |
| master-bin.000054 | 2344 | Write_rows  |         1 |        2397 | table_id: 24             |
| master-bin.000054 | 2397 | Write_rows  |         1 |        2471 | table_id: 26 flags:      |
|                  |      |             |           |             |   STMT_END_F             |
| master-bin.000054 | 2471 | Query       |         1 |        2540 | COMMIT                   |
| master-bin.000054 | 2540 | Query       |         1 |        2608 | BEGIN                    |
| master-bin.000054 | 2608 | Table_map   |         1 |        2657 | table_id: 24 (test.user) |
| master-bin.000054 | 2657 | Table_map   |         1 |        2704 | table_id: 26 (test.log)  |
| master-bin.000054 | 2704 | Update_rows |         1 |        2783 | table_id: 24             |
| master-bin.000054 | 2783 | Write_rows  |         1 |        2873 | table_id: 26 flags:      |
|                  |      |             |           |             |   STMT_END_F             |
| master-bin.000054 | 2873 | Query       |         1 |        2942 |COMMIT                    |
+------------------+------+-------------+-----------+-------------+-------------------------+
12 rows in set (0.00 sec)
```

Видно, что каждый оператор рассматривается как отдельная транзакция, включающая единственный оператор. Оператор изменяет две таблицы, — *test.user* и *test.log*, — следовательно, представление оператора в двоичном журнале начинается с двух событий сопоставления таблиц. При репликации на подчиненном сервере события из двоичного журнала выполняются непосредственно, не вызывая срабатывания триггера, что позволяет избежать запуска кода триггеров на подчиненном сервере.

Фильтрация

Механизм фильтрации также разнится для построчной и логической репликации. Как сказано в главе 3, фильтрация на уровне операторов применяется к оператору целиком, — либо оператор выполняется целиком, либо не выполняется вовсе, — потому что часть оператора выполнить невозможно. Значение параметра фильтрации уровня базы данных сравнивается с именем текущей базы данных, а не с именем базы данных, содержащей изменяемую таблицу.

Построчная репликация в данном контексте выглядит более гибкой. Поскольку для указанной таблицы выбираются и реплицируются строки, фильтр можно применить непосредственно к самой обновляемой таблице, можно даже пропустить отдельные строки через фильтр с произвольным условием. Поэтому построчная репликация при фильтрации изменений ис-

ходит из того, какая таблица обновляется, а не из того, какая база данных установлена для оператора текущей.

Выясним, что произойдет, если фильтр подчиненного сервера настроен на отбрасывание базы данных с именем *ignore_me*. Что даст следующий оператор при логической и построчной репликации?

```
USE test; INSERT INTO ignore_me.t1 VALUES (1),(2);
```

При логической репликации оператор будет выполнен (поскольку база *ignore_me* не текущая), но при построчной репликации изменения таблицы *t1* будут отброшены, поскольку база *ignore_me* указана в списке фильтрации.

Продвигаясь в этом направлении, посмотрим, что произойдет, если оператор обновляет несколько таблиц (в разных базах данных)?

```
USE test; UPDATE ignore_me.t1, test.t2 SET t1.a = 3, t2.a = 4 WHERE t1.a = t2.a;
```

При логической репликации оператор будет выполнен (в предположении, что таблица *ignore_me.t1* существует, — раз база данных игнорируется, таблицы может и не быть в наличии), и обе таблицы, *ignore_me.t1* и *test.t2*, окажутся обновленными. При построчной репликации, напротив, обновлена будет лишь таблица *test.t2*.

Частичное выполнение операторов

Как уже отмечалось, логическая репликация работает вполне успешно, если не принимать в расчет возможные отказы, сбои и недетерминированное поведение. Поскольку отказы и сбои происходят обычно в самый неподходящий момент, вполне вероятно, что часть операторов будет исполнена лишь частично.

Подобное происходит и тогда, когда количество строк, обрабатываемых операторами UPDATE, DELETE или INSERT, ограничивается искусственно. Явным образом такое поведение устанавливается предложением LIMIT, но похожие ситуации возникают и при работе с нетранзакционной таблицей, — к примеру, ошибка повторяющегося ключа обрывает выполнение оператора на середине.

В таких случаях изменения, описанные оператором, прикладываются только к начальному набору строк. Критерии упорядочивания строк подчиненного и главного сервера могут различаться, поэтому различаются и наборы строк, к которым применяется оператор на главном и на подчиненном серверах.

Механизм БД MyISAM содержит все строки в том порядке, в котором они добавлялись. Исходя из этого, вы можете решить, что в случае частичных изменений затронутыми окажутся одни и те же строки. К сожалению, это не так. Если подчиненный сервер был получен клонированием главного при помощи логического резервного копирования или восстановлен из резервной копии главного сервера, вполне вероятно, что при этом изменился порядок вставки.

Обычно проблему можно решить добавлением предложения ORDER BY, но даже этот путь не является стопроцентно надежным, и частичное выполнение оператора при сбое все еще представляется реальной угрозой.

Заключение

Завершилась последняя глава, посвященная репликации MySQL. В ней мы рассмотрели расширенные возможности репликации, — например, более надежный перевод подчиненных серверов в ранг главных, — обсудили секреты и приемы, позволяющие избежать повреждений баз данных при сбоях, высказали различные соображения о многоисточниковой репликации и проанализировали ее конфигурацию, а в конце подробно разобрали построчную репликацию.

В следующих главах мы обратимся к тематике, связанной с построением надежных центров обработки данных: мониторингу и настройке производительности механизмов БД и снова к репликации.

Возвращаясь с обеда, Джоэл встретил в коридоре босса.

— Добрый день, г-н Саммерсон.

— Привет, Джоэл.

— Вы прочли мой отчет?

— Да, он неплох. Я передал его в другие отделы на отзыв. А, вообще, твой отчет стоит включить в наши СРП.

Джоэл догадался, что речь идет о стандартных рабочих процедурах.

— Я попросил рецензентов переслать тебе свои комментарии. Для СРП текст придется немного подправить, но ты справишься, я уверен.

— Спасибо, сэр...

Г-н Саммерсон кивнул, похлопал Джоэла по плечу и продолжил свой путь.

ЧАСТЬ II

Мониторинг и восстановление после сбоев

Если вам вверена сложная многосерверная система, приходится следить за ее состоянием. Эта часть книги посвящена мониторингу и включает темы, связанные с производительностью, резервным копированием и борьбой с неизбежными сбоями.

ГЛАВА 7

Основы мониторинга

Джоэл расставил на столе чашку кофе, фрукты и чизкейк, иронизируя про себя над этим подобием завтрака. Надо сказать, его завтраки стали весьма оригинальными с тех пор, как он обнаружил торговый центр по пути на работу.

Джоэл включил монитор и, открывая бумажный стаканчик с кофе, проверил почту. Пробежав взглядом по заголовкам в надежде, что новых заданий от руководства нет, он обнаружил несколько сообщений от пользователей, судя по темам — о проблемах с производительностью.

— Да, пожалуй, и правда что-то идет не так, — бормотал он, вчитываясь в жалобы на приложения, которые слишком медленно выполняют запросы к базам данных.

Взяв кусок пирога, Джоэл задумался о причинах проблемы. «Еще вчера все было прекрасно», — размышлял он. После нескольких глотков кофе он вспомнил, что на практике в колледже он читал что-то о мониторинге производительности.

Джоэл доел пирог и потянулся к любимой книге по MySQL.

— Наверняка здесь что-то есть об этом, — решил он.

Как узнать, что с серверами не все в порядке? Если дожидаться жалоб от пользователей, может оказаться, что проблемы возникли довольно давно, а запущенные проблемы сложнее диагностировать и устранять.

Эту главу мы начнем с рассмотрения мониторинга MySQL на уровне операционной системы при помощи эталонных инструментов, доступных в различных ОС. Начинать следует именно с этого, поскольку производительность службы или приложения зависит от производительности ОС и оборудования. Если производительность ОС невысока, то низкой будет и производительность системы баз данных или приложения.

Сначала мы обсудим причины необходимости мониторинга, затем разберем простейшие задачи по мониторингу популярных ОС и выясним, как мониторинг облегчает обслуживание систем. После освоения этого материала можно приступать к мониторингу СУБД. В следующей главе мы подробно поговорим о мониторинге серверов MySQL и дадим некоторые практические советы по решению распространенных проблем с производительностью.

Способы мониторинга

Когда говорят о мониторинге, обычно представляют какую-то систему раннего оповещения, выявляющую возникшие проблемы. Однако, «вести мониторинг» означает «выявлять процессы, наблюдать за ними и записывать информацию о них, используя инструменты, не влияющие на эти процессы» (*http://www.dictionary.com*). Такая система раннего оповещения сочетает автоматизированный сбор информации и отправку предупреждений.

Операционные системы Linux и Unix весьма сложны и имеют множество параметров, влияющих на все аспекты системы. Настройка этих систем с целью повышения производительности — скорее искусство, чем наука. В отличие от некоторых настольных операционных систем, Linux, Unix и их разновидности не «прячут» инструментарий для настройки и не ограничивают параметры, которые можно настраивать. Другие же операционные системы, например Mac OS X и Windows скрывают многие системные механизмы за дружественным визуальным интерфейсом пользователя.

К примеру, Mac OS X — очень элегантная и надежная система, которая обычно не требует вмешательства пользователя. Однако для нее создано множество полезных инструментов для мониторинга и настройки, надо только знать, где их искать.

ОС Windows имеет множество версий, самая свежая на момент написания книги — Windows 7. К счастью, большинство версий включает один и тот же набор инструментов мониторинга, позволяющих настраивать систему под собственные нужды. Хотя Windows считается менее дружественной, чем Mac OS X, эта ОС предоставляет еще больше возможностей для настройки.

Существует три основных области мониторинга: производительность системы, производительность приложений и безопасность. Можно выделить более конкретные категории, но обычно задача относится к одной из этих областей.

Для каждой задачи имеется особые наборы инструментов (частично они перекрываются) и свои цели. Например, чтобы настроить систему для максимально эффективной работы, необходим мониторинг ее производительности. Мониторинг производительности приложений позволяет наиболее эффективно их использовать, а мониторинг системы безопасности — обеспечить максимальную защиту системы.

Мониторинг сервера MySQL относится к мониторингу приложений. Дело в том, что MySQL, как и большинство СУБД, позволяет наблюдать за множеством переменных и индикаторов состояния, никак или почти никак не связанных с операционной системой. Однако на производительность СУБД очень сильно влияет производительность ОС, поэтому, прежде чем приступать к диагностике проблем СУБД, важно убедиться, что операционная система работает нормально.

Поскольку наша цель — мониторинг MySQL для обеспечения максимально эффективной работы СУБД, в следующих разделах мы обсудим мониторинг производительности операционной системы. Мониторинг безопасности оставим руководствам, посвященным этому вопросу.

Что дает мониторинг

Существует два подхода к мониторингу. Иногда требуется убедиться, что ничего не изменилось (производительность не снизилась и не появились бреши в безопасности), а иногда выяснить, что изменилось или пошло не так. Мониторинг системы, позволяющий убедиться, что ничего не изменилось, называется упреждающим мониторингом, а мониторинг, показывающий, что пошло не так, называется реагирующим мониторингом. К сожалению, к реагирующему мониторингу прибегают гораздо чаще. Лишь некоторые специалисты имеют достаточно времени и ресурсов для проведения упреждающего мониторинга. Таким образом, некоторые специалисты считают мониторингом только реагирующий мониторинг.

Однако, если выделить время на упреждающий мониторинг системы, реагирующий может и не потребоваться. Например, если пользователи жалуются на плохую производительность системы (причина номер один для проведения реагирующего мониторинга), вы не сможете узнать, насколько велико ухудшение, если у вас нет предшествующих результатов мониторинга для сравнения. Регистрация подобных результатов называется *определением эталонных показателей* системы. То есть, в течение некоторого времени выполняется мониторинг производительности системы при низкой, нормальной и высокой нагрузке. Частое и последовательное выполнение таких замеров позволяет определить типичную производительность системы при различной нагрузке. Таким образом, когда пользователи начинают жаловаться на проблемы с производительностью, можно выполнить замеры и сравнить результаты с эталонными показателями. Если регистрировать информацию достаточно подробно, в большинстве случаев можно сразу понять, в какой части системы произошли изменения.

Компоненты системы, подлежащие мониторингу

При выполнении мониторинга производительности следует анализировать четыре основных компонента системы:

- *процессор* — проверяйте, насколько загружен процессор и какова его пиковая нагрузка;
- *оперативная память* — проверяйте, какой объем оперативной памяти используется, а какой остается доступным для выполнения программ;

- *дисковое пространство* — проверяйте, сколько дискового пространства доступно, сколько занято, сколько требуется и каково время отклика диска;
- *сеть* — проверяйте пропускную способность, время ожидания и процент ошибок при взаимодействии с другими системами по сети.

Процессор

Мониторинг центрального процессора системы позволяет убедиться, что процессов, вышедших из-под контроля нет, а циклы процессора равномерно распределяются среди выполняющихся программ. Для этого можно просмотреть список запущенных программ и определить, какой процент ресурсов процессора использует каждая из них. Другой способ — проверка средней загруженности системных процессов. Большинство операционных систем предоставляет несколько способов оценки производительности центрального процессора.

Процесс — рабочая единица в операционных системах Linux или Unix. Программа может запускать один или несколько процессов одновременно. Многопоточные приложения, такие как MySQL, обычно представлены несколькими процессами.

При загруженности процессора и большой конкуренции процессов за его ресурсы система может функционировать очень медленно и даже временами «зависать». В таких случаях необходимо уменьшить число процессов или ограничить использование процессора процессами, занимающими большую часть процессорного времени. Однако прежде следует провести мониторинг процессора и убедиться, что проблема вызвана именно высокой загрузкой процессора, так как чаще причиной медленной работы является нехватка оперативной памяти (об этом говорится в следующем разделе).

Перечислим распространенные способы решения проблем с перегрузкой процессора.

- *Добавление нового сервера для выполнения части процессов.* Это, безусловно, лучший вариант, но он требует денег. Опытные системные администраторы часто могут найти другие способы снижения загруженности процессора, особенно если руководство предпочитает экономить деньги организации, а не время сотрудников.
- *Исключение необязательных процессов.* Во множестве систем работают фоновые процессы, полезные в определенных ситуациях, но большую часть времени лишь обременяющие систему. Администратор обязан очень хорошо разбираться в системе, чтобы определить, какие процессы не являются необходимыми.
- *Завершение процессов, вышедших из-под контроля.* Такие процессы могут порождаться сбойными приложениями. Если производительность падает редко или скачкообразно, вероятно, это их вина. Если не удается завершить вышедший из-под контроля процесс корректно, возможно, придется его уничтожить.

- *Оптимизация приложений.* Некоторые приложения занимают больше процессорного времени и других ресурсов, чем им на самом деле нужно. Так, в СУБД к этому приводит плохо продуманный SQL-код запросов.
- *Понижение приоритета процесса.* Некоторые фоновые процессы, например, генераторы отчетов, можно заставить работать медленнее, отдав ресурсы интерактивным процессам.
- *Изменение графика выполнения.* Возможно, какие-то процессы можно выполнять ночью, когда система менее загружена.

Процессы, потребляющие слишком много процессорного времени, называют *привязанными к CPU* (*CPU-bound*) или *привязанными к процессору* (*processor-bound*), что означает, что они не приостанавливаются для операций ввода-вывода и не могут быть выгружены из оперативной памяти.

Если вы не обнаружили задач, конкурирующих за ресурсы процессора, а процессов, интенсивно потребляющих процессорное время, мало или нет вовсе, скорее всего проблема производительности связана с чем-то другим: с ожиданием операций дискового ввода-вывода, недостаточностью оперативной памяти, чрезмерной подкачкой страниц и т.п.

Оперативная память

Мониторинг оперативной памяти позволяет убедиться, что приложения не запрашивают слишком большого объема оперативной памяти, что приводит к перерасходу ресурсов системы на управление этой памятью. Со времени изобретения компьютера памяти произвольного доступа (ОЗУ или основной памяти) операционные системы всегда ощущали ее нехватку, и появились сложные механизмы для хранения неиспользуемых фрагментов или страниц основной памяти на диске. Такой механизм, называемый *подкачкой*, позволяет запускать в системе больше процессов, чем одновременно помещается в оперативной памяти, путем сохранения на диске страниц памяти, занятых приостановленными процессами, с последующим их считыванием при возобновлении процесса. Хотя затраты на перемещения страницы оперативной памяти на диск и обратно относительно высоки (эта операция выполняется гораздо дольше, чем прямой доступ к основной памяти), современные операционные системы делают это достаточно быстро. Проблем обычно не возникает, если только интенсивность страничного обмена не достигает такого уровня, что процессор и диск не успевают вовремя реагировать на запросы.

Однако периодически операционная система может довольно активно выполнять подкачку страниц, чтобы освободить память для своих нужд. Наблюдайте за использованием оперативной памяти в течение достаточно длительного времени, чтобы убедиться, что замедление вызвано не стандартной операцией подкачки.

Если происходит интенсивная подкачка страниц, скорее всего низкая доступность оперативной памяти вызвана вышедшим из-под контроля про-

цессом, запрашивающим слишком много памяти, или же большим количеством процессов, требующих большого объема памяти. При таком высоком страничном обмене (т.н. *пробуксовке*) следует действовать так же, как и при высокой конкуренции за процессорное время. Процессы, потребляющие слишком много памяти, называются *привязанными к оперативной памяти*.

Добавление оперативной памяти представляется естественным решением проблем производительности, связанных с памятью. Это, скорее всего, поможет справиться с ситуацией, но не исключено, что память некорректно распределяется среди подсистем.

В этом случае можно, к примеру, выделить разные объемы памяти разным частям системы, таким как ядро или файловая система, или разным приложениям, поддерживающим подобную настройку, таким как MySQL. Можно также изменить настройки виртуальной памяти, чтобы операционная система начинала подкачку раньше.

Будьте крайне осторожны, настраивая подсистемы памяти на серверах. Сверяйтесь с документацией или специальной литературой, посвященной повышению производительности конкретной операционной системы.

Если мониторинг оперативной памяти показал, что подкачка выполняется не слишком часто, но производительность все равно низкая, то проблемы, скорее всего, связаны с другими подсистемами.

Диск

Мониторинг использования дисков позволяет убедиться, что свободного дискового пространства достаточно, а пропускной способности ввода-вывода хватает для выполнения процессов без заметных задержек. Для этого можно измерять интенсивность передачи данных с диска и на диск для отдельных процессов или общую. Интенсивность передачи данных для отдельного процесса — это объем данных, которые может считать или записать этот процесс. Общая интенсивность передачи данных — это максимальная пропускная способность при чтении и записи данных на диск. В системах с несколькими контроллерами дисков общую скорость передачи данных можно измерять для каждого контроллера в отдельности.

Проблемы с производительностью возникают, когда один или несколько процессов занимают слишком большую долю максимальной интенсивности передачи данных. Это может весьма негативно сказываться на производительности системы, как в случае, когда какой-то процесс занимает слишком много циклов процессора: такой процесс «перекрывает кислород» остальным процессам, заставляя их подолгу дожидаться доступа к диску.

Процессы, забирающие слишком большую часть канала обмена данными с диском, называются *привязанными к диску*; эти процессы пытаются обращаться к диску чаще, чем допустимо. Если удастся уменьшить нагрузку, возлагаемую привязанным к диску процессом на систему ввода-вывода, остальным процессам будет доступно больше пропускной способности.

Один из способов оптимизации процесса, выполняющего много операций ввода-вывода, — увеличение размера блока файловой системы для повышения эффективности передачи большого объема данных и снижения нагрузки, вызываемой привязанным к диску процессом.

При настройке файловой системы на серверах с единственным контроллером или диском следует проявлять осторожность. Обращайтесь к документации или специальной литературе по конкретной операционной системе.

Если позволяют ресурсы, для решения проблемы с высокой конкуренцией за доступ к диску можно добавить еще один контроллер дисков и перенести на него данные для одного из привязанных к диску процессов. Другой способ — перенос привязанного к диску процесса на другой, не столь загруженный сервер. И, наконец, в некоторых случаях пропускную способность диска можно повысить модернизацией дисковой подсистемы с переходом на более быстрые технологии.

Имеются различные мнения насчет того, с чего начать оптимизацию и какой метод будет предпочтительнее. Мы полагаем, что:

- если требуется запустить много процессов, следует увеличить скорость передачи данных диска или поделить процессы между различными дисковыми массивами или системами;
- если требуется запустить несколько процессов, работающих с большими объемами данных, следует увеличить максимальную интенсивность обмена данных для отдельных процессов, увеличив размер блока файловой системы.

Возможно, придется искать нечто среднее между этими двумя вариантами, перемещая некоторые процессы на другие системы.

Сетевая подсистема

Мониторинг сетевых интерфейсов позволяет убедиться, что пропускная способность сети достаточна, а обмен данными выполняется на приемлемом уровне.

Процессы, занимающие слишком большую часть полосы пропускания сети в попытках передать больше данных, чем позволяет конфигурация сети или аппаратное обеспечение, называются *привязанными к сети*. Такие процессы не дают остальным процессам зарезервировать требуемую им часть сетевого канала, вызывая задержки.

Проблемы с пропускной способностью сети можно обнаружить, отслеживая процент от максимальной пропускной способности сетевого интерфейса, занятый процессами. Подобные проблемы решаются назначением процессам определенных портов сетевого интерфейса.

На проблемы с качеством данных, передающихся по сети, указывает большое число ошибок при передаче. К счастью, операционная система и приложения, передающие данные, обычно используют *контрольные суммы*

или другие алгоритмы для выявления ошибок, однако повторная передача данных может значительно увеличивать нагрузку на сеть и ОС. Для решения подобных проблем может потребоваться перенос части приложений на другие узлы сети или установка дополнительных сетевых карт. Обычно требуется выявление источника проблем с последующей заменой сетевого оборудования, перенастройкой сетевых протоколов или перемещением системы в другую подсеть.

Иногда говорят, что процесс *привязан к вводу-выводу* или что ему *не хватает ввода-вывода*. Обычно это означает, что процесс занимает слишком большую часть пропускной способности диска или сети.

Решения для мониторинга

Для каждой из четырех подсистем, рассмотренных выше, современные операционные системы предлагают собственные специализированные инструменты, предоставляющие информацию о состоянии подсистемы. В основном это автономные приложения, не связанные (по крайней мере, напрямую) с другими инструментами. Как будет показано далее, эти инструменты предоставляют много возможностей, но запись и анализ предоставляемой ими информации требует определенных усилий.

К счастью, для мониторинга большинства операционных систем и СУБД существуют решения сторонних производителей. Ниже перечислено несколько самых заметных из них. Имеет смысл проконсультироваться с поставщиками вашей системы о том, какое решение подойдет лучше всего и будет совместимо с имеющейся инфраструктурой. Большинство поставщиков предоставляют средства мониторинга в качестве дополнения к стандартной комплектации.

- up.time (*http://www.uptimesoftware.com*);
- Cacti (*http://www.cacti.net*);
- KDE System Guard (KSysGuard) (*http://docs.kde.org/stable/en/kdebase-workspace/ksysguard/index.html*);
- Gnome System Monitor (*http://library.gnome.org/users/gnome-system-monitor*);
- Nagios (*http://www.nagios.org*);
- Sun Management Center (*http://www.sun.com/software/products/sunmanagementcenter/index.xml*);
- MySQL Enterprise Monitor (*http://www.mysql.com/products/enterprise/monitor.html*).

Подробную информацию о MySQL Enterprise Monitor, а также автоматизации мониторинга и отчетности см. в гл. 13.

В следующих разделах описываются встроенные инструменты мониторинга нескольких распространенных операционных систем. Команды Linux

и Unix мы рассмотрим подробнее, так как они в точности подходят для изучения описанных нами стратегий и проблем производительности. Однако мы поговорим и об инструментах мониторинга для Mac OS X и Windows.

Мониторинг в системах Linux и Unix

При мониторинге систем баз данных в Linux или Unix применяются инструменты для наблюдения за процессором, оперативной памятью, дисками, сетью и даже за безопасностью и пользователями. Все инструменты запускаются из командной строки, в классическом стиле Unix, и большинство из них хранится в папках *bin* и *sbin*. В табл. 7-1 приведен список и краткое описание инструментов, которые мы считаем полезными.

Табл. 7-1. Инструменты мониторинга для Linux и Unix

Утилита	Описание
Ps	Выводит список выполняющихся процессов
Top	Показывает активность процессов с сортировкой по нагрузке на процессор
Vmstat	Отображает информацию об оперативной памяти, подкачке страниц, передаче блоков и активности процессора
Uptime	Показывает время работы системы. Также выводится информация о количестве подключенных пользователей и средней загрузке системы в течение одной, пяти и пятнадцати минут
Free	Отображает информацию об использовании памяти
Iostat	Показывает среднюю активность диска и загрузку процессора
Sar	Выдает отчет об активности системы. Позволяет собирать и выводить данные о различных аспектах деятельности системы
Pmap	Показывает карту использования памяти процессами
Mpstat	Показывает информацию об использовании процессоров в многопроцессорных системах
Netstat	Показывает информацию об активности сети
Cron	Подсистема, позволяющая запускать процессы по расписанию. Можно запланировать выполнение перечисленных выше утилит, чтобы регулярно собирать статистику или проверять статистику в отдельные моменты времени, например при пиковой или минимальной нагрузке

Некоторые операционные системы предоставляют дополнительные или альтернативные инструменты, о них рассказывается в документации ОС.

Видно, насколько широк выбор инструментов, предоставляющих много потенциально полезных сведений. В следующих разделах обсуждаются некоторые из наиболее популярных инструментов и вкратце объясняются их возможности по выявлению проблем, описанных выше.

Активность процессов

Информацию о процессах, выполняющихся в системе, можно получать несколькими командами. Чаще всего используются команды top, iostat, mpstat и ps.

Команда top

Команда top предоставляет общую информацию о системе и показывает в динамике процессы системы, упорядоченные по убыванию интенсивности использования CPU. Обычно отображается информация о процессе, включая идентификатор процесса, имя пользователя, запустившего процесс, приоритет процесса, процент использования CPU, количество времени, потребляемое процессом, и, конечно, команду, запустившую процесс. Однако в некоторых ОС отчеты могут иметь отличия. Вероятно, это самая популярная утилита в наборе, поскольку она предоставляет моментальный снимок состояния системы каждые несколько секунд. На рис. 7-1 показан результат выполнения команды top в Linux (Ubuntu) при умеренной нагрузке.

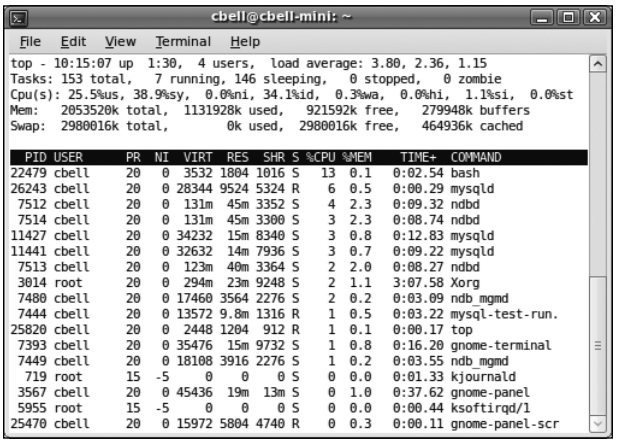


Рис. 7-1. Команда top

Общая информация о системе показана в верхней части листинга и содержит некоторые интересные данные. Там приведены проценты использования процессорного времени для пользователей (%us), для системы (%sy), для пользовательских процессов с измененным приоритетом (%ni), для ожидания операций ввода-вывода (%wa) и даже процент времени, затраченный на обработку аппаратных и программных прерываний. Также указан объем оперативной памяти и доступное для подкачки пространство: сколько занято, сколько свободно и размер буферов.

За общей информацией следует список процессов в порядке убывания процента использования времени процессора (отсюда и название команды top). В нашем примере лидером является командная оболочка Bash, за которой следует один или несколько экземпляров MySQL.

Приоритет процессов

В Linux и Unix можно менять приоритет процесса. Это может потребоваться для понижения приоритета процессов, требующих слишком большую долю ресурсов процессора, или для процессов, которые не являются срочными и могут выполняться долго, но которые нежелательно завершать и перезапускать в другое время. Для изменения приоритетов процессов используются команды nice, ionice и renice.

В большинстве версий Linux и Unix процессы, приоритет которых был изменен, объединяются в группу с названием nice. Это позволяет получать статистику по модифицированным процессам, избавляя от необходимости запоминать и вручную сопоставлять информацию. Команды, показывающие процессорное время, занимаемое процессами из группы nice, дают возможность просматривать, какую часть ресурсов процессора потребляют эти процессы по отношению к оставшейся части системе. К примеру, большое значение этого параметра указывает на то, что как минимум один процесс имеет слишком высокий приоритет.

Возможно, лучший способ применения команды top — просто запустить ее и позволить обновляться каждые три секунды. Если периодически проверять вывод команды в течение некоторого времени, можно увидеть, какие процессы потребляют основную часть процессорного времени. Это позволяет быстро определять наличие вышедших из-под контроля процессов.

Можно изменить частоту обновления выходных данных команды, указав интервал задержки. Например, задержка в три секунды задается командой top –d 3.

В большинстве версий Linux и Unix команда top работает так, как мы описали. Некоторые варианты имеют интерактивные горячие клавиши, позволяющие включать и отключать вывод информации, сортировать список процессов и даже изменять цвет выходных данных. Горячие клавиши и интерактивные функции различаются в разных системах, за информацией о них обращайтесь к странице описания команды top для вашей операционной системы.

Команда iostat

Команда iostat предоставляет различные наборы данных о системе, включая статистику использования процессорного времени, ввода-вывода устройств и даже разделов и сетевых файловых систем. Эта команда полезна при мониторинге процессов, поскольку дает общую картину работы процессов системы и показывает время ожидания системой операций ввода-вывода. На рис. 7-2 показан пример выходных данных команды iostat в системе с умеренной нагрузкой.

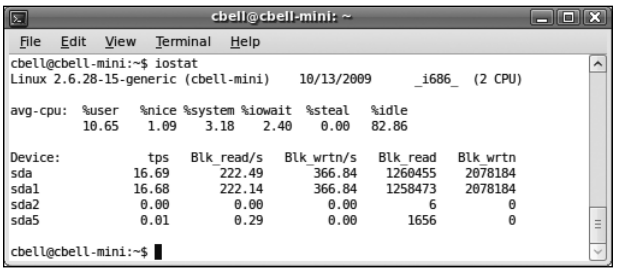


Рис. 7-2. Команда iostat

Команды iostat, mpstat и sar могут не быть установлены по умолчанию, но их можно установить дополнительно. Например, в дистрибутивах Ubuntu они входят в пакет sysstat. Информацию об установке и использовании этих команд см. в документации по вашей операционной системе.

На рис. 7-2 показана информация об использовании процессора с момента запуска системы. Указаны средние значения для всех процессоров. Как видите, система работает на двуядерном процессоре, но выводится только одна строка значений. Эти данные включают проценты использования процессора по следующим позициям:

- выполнение на пользовательском уровне (запущенные приложения);
- выполнение на пользовательском уровне с измененным приоритетом;
- выполнение на уровне системы (процессы ядра);
- ожидание ввода-вывода;
- ожидание виртуальных процессов;
- время простоя (бездействия).

Такие отчеты дают представление о работе системы с момента запуска. В них можно не заметить отдельные периоды с низкой производительностью (т.к. данные усреднены по времени), но вместо этого дается уникальный взгляд на то, как процессы использовали доступные ресурсы процессора или ожидали завершения операций ввода-вывода. К примеру, если значение %idle очень мало, значит, система была очень загружена. Большое значение %iowait может указывать на проблемы с диском. Если значение %system или %nice намного больше, чем %user, вероятно существует дисбаланс между системными процессами и процессами с измененным приоритетом, которые мешают выполнению обычных процессов.

Команда mpstat

Команда mpstat предоставляет во многом ту же информацию, что и iostat, но разбивает данные по процессорам. Если выполнять команду на многопроцессорной системе, будут показаны сведения для каждого из процессоров в отдельности, а также общие данные по всем процессорам. На рис. 7-3 приведен пример выходных данных команды mpstat.

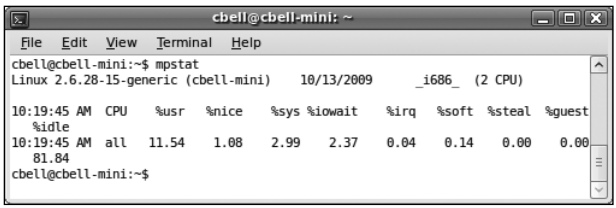


Рис. 7-3. Команда mpstat

Команду mpstat можно запустить, указав интервал обновления. Это полезно, если требуется наблюдать за производительностью процессоров на протяжении периода времени. Например, это можно увидеть, что процессоры не сбалансированы (одному процессору назначено слишком много процессов).

Подробнее о команде mpstat см. в документации операционной системы.

Команда ps

Команда ps относится к тем командам, которые мы используем повседневно, но даже не пытаемся разобраться во всех их возможностях. Эта команда предоставляет моментальный снимок всех процессов системы. Она выводит идентификатор процесса, терминал, с которого запущен процесс, время выполнения процесса и команду, которой процесс был запущен.

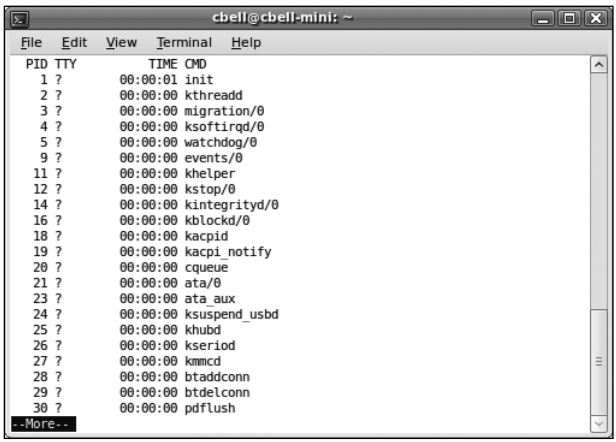


Рис. 7-4. Команда ps

Команда ps может выводить данные несколькими способами, что делает ее очень гибкой. Можно отобразить процессы отдельного пользователя, получить список процессов, связанных с выбранным процессом, и показать их в виде дерева, и даже изменить формат выходных данных. Подробную информацию обо всех возможностях этой команды, доступных в вашей операционной системе, см. в документации.

Выходные данные команды ps можно использовать при диагностике, когда требуется найти процессы, выполняющиеся слишком долго, или проверить состояние процессов (например, не завис ли процесс в состоянии ожи-

дания или в другом подозрительном состоянии). Если процесс не является известным приложением (как, например, MySQL), возможно, следует выяснить, почему он выполняется так долго.

На рис. 7-4 приведен сокращенный пример выходных данных команды ps, запущенной на умеренно загруженной системе.

Также вывод команды ps помогает найти процессы, которые вы не можете опознать, или же множество процессов, запущенных одним пользователем. Часто это указывает на сценарий, порождающий процессы, возможно, просто некорректно настроенный, но нельзя исключить и проблемы в системе безопасности.

Пожалуй, чаще всего команду ps используют для определения идентификаторов процессов определенного приложения. Например, чтобы получить идентификаторы процессов всех запущенных программ mysqld, выполните следующую команду:

```
ps -A | grep mysqld
```

Список всех процессов будет передан команде grep, которая выведет только строки, содержащие подстроку «mysqld». Используя полученные идентификаторы, можно запросить подробную информацию о процессах при помощи других команд.

Существует множество других встроенных в ОС утилит, выводящих информацию о процессах. Как обычно, лучшим источником подробной информации о мониторинге процессов будет документация, посвященная настройке производительности конкретной системы.

Использование памяти

Для получения информации об использовании памяти в системе используется несколько команд. К самым распространенным относятся free и pmap.

Команда free

Команда free показывает объем доступной физической памяти. Отображается общий объем памяти, используемый объем, объем свободной физической памяти, а также аналогичные показатели для пространства подкачки. Кроме того, отображаются буферы памяти, используемые ядром и размер кэша. На рис. 7-5 приведен пример выходных данных команды free для умеренно загруженной системы.

Рис. 7-5. Команда free

В выходных данных для системы Ubuntu (рис. 7-5) колонка shared является устаревшей и не несет никакой информации.

Команду free можно запустить и в режиме опроса, когда статистика регулярно обновляется через заданное количество секунд. Например, команда free -t -s 5 будет опрашивать память каждые 5 секунд.

Команда pmap

Команда pmap предоставляет подробную карту памяти, используемой процессом. Для работы с этой командой необходимо сначала найти идентификатор нужного процесса. Его можно получить командой ps, или даже командой top, если речь идет о процессе, потребляющем много процессорного времени.

Можно также получить карту памяти для нескольких процессов, если перечислить в командной строке их идентификаторы. Например, команда pmap 12578 12579 покажет карту памяти для процессов с идентификаторами 12578 и 12579.

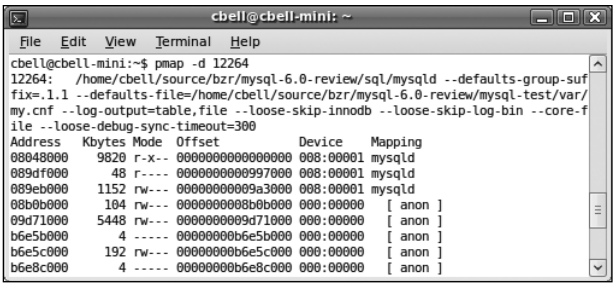


Рис. 7-6. Команда pmap, часть 1

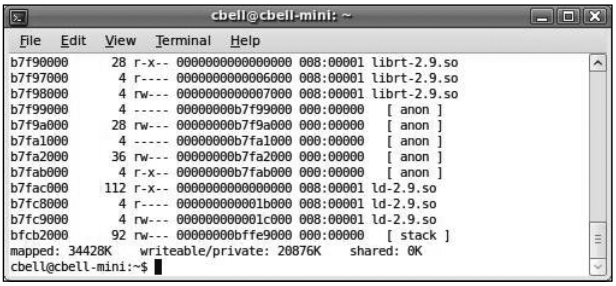


Рис. 7-7. Команда pmap, часть 2

Выходная информация представляет собой подробную карту адресов памяти и частей памяти, используемых процессом в момент создания отчета. Показывается команда, использованная для запуска процесса, включая полный путь и параметры, что очень полезно для определения того, откуда и как был запущен процесс. Вы будете приятно удивлены, обнаружив, как легко таким образом выявить причину ненормального поведения процесса.

Также отображается режим (права доступа) блока памяти. Это удобно для диагностики проблем взаимодействия процессов. На рис. 7-6 и 7-7 приведен пример карты памяти процесса mysqld, запущенного в системе с умеренной нагрузкой.

Обратите внимание, что для листинга выбран формат выходных данных для устройств (указан параметром –d при запуске), и показано, где находится используемая память. Это удобно для диагностики причин потребления процессом большого объема памяти и определения части (например, библиотеки), потребляющей наибольший объем.

На рис. 7-7 видна последняя строка выходных данных команды pmap с полезной общей информацией. В ней показано, сколько памяти выделено файлам, общий объем закрытого адресного пространства и пространства, используемого процессами совместно. Эта информация может оказаться ключевой для решения проблем с выделением и совместным использованием памяти.

Существуют и другие команды и утилиты, выводящие информацию о распределении памяти (например, команда dmesg, выводящая сообщения, появляющиеся при запуске). Информацию о них ищите в соответствующей документации.

Использование диска

Для получения статистики по использованию диска существуют различные команды. В этом разделе мы опишем и продемонстрируем использование команд iostat и sar.

Команда iostat

Как говорилось выше, команда iostat отображает информацию об использовании процессора, а также выводит список всех дисков и сведения о них. В частности, эта команда перечисляет устройства, их скорость передачи данных, число операций записи и чтения блоков в секунду, а также общее число прочитанных и записанных блоков. Для простоты рис. 7-8 повторяет рис. 7-2, представляющий собой пример выполнения команды iostat в системе с умеренной нагрузкой.

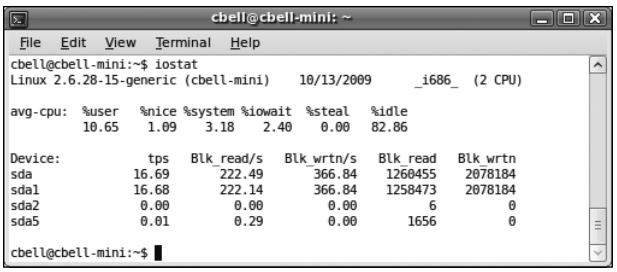


Рис. 7-8. Команда iostat

Этот отчет очень важен для диагностики проблем с диском. С первого взгляда можно понять, какие устройства используются интенсивнее других. Если проблема заключается в этом, можно перенести часть процессов на другие устройства, чтобы снизить нагрузку на отдельный диск. Также отчет может показать, на какой из дисков приходится наибольшее количество операций чтения/записи, что поможет определить устройство, которое следует заменить на более быстрое. И наоборот, можно выяснить, какие устройства, загружены недостаточно. Например, если оказывается, что обращений к новому сверхбыстрому диску производится немного, наверное, вы не настроили процессы на использование именно этого диска. С другой стороны, не исключено, что программа активно использует кэширование, поэтому операции ввода-вывода выполняются редко.

Команда sar

Команда sar предоставляет очень много возможностей. Она может отображать самую разную информацию о системе и записывать данные в течение указанного промежутка времени, а количество возможных вариантов настройки делает эти задачи не такими простыми. Обратитесь к соответствующей документации, чтобы убедиться, что параметры заданы верно. Как большинство подобных команд, команду sar можно настроить для генерации отчетов через определенные периоды времени.

Команда sar также отображает информацию об использовании процессора, оперативной памяти, кэша и множество других сведений, аналогичных данным, предоставляемым другими командами. Некоторые администраторы настраивают периодическое выполнение команды sar, чтобы собирать оценочные данные о системе. Полный обзор возможностей этой команды выходит за рамки нашей книги. Более полную информацию см. в книге System Performance Tuning (Настройка производительности системы), Жан-Пауло Д. Мусумеци (Gian-Paolo D. Musumeci), Майк Лукидс (Mike Loukides) (O'Reilly, http://oreilly.com/catalog/9780596002848/).

В этом разделе мы покажем, как использовать команду sar для получения информации об использовании диска. Для этого мы объединим данные о скорости ввода-вывода, пространстве подкачки, статистике подкачки и использовании блочных устройств. На рис. 7-9 показан пример использования команды sar для отображения сведений об использовании дисков.

Предоставленный отчет содержит так много информации, что на первый взгляд она кажется избыточной. Обратите внимание на первый раздел за заголовком. Это информация о подкачке, по которой можно судить о производительности подсистемы подкачки. Ниже располагаются данные о скорости ввода-вывода и пространстве подкачки, а затем список устройств с соответствующей информацией. В заключительной части отчета представлены средние величины по всем выводимым параметрам.

В отчете о подкачке указана скорость обмена страницами с оперативной памятью, число отказов предоставления страницы, не требующих доступа к

диску, в секунду, число более серьезных отказов, требующих доступа к диску, а также дополнительная информация о страничном обмене. Эта информация может оказаться полезной, если число отказов (особенно серьезных) велико, что может говорить о том, что запущено слишком много процессов. Большое число серьезных отказов может приводить к проблемам с использованием диска. Т. е., если это значение очень велико и диск используется интенсивно, возможно, дело не в плохой производительности диска, а в ошибках приложения или операционной системы.

Отчет о передаче данных подсистемой ввода-вывода включает число транзакций в секунду (tps), число запросов на чтение и запись, а также общее количество прочитанных и записанных блоков. В приведенном примере подсистема ввода-вывода не задействована, но очень загружен процессор. Это показатель хорошо работающей системы. Когда значения, относящиеся к вводу-выводу, очень высоки, вероятно, один или несколько процессов оказались привязанными к вводу-выводу. Если говорить о MySQL, то подобная ситуация может возникнуть из-за запроса, выполняющего множество произвольных обращений к диску, или таблиц, размещенных на фрагментированном диске.

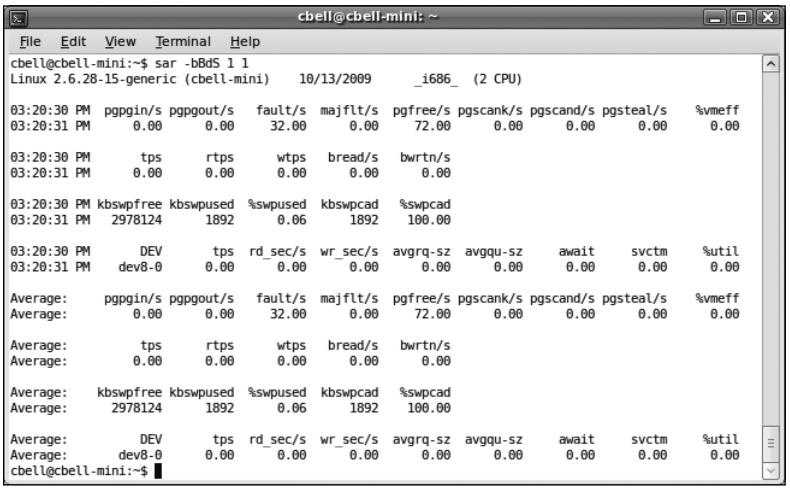


Рис. 7-9. Выходные данные команды sar, относящиеся к использованию диска

Отчет о пространстве подкачки показывает доступный объем пространства подкачки, сколько и какой процент пространства используется, а также насколько используется кэш-память. Эта информация может быть полезной для обнаружения проблем, связанных с выгрузкой на диск слишком большого числа процессов. Как и другие отчеты, этот отчет помогает выяснить, связана ли проблема с дисками и другими устройствами, с оперативной памятью или слишком большим числом процессов.

Отчет о блочных устройствах (любых частях системы, перемещающих данные блоками, таких как диск, оперативная память и т. д.) показывает

частоту передачи (tps), количество операций чтения и записи в секунду и среднее время ожидания. Эта информация может быть полезной для диагностики проблем блочных устройств. Если эти значения очень велики (в приведенном примере, наоборот, активность устройств практически отсутствует), возможно, пропускная способность устройств используется полностью. Однако следует сопоставить эту информацию с остальными отчетами, чтобы исключить варианты с пробуксовкой, большим количеством процессов или нехваткой оперативной памяти (а также комбинации этих проблем),

Такой сводный отчет полезен для выяснения причин проблем с использованием диска. Если отчет о подкачке показывает необычно интенсивную подкачку или большое число отказов, это может означать, что выполняется слишком много приложений или не хватает оперативной памяти. Однако, если эти значения низкие или средние, проверьте пространство подкачки. Если с ним все в порядке, ищите аномалии в отчете об устройствах.

Анализатор использования дисков

Помимо утилит операционных систем существует GUI-приложение Disk Usage Analyzer, разработанное в рамках проекта GNOME. Этот инструмент предоставляет подробный отчет о функционировании запоминающих устройств в графическом виде. Эта утилита входит в большинство дистрибутивов Linux.

На рис. 7-10 приведен пример отчета Анализатора использования дисков.

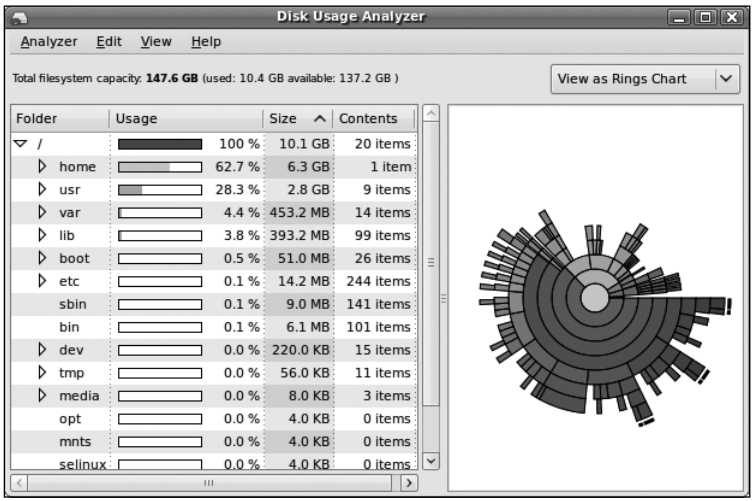


Рис. 7-10. Анализатор использования дисков

По сути, в этом отчете показана работа устройств и систем подкачки и страничного обмена. Конечно, если система интенсивно выгружает и загружает большое число процессов, использование диска будет аномальным. Поэтому полезно собрать такие характеристики в общем отчете.

Диагностика проблем диска может оказаться сложной задачей, и подробную информацию для этого предоставляют лишь немногие команды, описанные выше. Однако в некоторых операционных системах имеются специализированные инструменты, предоставляющие подробные отчеты об использовании диска. Не забывайте, что выяснить объем доступного пространства, получить список смонтированных дисков, определить их файловые системы и т.д. можно с помощью общих команд, таких как ls, df и fdisk. Полный список и описание команд для работы с дисками ищите в документации по вашей ОС.

Команда vmstat, о которой мы расскажем далее в этой главе, также предоставляет подобную информацию. Для получения данных в текстовом виде выполните команду vmstat –d.

Сетевая активность

Для диагностики проблем сети могут потребоваться дополнительные знания об оборудовании и сетевых протоколах. Подробная диагностика обычно выполняется специалистами по сетям, но есть две команды, с помощью которых администратор MySQL может дать предварительное заключение по ситуации.

Команда netstat

Команда netstat позволяет просматривать сетевые подключения, таблицы маршрутизации, сведения об интерфейсах и дополнительную информацию о сети. Эта команда предоставляет много данных, помогающих специалисту диагностировать и устранять сложные проблемы с сетью. Однако они окажутся полезными и для оценки сетевого трафика, и для определения наиболее интенсивно используемых интерфейсов. На рис. 7-11 показан пример отчета обо всех сетевых интерфейсах с указанием объема данных, передаваемых по каждому из них.

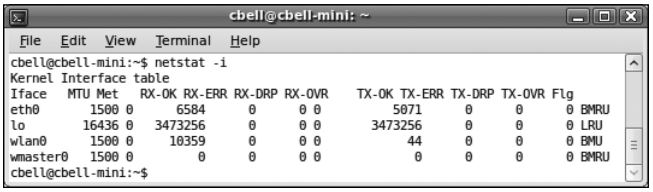


Рис. 7-11. Команда netstat

В системах с несколькими сетевыми интерфейсами такие данные могут быть полезными для выявления чрезмерно загруженных интерфейсов и интерфейсов, которые не должны быть активными.

Команда ifconfig

Команда ifconfig — основной инструмент для диагностики сетей, отображающий список сетевых интерфейсов системы с информацией о состоянии

и настройках каждого из интерфейсов. Пример выходных данных команды ifconfig приведен на рис. 7-12.

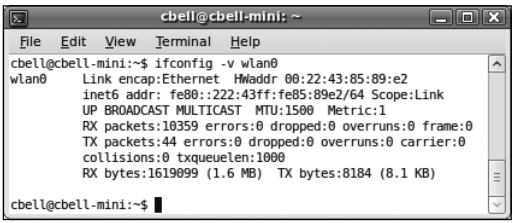


Рис. 7-12. Команда ifconfig

Обратите внимание на то, что показаны как включенные, так и отключенные интерфейсы, а также информация об их настройках. Это позволяет определить конфигурацию интерфейса и, например, выяснить, что данные проходят не через сверхбыстрый адаптер Ethernet, а через значительно более медленный сетевой интерфейс. Проблемы в сетях гораздо чаще связаны не с трафиком, а с выбором и настройкой сетевого интерфейса.

Если вы получили все описанные здесь отчеты, но не справились с проблемой самостоятельно, эти отчеты помогут специалисту по сетям быстрее разобраться в ситуации. Как только вы исключите процессы, занимающие слишком много пропускной способности сети, и выявите рабочий сетевой интерфейс, специалисту по сетям останется только настроить этот интерфейс для получения оптимальной производительности.

Общая информация о системе

Наряду с командами, ориентированными на ту или иную подсистему, описанными выше, а также командами для получения статистических отчетов, в Linux и Unix имеются дополнительные команды, предоставляющие общую информацию о системе. К таким командам относятся uptime и vmstat.

Команда uptime

Команда uptime отображает время работы системы, текущее время, количество использовавших систему пользователей (вошедших в систему), а также среднюю нагрузку за последние одну, пять и пятнадцать минут. Пример выходных данных команды uptime приведен на рис. 7-13.

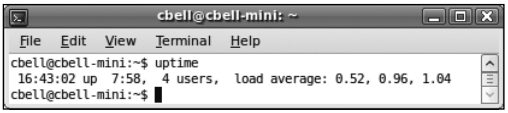


Рис. 7-13. Команда uptime

Эта информация поможет оценить среднюю производительность системы за последнее время. Средняя нагрузка вычисляется для процессов в активном состоянии (не ожидающих ввода-вывода или процессора).

Следовательно, возможности команды uptime ограничены, когда речь идет о диагностике проблем с производительностью, но с ее помощью можно оценить общую работоспособность системы.

Команда vmstat

Команда vmstat представляет собой инструмент для получения самых разных отчетов: о процессах, оперативной памяти, подсистеме подкачки, блочном вводе-выводе, дисках и активности процессора. Иногда решение проблем производительности начинают именно с нее. Если значения отдельных показателей окажутся высокими, используйте соответствующие команды, описанные в этой главе, для получения более подробной информации.

На рис. 7-14 показан пример выполнения команды vmstat в системе с низкой нагрузкой.
Здесь приводятся данные о количестве процессов, где r — число ожидающих выполнения, а b — число находящихся в непрерываемом состоянии. Следующий набор столбцов содержит сводные данные о пространстве подкачки, в том числе объем памяти, подкачанной с диска (si), и объем памяти, выгруженной на диск (so). В следующей секции приведены отчеты по вводу-выводу для блоков, отправленных (bi) и полученных (bo) с блочного устройства. Далее показано число прерываний в секунду (in), число переключений контекста в секунду (cs), время выполнения процессов в пространстве пользователя (us), время выполнения процессов в пространстве ядра (sy), время бездействия (id) и время ожидания ввода-вывода (wa). Время указывается в секундах.

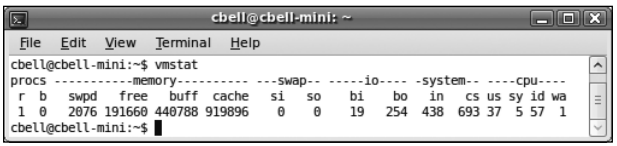


Рис. 7-14. Команда vmstat

Команда vmstat имеет и другие параметры. Подробнее см. в документации по ОС.

Автоматизация мониторинга при помощи планировщика cron

Возможно, самым важным из обсуждаемых здесь инструментов является cron. Как говорилось в разделе «Планирование заданий в Unix», при помощи cron можно запускать процессы в указанное время. Это позволяет выполнять команды и сохранять их выходные данные для последующего анализа. Преимуществом такого подхода является возможность регулярно делать моментальные снимки системы. На основе полученных данных можно рассчитывать средние значения параметров системы и использовать их в

качестве эталонных, с которыми вы сможете сверяться при возникновении проблем. Это важно, так как позволяет быстро определять, что изменилось, экономя много времени при диагностике проблем производительности.

Если выполнять мониторинг производительности ежедневно, а затем анализировать результаты и сравнивать их с эталонными, можно обнаруживать проблемы еще до того, как пользователи начнут жаловаться. Собственно, это и есть основная цель активного мониторинга.

Мониторинг Mac OS X

Поскольку в основе операционной системы Mac OS X лежит ядро Unix Mac, большинство описанных выше команд доступны и в этой ОС. Однако имеются и средства, специфичные для Mac. К ним относятся следующие инструменты администрирования с графическим интерфейсом:

- системный профайлер;
- приложение Console;
- монитор активности.

В этом разделе мы представим обзор перечисленных инструментов с точки зрения мониторинга Mac OS X. Эти инструменты являются основными средствами мониторинга и создания отчетов в Mac OS X. Они добротно разработаны, в лучших традициях Mac, и имеют удобный графический интерфейс. Через графический интерфейс можно просматривать даже отчеты, берущие информацию из файлов. Каждый из этих инструментов играет очень важную роль и может быть очень полезным при диагностике проблем с производительностью в Mac OS.

Системный профайлер

Системный профайлер позволяет делать моментальные снимки состояния системы. Он предоставляет очень много подробных данных почти обо всех частях системы, включая все оборудование, сеть и установленные приложения (см. рис. 7-15).

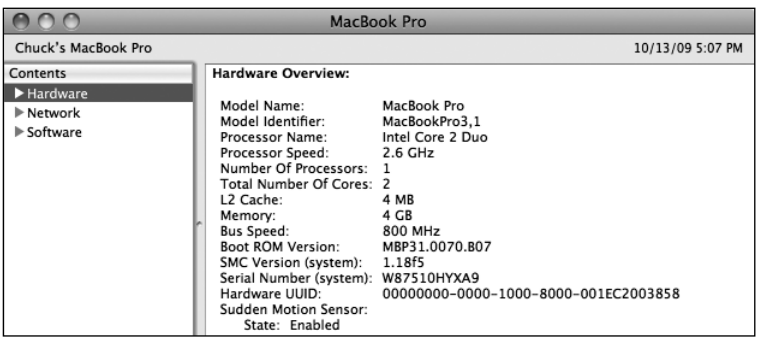


Рис. 7-15. Системный профайлер

Системный профайлер находится в папке *Applications/Utilities*. Его можно также запустить через Spotlight. Как показано на рис. 7-15, интерфейс Системного профайлера включает панель с древовидным списком (слева) и панель с подробной информацией (справа). При помощи списка можно переходить к различным компонентам системы.

Для тех, кто предпочитает отчеты в консольном режиме, Системный профайлер имеет эквивалентное приложение командной строки /usr/sbin/system_profiler, имеющее множество параметров, позволяющих настраивать выходные данные. Для получения более подробной информации откройте терминал и наберите man system_profiler.

Если раскрыть ветвь дерева Hardware, будет показан список всего оборудования системы. Например, чтобы посмотреть, какого типа память установлена в системе, щелкните на элементе Memory в ветви Hardware.

Системный профайлер предоставляет сетевые отчеты, которые мы видели в Linux. Чтобы получить общий отчет обо всех сетевых интерфейсах системы, раскройте ветвь дерева Network. Выберите один из сетевых интерфейсов в древовидном списке или на панели информации и увидите ту же информацию (и даже больше), которую предоставляют команды Linux и Unix. Также можно увидеть сведения о брандмауэрах, определенных расположениях и даже томах, к которым открыт доступ по сети.

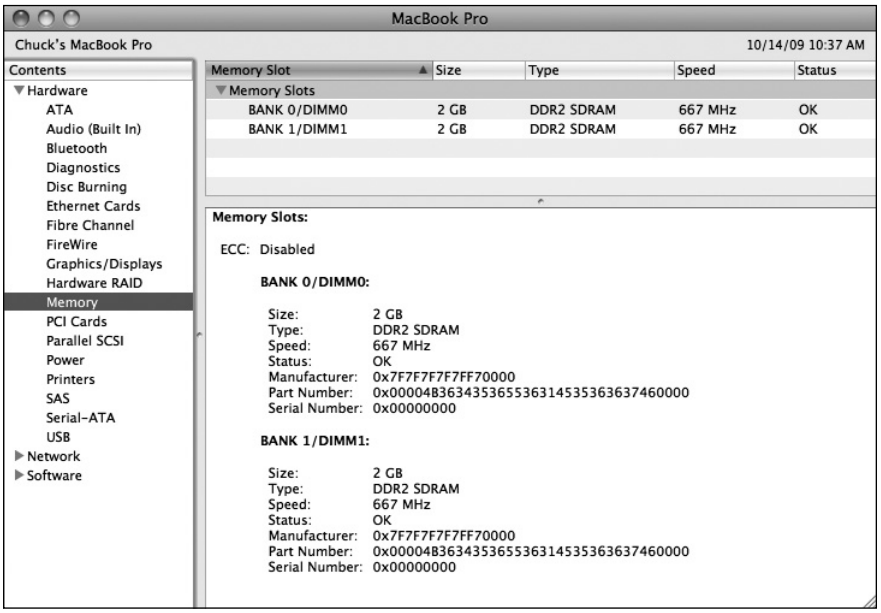


Рис. 7-16. Отчет системного профайлера о памяти

Еще один очень полезный отчет содержит сведения о приложениях, установленных в системе. Раскройте ветвь Software и выберите узел Applications report, чтобы получить список всех приложений системы, включающий название, версию, время обновления, разрядность (32 или 64) и тип приложе-

ния (например, является ли оно универсальным или исходным двоичным версии Intel). Последняя характеристика очень важна. Например, можно ожидать, что универсальный двоичный код будет работать медленнее, чем двоичный код Intel. Полезно знать это заранее, чтобы понять, какой производительности следует ожидать.

На рис. 7-16 показан пример такого отчета.

Как видите, отчет весьма подробный. Можно узнать, сколько плат памяти установлено, какова их скорость и даже код производителя и номер детали.

Мы называем каждую панель с информацией *отчетом*, так как это, по сути, подробный отчет об определенной категории компонентов. Иногда отчетом называют все данные в целом, что неправильно, и мы считаем, что лучше называть все данные набором отчетов.

Если вас впечатлили возможности этого инструмента, можете поэкспериментировать с древовидным списком, чтобы получить больше информации о своей системе. Там можно найти почти любую информацию о ней.

Системный профайлер может быть очень полезным при диагностике проблем. Если вы обратитесь за помощью, представителям AppleCare и техникам Apple могут не раз понадобиться отчеты о системе. Чтобы сгенерировать отчет Системного профайлера, выполните команду File | Save. Будет создан файл XML, который смогут использовать специалисты из Apple. Можно также экспортировать отчет в RTF, используя команду File | Export. Кроме того, можно распечатать отчет, предварительно сохранив его как файл PDF.

При помощи меню View можно изменять степень подробности отчета. Доступны варианты *Mini*, *Basic* и *Full* — от минимального до полного набора данных. Специалисты из Apple обычно просят представить полный отчет.

Отчет Системного профайлера — лучший способ определить, что происходит в системе, не углубляясь во внутренности системы. Его следует использовать в первую очередь, когда требуется определить конфигурацию системы.

Приложение Console

Приложение Console отображает файлы журналов системы. Найти его можно в папке */Applications/Utilities* или через Spotlight. В отличие от Системного профайлера, это приложение предоставляет не только дамп данных, но и возможность искать в журналах важную информацию. При диагностике проблем может потребоваться просмотреть сообщения из журналов, содержащие информацию о событиях. На рис. 7-17 показан снимок экрана этого приложения.

После запуска приложение считывает все системные журналы и разделяет сообщения из них по категориям диагностических сообщений. Как видно на рис. 7-17, в левой части находится панель поиска, а в правой — секция просмотра журнала. В древовидном списке Files можно выбирать отдельные

файлы журналов, чтобы просматривать их содержимое. Перечислим файлы журналов:

- *~/Library/Logs* — хранит все сообщения, относящиеся к пользовательским приложениям. Ищите здесь сообщения о приложениях, завершившихся с ошибкой после входа в систему, информацию об активности iDisk и других задачах, имеющих отношение к пользователям;
- */Library/Logs* — хранит все системные сообщения. Ищите здесь сгенерированную на уровне системы информацию о сбоях и других необычных событиях;
- */private/var/log* — хранит все сообщения, относящиеся к процессам Unix BSD. Ищите здесь информацию о системных демонах и утилите BSD.

Журналы — это текстовые файлы. Данные всегда дописываются в конец, никогда — в середину, и редко удаляются.

Наиболее интересная функция приложения Console — возможность поиска. Можно создавать отчеты, содержащие сообщения с определенной фразой или ключевым словом, и просматривать их позже. Чтобы запустить новый поиск, выберите команду меню File | New Database Search. Появится общий конструктор запросов, в котором можно указать критерии поиска. Когда закончите, можете назвать и сохранить отчет для дальнейшей обработки. Это может быть очень полезным для наблюдения за приложениями, с которыми возникают проблемы.

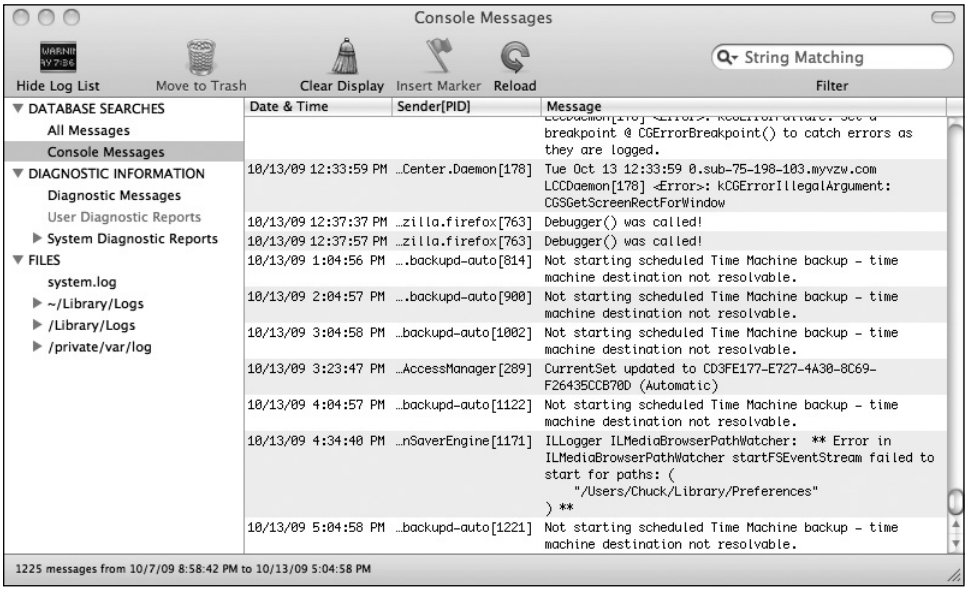


Рис. 7-17. Приложение Console

Еще одна очень полезная функция — возможность отмечать место в журнале, указав текущее время и дату. Это позволяет узнать, когда вы в послед-

ний раз заглядывали в журнал. Бывает так, что вы находите интересные сообщения в различных местах журналов, которые нужно просмотреть позже, но не знаете, где их нашли, или на каком месте закончили просматривать журнал. В таких случаях очень помогает возможность делать пометки в журнале. Чтобы сделать отметку, выделите место в файле и щелкните на кнопке Mark на панели инструментов.

Хотя данные отчетов являются статическим мгновенным снимком журналов, полученным при запуске, и все отчеты ограничены этим снимком, можно также устанавливать оповещения о новых сообщениях, появляющихся в журналах. Чтобы включить уведомления, выберите команду Console | Preferences. При появлении уведомления в секции Dock будет появляться прыгающий значок или приложение Console будет выводиться на передний план после некоторой задержки.

Приложение Console может быть очень полезным для наблюдения за работой различных компонентов системы путем мониторинга происходящих событий и поиска ошибок приложений и оборудования. При возникновении проблем с производительностью или других «проблемных» событий проверяйте журналы на предмет информации о соответствующем приложении или событии. Иногда решение проблемы можно найти в сообщении, сгенерированном самим приложением.

Монитор активности

В отличие от статической природы описанных выше утилит, Монитор активности (Activity Monitor) является динамическим инструментом, предоставляющим информацию о системе в процессе выполнения. Большой объем данных, необходимый для решения проблем производительности, можно получить при помощи Монитора активности. Здесь можно найти ту же информацию, что предоставляется всеми утилитами Linux и Unix: сведения о процессоре, памяти, активности дисков, использовании дисков и сетевых интерфейсов.

К примеру, с помощью Монитора активности можно выяснить, какие процессы запущены и сколько используют памяти и ресурсов процессора. В этом случае Монитор активности предоставляет те же сведения, что и команда top в Linux.

Сведения о процессоре содержат полезные данные, такие как процент времени, потраченного на выполнение в пространстве пользователя (время пользователя), процент времени в пространстве системы (время системы) и процент времени простоя. Здесь же показано число запущенных потоков и процессов и приведен цветной график со сводными показателями по времени пользователя и системы. В сочетании с информацией, представленной в стиле команды top, этот инструмент может быть очень полезным для поиска причин проблем, связанных с процессами, привязанными к процессору.

На рис. 7-18 показано окно Монитора активности с отчетом о ресурсах процессора.

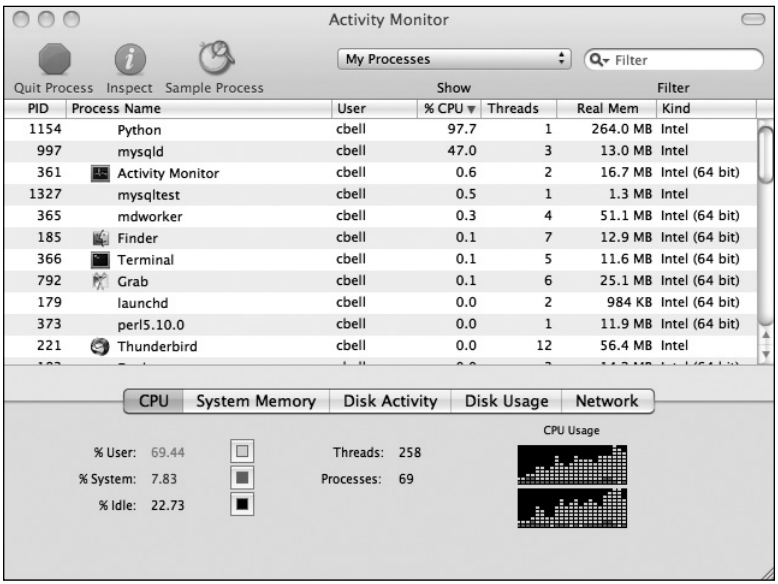


Рис. 7-18. Монитор активности, отчет о ресурсах процессора

Обратите внимание на Python-сценарий, который во время создания отчета занимал значительную часть процессорного времени. В этом случае в системе была запущена ветвь Bazaar в окне терминала. Монитор активности показывает, почему система замедлила работу во время ветвления дерева кода.

Чтобы получить дополнительную информацию о процессе, дважды щелкните на его имени. Можно также управляемо или принудительно завершить процесс. На рис. 7-19 показан пример диалога проверки процесса.

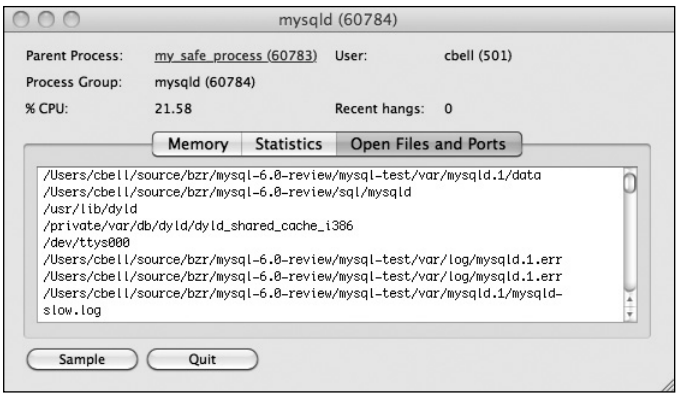


Рис. 7-19. Монитор активности, диалог проверки процесса

Список процессов можно экспортировать, выбрав команду File | Save. Можно сохранить список процессов как текстовый файл или как файл XML. Некоторые специалисты из Apple при диагностике проблем просят предоставить список процессов в дополнение к отчету Системного профайлера.

На странице System Memory (рис. 7-20) отображается информация о распределении памяти. Показано, сколько памяти свободно, сколько не может быть кэшировано и должно оставаться в ОЗУ (привязанная память), сколько используется и сколько неактивно. Из этого отчета можно быстро понять, что возникли проблемы с памятью.

На странице Disk Activity (рис. 7-21) отображается информация об активности всех дисков. В первом столбце показано общее число операций передачи данных с диска (чтение) и на диск (запись), а также указана производительность дисковых операций записи и чтения в секунду. В следующем столбце показан общий размер прочитанных и записанных данных и пропускная способность по каждому пункту. Кроме этого, показан цветной график операций чтения и записи за некоторое время.

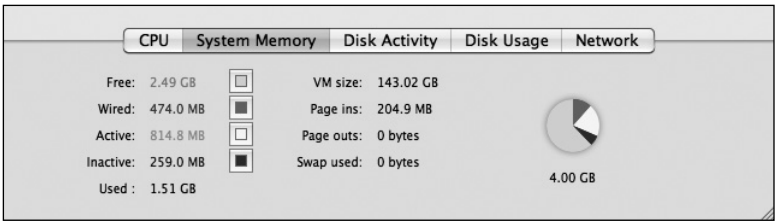


Рис. 7-20. Монитор активности, страница System Memory

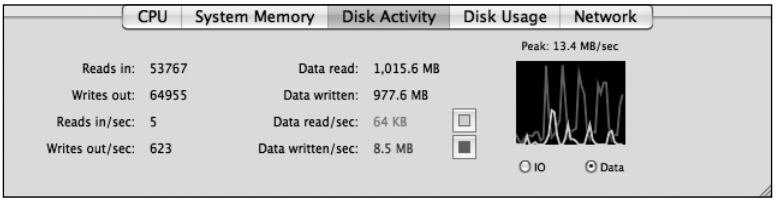


Рис. 7-21. Монитор активности, страница Disk Activity

Данные об активности дисков могут указать на то, что выполняется много обращений к диску, и что число операций записи и чтения необычно высоко (как и общий объем данных). Необычно высокое значение может говорить о том, что процессы следует запускать в разное время, чтобы они не конкурировали за доступ к диску, или следует добавить еще один диск для балансировки нагрузки.

На странице Disk Usage (рис. 7-22) отображается используемое и свободное пространство на каждом диске. Также показана цветная круговая диаграмма, позволяющая быстро оценить, насколько используется диск. Чтобы просмотреть информацию о другом диске, выберите его в раскрывающемся списке.

Эта страница позволяет наблюдать за свободным пространством дисков, чтобы знать, когда потребуется добавить новые диски и/или расширить разделы, чтобы появилось больше места.

На странице Network (рис. 7-23) содержится много информации о взаимодействии системы по сети. В первом столбце показано, сколько пакетов

прочитано или получено (входящие пакеты) и записано или отправлено (исходящие пакеты) по сети. Также приведены данные о производительности операций чтения и записи, измеряемые в пакетах в секундах. В следующем столбце показан размер прочитанных и записанных по сети данных, а также скорость передачи в каждом направлении. Цветной график показывает относительную производительность сети. Обратите внимание на пиковое значение на графике. Данные с этой страницы позволяют определить наличие процесса, потребляющего максимум пропускной способности сетевых интерфейсов системы.

Рис. 7-22. Монитор активности, страница Disk Usage

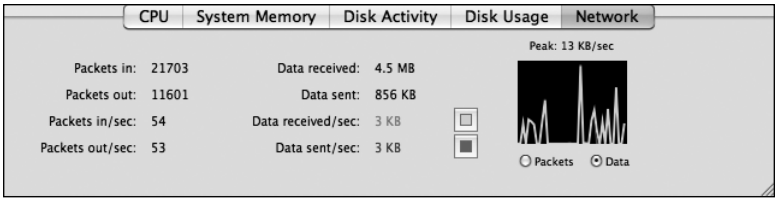


Рис. 7-23. Монитор активности, страница Network

В этом разделе мы познакомили вас со средствами мониторинга, доступными в Mac OS X. Это далеко не полное руководство, но приведенной информации достаточно, чтобы выполнять мониторинг систем Mac OS X. Подробные сведения о каждом из представленных приложений можно найти в документации Apple, доступ к которой можно получить при помощи меню Help соответствующего приложения.

Мониторинг Microsoft Windows

Windows имеет репутацию системы, в которой нет средств мониторинга. Кое-кто даже говорит, что средства мониторинга Windows алогичны. Однако уверяем вас, что трудность мониторинга систем Windows — миф. В действительности, в Windows имеются очень мощные инструменты, в том числе и планировщик для запуска задач. Можно получать мгновенные снимки производительности, просматривать ошибки в оснастке Просмотр событий (Event Viewer) и проводить мониторинг в реальном времени.

Рисунки, приведенные в этом разделе, получены на нескольких системах Windows Vista. Инструменты, доступные в Windows XP и новых версиях, включая Windows Server 2008 и Windows 7, особо не отличаются. Существуют некоторые отличия в способе доступа к инструментам в Windows 7. Эти различия указаны для каждого инструмента.

Администратору Windows доступно множество инструментов. Мы не будем рассматривать их все, а остановимся на тех, которые позволяют выполнять мониторинг Windows в реальном времени. Сначала рассмотрим некоторые эталонные инструменты для создания отчетов.

К наиболее популярным инструментам для диагностики и мониторинга производительности Windows относятся следующие:

- Индекс производительности Windows (Windows Experience Index);
- Отчет о работоспособности системы (System Health Report);
- Просмотр событий (Event Viewer);
- Диспетчер задач (Task Manager);
- Монитор стабильности системы (Reliability Monitor);
- Системный монитор (Performance Monitor).

Отличный источник информации о производительности, инструментах, приемах работы и документации по Windows — веб-сайт Microsoft Technet (*http://technet.microsoft.com/en-us/windows*).

Индекс производительности Windows

Если хотите быстро получить представление о том, как система функционирует по сравнению с ожидаемыми индексами производительности оборудования, рассчитанными Microsoft, можете запустить отчет Индекс производительности Windows.

Чтобы запустить этот отчет, щелкните на кнопке **Пуск** (Start) и выберите **Панель управления** | **Система и ее обслуживание** | **Счетчики и средства производительности** (Control Panel | System and Maintenance | Performance Information and Tools). Для продолжения необходимо принять условия контроля учетных записей пользователей (UAC).

Отчет о работоспособности системы можно также запустить из меню **Пуск** (Start). Щелкните на кнопке **Пуск** (Start), введите в поле поиска текст «производительность» («performance») и щелкните на ссылке **Счетчики и средства производительности** (Performance Information and Tools). Щелкните на ссылке **Дополнительные инструменты** (Advanced Tools), а затем на ссылке **Создать отчет о работоспособности системы** (Generate a system health report).

В Windows 7 отчет Windows Experience изменился. Он похож на отчеты из ранних версий Windows, но содержит больше информации, по которой можно судить о производительности системы.

Этот отчет запускается один раз после установки, но его можно пересоздавать, щелкая на кнопке **Повторить оценку** (Update My Score).

Отчет содержит информацию о пяти областях производительности системы: процессор, память, контроллер видео (графика), ускоритель видеографики (игровая графика) и основной жесткий диск. На рис. 7-24 показан пример отчета Индекс производительности Windows.

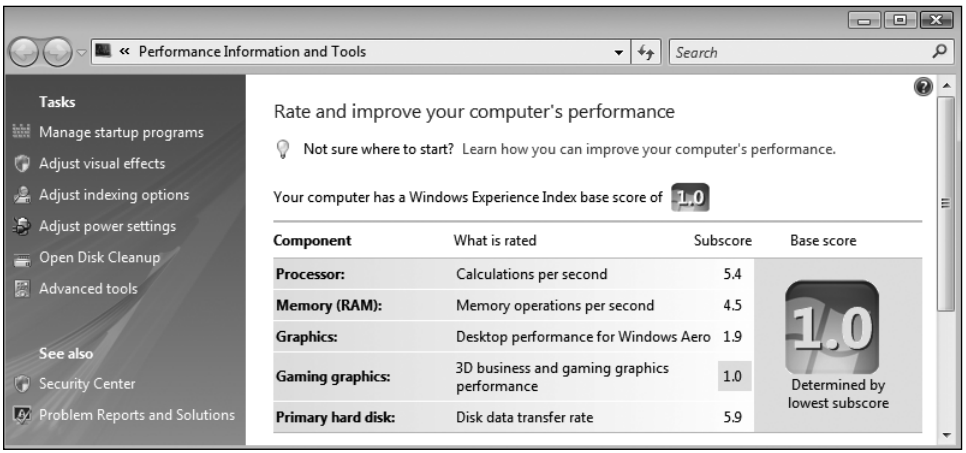


Рис. 7-24. Отчет Индекс производительности Windows

Одна из малоизвестных, но ценных возможностей отчета — ссылка **Рекомендации по повышению производительности компьютера** (Learn how you can improve your computer's performance), ведущая к списку рекомендаций по повышению каждого из оцениваемых показателей.

Этот отчет следует запускать и обновлять показатели каждый раз, когда меняется конфигурация системы. Это поможет выявлять случаи, когда изменения в конфигурации повлияли на производительность сервера.

Этот инструмент лучше всего использовать для получения общего впечатления о производительности системы, не анализируя множество характеристик. Низкие показатели в любой из категорий могут указывать на проблемы производительности. Например, отчет на рис. 7-24 показывает, что система имеет очень низкие показатели графики и игровой графики. Это не удивительно для системы, работающей как виртуальная машина или сервер без монитора, но должно насторожить человека, выложившего несколько тысяч долларов за высококлассную игровую систему.

Отчет о работоспособности системы

Одной из уникальных функций и диагностических улучшений в Windows Vista и Windows 7 является возможность генерации отчета, создающего мгновенный снимок всех приложений, оборудования и показателей производительности системы. Этот отчет аналогичен Системному профайлеру в Mas OS X, но содержит также счетчики производительности.

Чтобы запустить отчет о работоспособности системы, нажмите кнопку **Пуск** (Start) и выберите **Панель управления** | **Система и ее обслуживание** | **Счетчики и средства производительности** (Control Panel | System and Maintenance | Performance Information and Tools). Затем перейдите в раздел **Дополнительные инструменты** (Advanced Tools) и щелкните на ссылке

Создать отчет о работоспособности системы (Generate a system health report). Необходимо подтвердить запрос UAC.

Отчет о работоспособности системы можно также запустить из меню **Пуск** (Start). Щелкните на кнопке **Пуск** (Start), введите в поле поиска текст «производительность» («performance») и щелкните на ссылке **Счетчики и средства производительности** (Performance Information and Tools). Щелкните на ссылке **Дополнительные инструменты** (Advanced Tools), а затем на ссылке **Создать отчет о работоспособности системы** (Generate a system health report). Кроме этого, можно ввести в поле поиска текст «отчет о работоспособности системы» («system health report»), щелкнуть на появившейся ссылке и подтвердить сообщение UAC. На рис. 7-25 показан пример отчета о работоспособности системы.

Этот отчет содержит все: полный список оборудования, приложений и других компонентов системы. Обратите внимание на то, что отчет разделен на разделы, которые можно сворачивать и разворачивать для удобства просмотра. Ниже вкратце описан каждый из разделов.

- **Отчет о диагностике системы** (System Diagnostics Report) — название системы и дата создания отчета.
- **Результаты диагностики** (Diagnostic Results) — предупреждения, сгенерированные во время создания отчета, указывающие на потенциальные проблемы. Также сюда включены краткие сведения о производительности системы во время создания отчета.
- **Конфигурация программного обеспечения** (Software Configuration) — полный список приложений, установленных в системе, включающий параметры безопасности системы, системные службы и автоматически запускаемые программы.
- **Конфигурация оборудования** (Hardware Configuration) — список важных метаданных о дисках, счетчиков производительности процессора, параметров BIOS и устройств.
- **ЦП** (CPU) — список процессов, выполняющихся во время создания отчетов, и метаданные о системных компонентах и службах.
- **Сеть** (Network) — метаданные о сетевых интерфейсах и протоколах системы.
- **Диск** (Disk) — счетчики производительности и метаданные обо всех дисковых устройствах.
- **Память** (Memory) — счетчики производительности памяти, включающие список процессов и сведения об использовании памяти.
- **Отчет о статистике** (Report Statistics) — общая информация о системе на момент создания отчета, например, скорость процессора и объем установленной памяти.

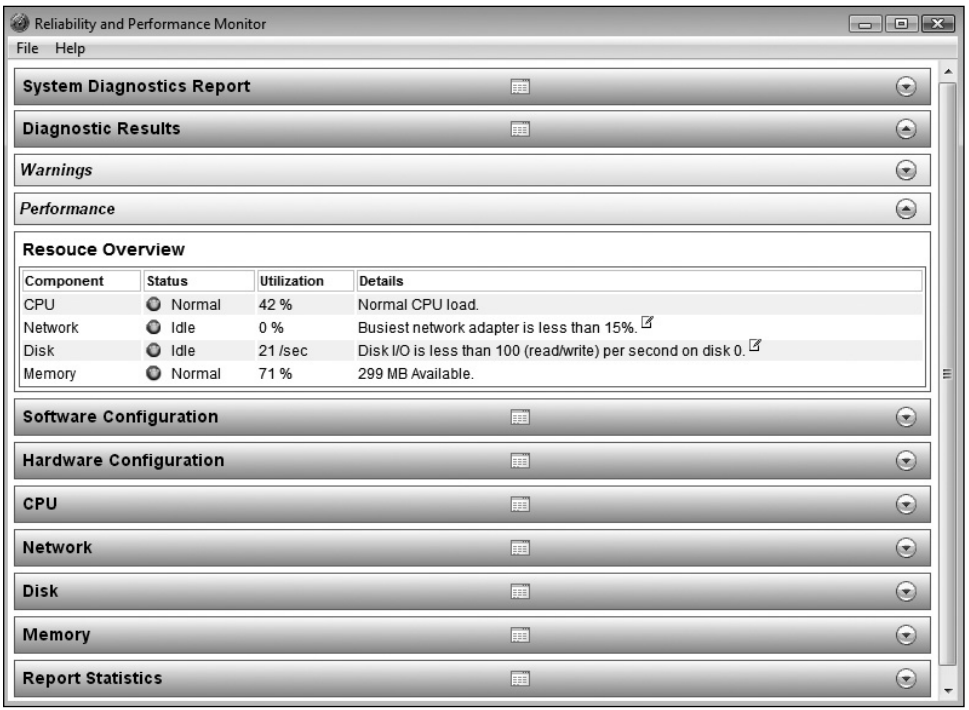


Рис. 7-25. Отчет о работоспособности системы

Отчет о работоспособности системы — ключ к быстрому пониманию того, как настроена система, и какова ее производительность. Это статичный отчет, представляющий мгновенный снимок системы.

Разделы **Конфигурация оборудования** (Hardware Configuration), **ЦП** (CPU), **Сеть** (Network), **Диск** (Disk) и **Память** (Memory) содержат большой объем подробной информации, позволяющей многое узнать о системе.

Лучше всего эту утилиту использовать для проверки счетчиков производительности и для сохранения отчетов с дальнейшим их сравнением в случае возникновения проблем с производительностью. Чтобы сохранить отчет в формате HTML, выберите команду меню **Файл | Сохранить как** (File | Save As).

Сохраненный отчет можно использовать как эталон для оценки производительности системы. Если создать отчеты при низкой, средней и высокой нагрузке, можно понять, какой производительности системы следует ожидать. Важно определить ожидаемую производительность, так как это позволит узнать, работает ли система в пределах нормы или возникли проблемы производительности. Если система показывает необычное повышение нагрузки в то время, когда ожидается низкая нагрузка, пользователи могут начать жаловаться. Если имеются отчеты для сравнения, можно сэкономить много времени на выяснении причин медленной работы.

Просмотр событий

Утилита Просмотр событий (Event Viewer) показывает все записанные сообщения о событиях приложений, безопасности и системы. Это очень полезный источник информации о произошедших (или продолжающих происходить) событиях, который должен стать одним из основных инструментов для диагностики и мониторинга системы.

Эта утилита позволяет выполнять много задач. Например, можно создавать настраиваемые представления любого журнала, сохранять журналы для дальнейшей проверки, настраивать оповещения об определенных событиях. Мы остановимся на просмотре журналов. Подробнее о просмотре событий, созданию отчетов и настройке оповещений см. в справочных файлах Windows.

Чтобы запустить Просмотр событий, щелкните на кнопке **Пуск** (Start), щелкните правой кнопкой мыши на пункте меню **Компьютер** (Computer) и выберите из контекстного меню команду **Управление** (Manage). Для продолжения необходимо подтвердить сообщение UAC. Затем выберите узел **Просмотр событий** (Event Viewer) в левой части открывшегося окна. Можно также щелкнуть на кнопке **Пуск** (Start), ввести «просмотр событий» («event viewer») и нажать Enter.

По умолчанию окно утилиты имеет три секции. В левой части находится древовидный список пользовательских представлений, файлов журналов, журналов служб и приложений. Журналы отображаются в средней секции, а справа находятся команды меню **Действия** (Action). По умолчанию записи журналов отсортированы по убыванию даты и времени. Это позволяет видеть недавние события в первую очередь.

Представления утилиты Просмотр событий можно настраивать в соответствии с собственными предпочтениями. Можно даже группировать и сортировать события, щелкая на столбцах в заголовке журнала.

Раскройте ветвь **Журналы Windows** (Windows logs), чтобы просмотреть файлы журналов приложений, безопасности и системы (а также некоторые другие). На рис. 7-26 показано окно утилиты Просмотр событий с раскрытым деревом журналов.

Для просмотра и поиска доступны следующие журналы:

- **Приложение** (Application) — содержит все сообщения, сгенерированные пользовательскими приложениями и службами операционной системы. Проверяйте этот журнал при диагностике проблем с приложениями.
- **Безопасность** (Security) — содержит сообщения, относящиеся к доступу и привилегиям, а также сообщения о неудачных попытках доступа к любому защищенному объекту. Здесь можно искать причины проблем с приложениями, связанных с учетными данными.
- **Установка** (Setup) — содержит сообщения, относящиеся к установке приложений. Это лучший источник информации о сбоях при установке или удалении программ.

- **Система** (System) — содержит сообщения о драйверах устройств и компонентах Windows. Это, возможно, самый полезный набор журналов для диагностики проблем с устройствами и системой в целом. Здесь содержится информация обо всех устройствах, работающих на уровне системы.
- **Перенаправленные события** (Forwarded Events) — содержит сообщения, направленные с других компьютеров. Информацию о работе с удаленными журналами событий см. в документации Windows.

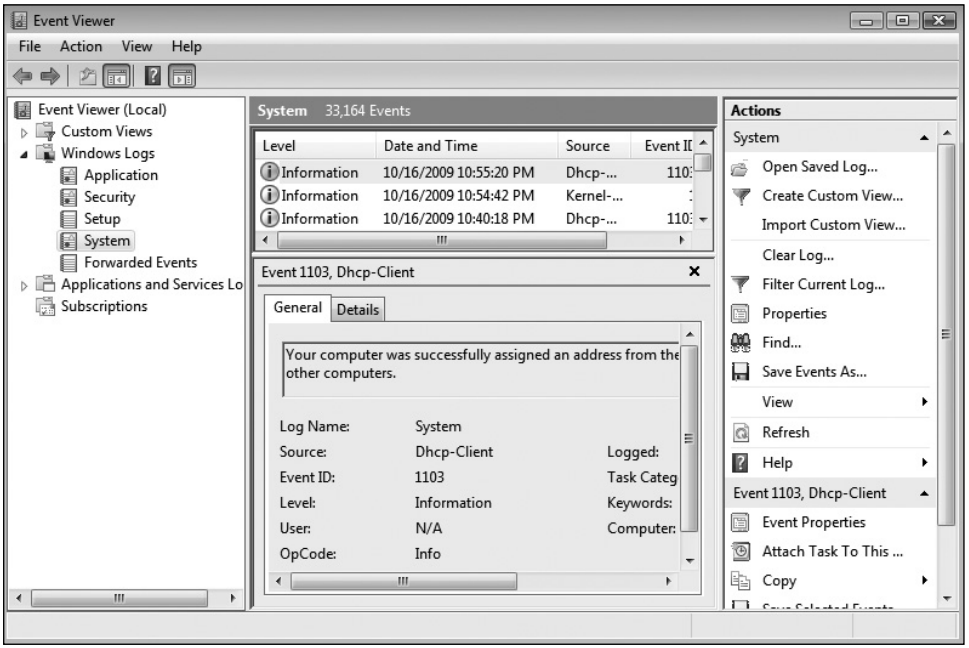


Рис. 7-26. Просмотр событий

Изучение этих журналов может быть сложной задачей, так как в них содержится много информации, интересной разработчикам, но непонятной для обычных смертных. Чтобы облегчить себе работу, можете выполнить поиск по журналам, выбрав в секции **Действия** (Actions) команду **Найти** (Find) и введя искомый текст. Например, решая вопросы с памятью, можете ввести текст «memory», чтобы в центральной секции отображались только те записи, которые содержат это слово.

Все сообщения журналов попадают в одну из трех категорий. Это относится и к пользовательским процессам, и к системным компонентам, и к приложениям.

- **Ошибка** (Error) — указывает на какой-то сбой, например, ошибку в процессе, нехватку памяти или системный сбой.
- **Предупреждение** (Warning) — указывает на менее серьезное происшествие или событие, на которое следует обратить внимание, например, на то, что осталось мало памяти или дискового пространства.

- **Сведения** (Information) — содержит информацию о событии. Обычно не указывает на проблему, но может предоставить дополнительные сведения, полезные при диагностике проблем, например, на то, что устройство USB отключено.

Для просмотра журнала откройте в левой части окна соответствующую ветвь дерева. Выберите событие, чтобы просмотреть информацию о нем. Сообщение отображается под записями журнала (см. рис. 7-26). Вкладка **Общие** (General) в нижней части центральной секции содержит общую информацию о сообщении: текст сообщения, время события, имя журнала, имя пользователя, запустившего процесс или приложение. На вкладке **Подробности** (Details) приведен отчет о записанных данных, который можно просмотреть в виде текста [**Понятное представление** (Friendly View)] или в формате XML [**Режим XML** (XML View)]. Эту информацию можно сохранить для дальнейшего просмотра. Режим XML полезен для работы с отчетом при помощи утилит, поддерживающих этот формат.

Монитор стабильности системы

Наиболее интересный инструмент мониторинга в Windows — Монитор стабильности (Reliability Monitor). Это специализированная утилита, строящая график на основе данных об ошибках и значительных событиях, влияющих на производительность.

Вертикальная ось представляет дни, горизонтальная — сводные показатели индексов производительности в эти дни. Если возникали ошибки или значительные события, на графике появляется красный знак X. Ниже находится набор списков, содержащих сведения об установке и удалении программ, сбоях приложений и оборудования, сбоев Windows и любых других сбоях.

Эта утилита очень полезна для наблюдения за производительностью системы в течение длительного времени. Она помогает диагностировать ситуации, когда программа или служба нормально работала в прошлом, но затем стала функционировать некорректно, или когда система начинает генерировать сообщения об ошибках. С помощью этого монитора можно выяснить, в какой день случилось первое подобное событие, и понять, как функционировала система, когда все было нормально.

Еще одно преимущество этой утилиты состоит в том, что она предоставляет набор ежедневных эталонных характеристик системы на длительном промежутке времени. Это помогает диагностировать проблемы, связанные с заменой драйверов устройств (это одно из проклятий администраторов Windows), которые могут оставаться незамеченными, пока производительность системы не снизится до предела.

Короче, Монитор стабильности позволяет вернуться в прошлое и посмотреть, как работала система. Что может быть лучше? Эту утилиту не надо включать, она работает автоматически, собирая большинство данных из журналов, и таким образом ведет хронологию записей о системе.

Известный источник проблем в Windows — подключение и настройка оборудования. Мы не будем обсуждать эту тему, так как для этого потребуется отдельная книга. Если у вас возникают проблемы с оборудованием и драйвером, можете обратиться к руководству *Microsoft Windows XP Inside Out*, Ed Bott и др. (Microsoft Press).

Запустить Монитор стабильности можно из строки поиска меню **Пуск** (Start) по ключевому слову «стабильность» («reliability»). Подтвердите сообщение UAC и выберите в древовидном списке слева узел **Монитор стабильности** (Reliability Monitor) (рис. 7-27).

Чтобы запустить Монитор стабильности в Windows 7, введите в строке поиска меню **Пуск** (Start) фразу «центр поддержки» («action center») и нажмите Enter. Затем в разделе **Обслуживание** (Maintenance) щелкните на ссылке **Показать журнал стабильности работы** (View reliability report). Этот отчет отличается от отчетов из предыдущих версий Windows, но предоставляет ту же информацию в более сжатом виде. Например, вместо раскрывающихся списков в новом Мониторе стабильности отображается один список с известными происшествиями.

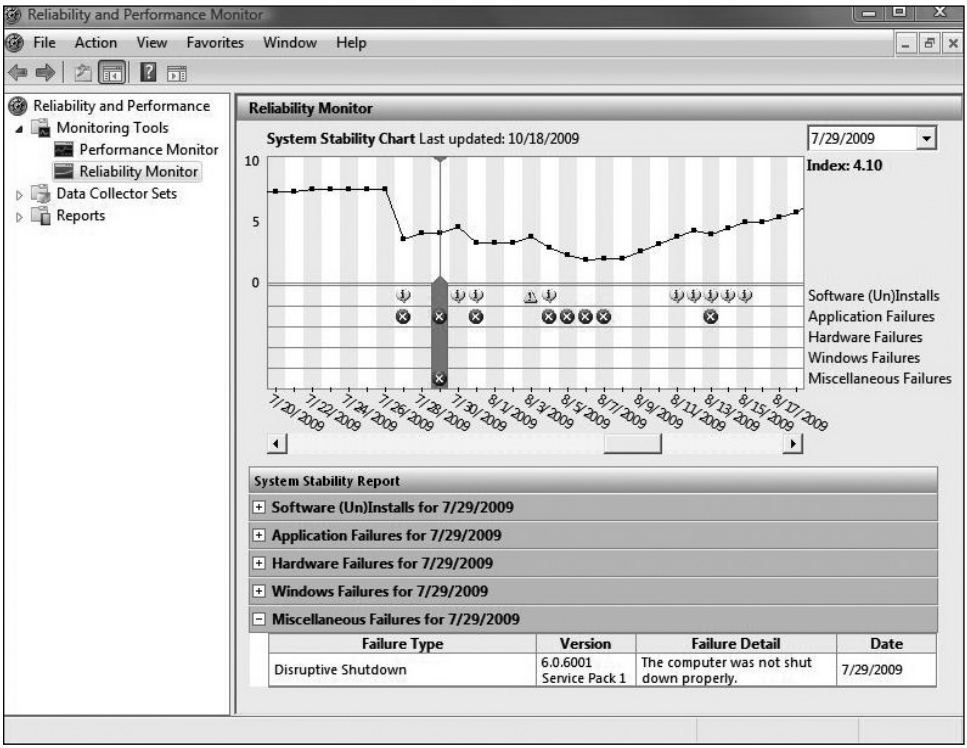


Рис. 7-27. Монитор стабильности системы

Диспетчер задач

Диспетчер задач (Task Manager) отображает динамический список выполняющихся процессов. Он появился в Windows давно и совершенствовался с выходом новых версий ОС.

Диспетчер задач представлен диалоговым окном с вкладками, содержащих списки запущенных приложений, процессов (этот список больше всего похож на выходные данные команды Linux top), активных служб, показателей производительности процессора, показателей производительности сети и список подключенных пользователей. В отличие от других отчетов, эта утилита генерирует данные динамически, периодически обновляя их. Это делает Диспетчер задач немного более полезным для наблюдения за системой в периоды низкой производительности.

Здесь отображаются те же данные, что и в отчете о работоспособности системы, но в гораздо более компактной форме и непрерывно обновляющиеся. В этих данных можно найти все необходимые показатели, необходимые для диагностики проблем производительности процессора, памяти, сети и процессов, занимающих много ресурсов. Как ни странно, отсутствует отчет о производительности дисков.

Одна из интересных особенностей Диспетчера задач — миниатюрный индикатор производительности в области уведомлений на панели задач, позволяющий наблюдать за пиками нагрузки, выполняя другую работу.

Динамические инструменты для наблюдения за производительностью требуют ресурсов и могут ухудшить работу системы, и без того имеющей плохую производительность.

Чтобы запустить Диспетчер задач, нажмите Ctrl+Shift+Esc или нажмите Ctrl+Alt+Del и выберите в меню соответствующую команду. На рис. 7-28 показано окно Диспетчера задач, отображающее список процессов.

Системный монитор

Системный монитор (Performance Monitor) — основной инструмент для наблюдения за производительностью системы Windows. Он позволяет выбирать ключевые показатели и строить графики по их значениям. Можно сохранять сеансы работы, чтобы в дальнейшем просматривать их и использовать в качестве эталонов.

Системный монитор включает практически любые показатели системы. Имеются счетчики для многих компонентов, имеющих отношение к основным аспектам производительности: процессору, памяти, дискам и сети. Эти показатели и счетчики разделены на множество категорий.

Чтобы запустить Системный монитор, в меню **Пуск** (Start) выберите **Панель управления** | **Система и ее обслуживание** | **Счетчики и средства производительности** (Control Panel | System and Maintenance | Performance Information and Tools). Затем перейдите в раздел **Дополнительные инструменты** (Advanced Tools) и щелкните на ссылке **Открыть системный монитор** (Open Reliability and Performance Monitor). Подтвердите сообщение UAC. В открывшемся окне выберите в древовидном списке узел **Системный монитор** (Performance Monitor).

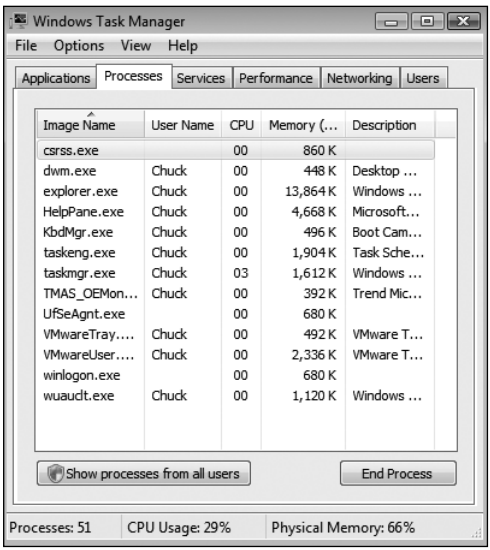


Рис. 7-28. Диспетчер задач

Можно также открыть меню **Пуск** (Start), ввести в строку поиска ключевое слово «стабильность» («reliability») и нажать Enter или щелкнуть на появившейся ссылке. Подтвердите сообщение UAC и выберите узел **Системный монитор** (Performance Monitor). Окно Системного монитора показано на рис. 7-29.

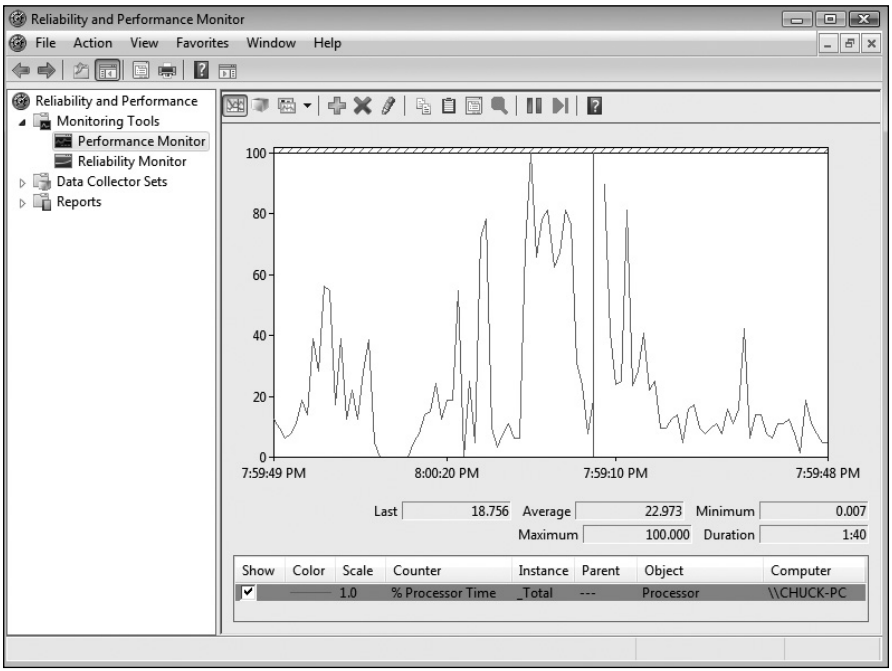


Рис. 7-29. Системный монитор

Microsoft предусмотрела два уровня показателей: объекты, предлагающие высокоуровневое представление таких категорий, как процессор или память, и счетчики, представляющие конкретные показатели. Таким образом, можно наблюдать за производительностью процессора в целом или следить за отдельными характеристиками, такими как процент времени простоя или число выполняющихся процессов пользователей. Эти объекты и счетчики можно добавлять на основной график, щелкая на кнопке с зеленым знаком плюса на панели инструментов. Откроется диалоговое окно, в котором можно выбрать нужные элементы из длинного списка. Чтобы добавить элемент на график, раскройте категорию в левой части окна и перетащите нужный объект в список справа.

На график можно добавить любое количество элементов. Оси изменятся соответствующим образом. Однако, если добавить слишком много элементов или если значения элементов слишком различаются, график может стать ненадежным ориентиром. Лучше всего наблюдать одновременно за небольшим числом связанных элементов (например, только за счетчиками памяти), чтобы получать от графика наибольшую пользу.

Полное описание возможностей Системного монитора выходит за рамки этой главы. Мы рекомендуем вам самостоятельно изучить дополнительные функции, такие как группы сборщиков данных и изменение параметров графика. На эту тему написано много хороших руководств.

Гибкость Системного монитора делает его лучшим выбором для формирования эталонных показателей и записи поведения системы на протяжении длительного времени. Его можно использовать и для диагностики в реальном времени.

Используя Системный монитор или Монитор стабильности системы, вы могли заметить редко описываемую функцию Обзор ресурсов (Resource Overview). Это стандартное представление Монитора надежности и производительности, предоставляющее четыре динамических графика производительности: процессор, диски, сеть и память. Под графиками расположены раскрывающиеся панели, содержащие информацию об этих категориях. Этот отчет является расширенной формой графиков Диспетчера задач и предоставляет еще один способ наблюдения и диагностики производительности систем Windows.

Это краткое введение в мониторинг систем Windows опровергает мнение о том, что выполнять мониторинг на платформе Windows сложно, и для этого недостаточно инструментов. Имеющиеся инструменты предоставляют широкие возможности и позволяют наблюдать за системными данными различными способами.

Мониторинг как профилактическое средство

Описанные выше приемы позволяют получить мгновенный снимок состояния системы. Однако, большинство специалистов сходятся во мнении о

том, что мониторинг — это обычно автоматически выполняемая задача по обнаружению аномалий в доступных данных. Если аномалия обнаружена, администратору (или группе администраторов) отправляется уведомление о проблеме. Это превращает задачу проверки состояния системы в задачу профилактики.

Существуют различные утилиты сторонних производителей, объединяющие задачи мониторинга, создания отчетов и оповещения и предоставляющие удобный интерфейс. Есть даже системы мониторинга и оповещений для целых инфраструктур. Например, система Nagios позволяет выполнять мониторинг всей ИТ-инфраструктуры и настраивать уведомления об аномалиях.

Кроме того, существуют системы мониторинга и оповещений, доступные как часть или дополнение ОС или СУБД. Одной из таких систем является Enterprise Monitor для MySQL (см. гл. 13).

Заключение

На тему настройки производительности и мониторинга безопасности написано множество руководств. Эта глава представляет собой введение в мониторинг системы. Хотя приведенный материал далеко не полный, он служит хорошим знакомством с инструментами, приемами и концепциями мониторинга производительности ОС и сервера. В следующей главе мы поговорим о мониторинге MySQL и приведем некоторые общие рекомендации по обеспечению максимальной производительности серверов MySQL.

— Джоэл!

Джоэл слишком хорошо знал и этот голос и эту интонацию: явно Саммерсон идет к нему с очередным заданием. Джоэл обернулся к двери как раз в тот момент, когда в нее входил его руководитель.

— Ты читал письмо Салли о том, что все работает так медленно?

Джоэл вспомнил, что Салли была среди тех, кто жаловался на плохую работу приложений. Он только что проверил простейшую версию причины сбоев, но памяти было достаточно, как и места на диске.

— Да, я как раз разбираюсь с этой проблемой...

— Сделай это в первую очередь. Отделу маркетинга надо срочно заканчивать квартальный план продаж. Дай мне знать, что выяснишь, — Саммерсон, кивнув, вышел.

Джоэл вздохнул и снова погрузился в отчеты об использовании ресурсов процессора, пытаясь придумать способ объяснить технические подробности человеку, ничего в технике не понимающему.

ГЛАВА 8

Мониторинг MySQL

Джоэла не покидало предчувствие, что сегодняшний день будет удачным. И правда, все шло хорошо: оценки производительности серверов были неплохи, да и пользователи особо не жаловались. Ему удалось перенастроить сервер, значительно повысив его эффективность. Только одно приложение все еще работало из рук вон плохо, но Джоэл был уверен, что дело не в оборудовании или операционной системе, а в плохо написанном запросе. Тем не менее, он отправил руководителю сообщение с описанием результатов своих поисков, не забыв упомянуть, что работает над оставшимися проблемами.

Услышав звук быстро приближающихся шагов, Джоэл инстинктивно взглянул на дверь, ожидая привычного появления своего босса, и был очень удивлен, когда тот прошел мимо, даже не кивнув.

Пожав плечами, Джоэл вернулся к чтению почты, и тут же появилось новое письмо с темой «ОЧЕНЬ СРОЧНО». Это было письмо от Саммерсона. У Джоэла перехватило дыхание, но он усилием воли расслабился и открыл сообщение. При чтении в голове звучал голос шефа:

«Джоэл, ты хорошо справился с теми отчетами. Мне особенно понравилось, что ты включил подробные сведения о производительности памяти и дисков. Сделай такой же отчет о сервере баз данных. И посмотри, что за проблема у разработчиков с запросом, каким — тебе скажет Сьюзан».

Глубоко вздохнув, Джоэл снова открыл свою любимую книгу, чтобы подробнее узнать о мониторинге СУБД.

— Надеюсь, эту задачу удастся разбить на шаги, — пробормотал он, понимая, что ему придется быстро разобраться с довольно сложной темой.

Вы уже понимаете, что такое мониторинг, и как поддерживать операционную систему в максимально эффективном состоянии. Но как узнать, работают ли серверы MySQL в максимально эффективном режиме? И, что еще важнее, как узнать, когда они работают неэффективно?

Эту главу мы начнем с рассмотрения мониторинга MySQL, а затем перейдем к обсуждению мониторинга и повышения производительности баз данных. В конце мы дадим советы по улучшению производительности баз данных.

Что такое производительность?

Прежде чем мы начнем говорить о производительности баз данных и давать советы по мониторингу и настройке сервера MySQL, важно определить, что мы подразумеваем под производительностью. В этой главе будем считать хорошей такую производительность системы, которая удовлетворяет пользователя, а плохой — не соответствующую ожиданиям пользователя. Обычно говорят, что производительность хорошая, когда пользователь не имеет претензий к времени отклика и скорости обработки данных. Хотя это не очень научное определение, сообразительные администраторы знают, что лучший показатель хорошей работы — удовлетворенность пользователей.

Это не значит, что мы никак не измеряем производительность. Наоборот, мы можем и должны измерять ее, чтобы знать, что, когда и как исправлять. Более того, если оценивать производительность регулярно, можно даже предугадывать, когда пользователи начнут проявлять недовольство. Пользователей не волнует, что вы снизили нагрузку на кэш на 3%, установив новый рекорд. Вы можете гордиться своими достижениями, но проценты и цифры не имеют значения, если смотреть с точки зрения пользователя, сидящего за клавиатурой.

В настройке производительности есть очень важные принципы, которым вы должны всегда следовать. Никогда не изменяйте параметры сервера, базы данных и механизма БД, если не имеете четкого плана и полного понимания выполняемой операции и ее последствий. Что еще важнее, никогда не вносите изменений, не отслеживая их результаты во времени. Вполне возможно, что производительность сервера ненадолго улучшится, но в долгосрочной перспективе эффект будет негативным. Наконец, всегда проверяйте информацию из нескольких источников, включая справочные материалы.

Мониторинг сервера MySQL

Управление серверами MySQL относится к мониторингу приложений, так как большинство параметров их производительности определяется в коде MySQL, а не операционной системы. Как мы уже говорили, мониторинг MySQL всегда следует выполнять вместе с мониторингом операционной системы, так как серверы MySQL очень чувствительны к проблемам производительности операционной системы.

В онлайновом руководстве MySQL Reference Manual мониторингу и повышению производительности посвящена целая глава с интригующим названием «Оптимизация» (см. *http://dev.mysql.com/doc/refman/5.5/en/optimization.html*). Чтобы не повторять то, что написано в этом отличном руководстве, мы обсудим общий подход к мониторингу сервера MySQL и рассмотрим некоторые доступные инструменты.

Этот раздел представляет собой введение в более подробные вопросы мониторинга MySQL. Сначала мы расскажем, как изменять и отслеживать по-

ведение системы, а затем обсудим, как выполнять мониторинг для и оценки производительности и диагностики проблем с ней. Также мы дадим рекомендации по диагностированию проблем производительности и рассмотрим мониторинг подуровня механизма БД (эта тема не очень подробно раскрывается в других источниках).

Управление производительностью в MySQL

Существует два механизма для управления и слежения за поведением сервера MySQL. Переменные сервера управляют поведением, а переменные состояния позволяют следить за конфигурацией и получать статистическую информацию о компонентах и производительности сервера.

Есть много переменных, с помощью которых можно настраивать сервер. Некоторые можно изменять только при загрузке (они называются параметрами загрузки и могут устанавливаться в файлах параметров). Другие можно устанавливать на глобальном уровне (для всех подключений), на уровне сеанса (для одного подключения) или на обоих уровнях.

Параметры переменных сеанса не сохраняются вне текущего подключения и сбрасываются при его закрытии.

Переменные сервера можно считывать при помощи следующих команд:

```
SHOW [GLOBAL | SESSION] VARIABLES;
```

Если переменные не статические (доступны не только для чтения), их значения можно изменять при помощи следующих команд (в одной строке можно указать несколько параметров, разделяя их запятыми):

```
SET [GLOBAL | SESSION] <variable_name> = <value>;
SET [@@global. | @@session. | @@]<variable_name> = <value>;
```

Ниже указаны команды для считывания переменных состояния. Первые две команды показывают значения всех локальных переменных или переменных сеанса (по умолчанию выбрана область действия в пределах сеанса). Третья команда показывает переменные, имеющие глобальные область действия.

```
SHOW STATUS;
SHOW SESSION STATUS;
SHOW GLOBAL STATUS;
```

О том, как и когда использовать эти команды, рассказывается в следующем разделе.

Мониторинг производительности

Мониторинг производительности MySQL выполняется при помощи указанных выше команд, а именно, путем изменения и чтения значений системных переменных и чтения значений переменных состояния. Команды

SHOW и SET — только два из возможных инструментов мониторинга серверов MySQL.

Существуют и другие инструменты для мониторинга серверов MySQL. Инструменты, входящие в стандартную комплектацию, несколько ограничены в том смысле, что работать с ними можно только из консоли. Сюда относятся специальные команды, которые можно выполнять из клиента MySQL (например, SHOW STATUS), и утилиты, запускаемые из командной строки (например, mysqladmin).

Клиент MySQL иногда называют монитором MySQL, но его не следует путать с инструментами мониторинга.

Есть и инструменты с графическим интерфейсом, немного облегчающие работу тем, кто предпочитает работать с визуальными средствами. Можно также скачать утилиты MySQL GUI, позволяющие выполнять мониторинг системы, управлять запросами и переносить данные из других СУБД.

Сначала мы рассмотрим команды SQL, а затем инструменты Administrator GUI и Query Browser. Также мы расскажем об одном из часто игнорируемых инструментов — журналах сервера.

Некоторые администраторы считают журналы сервера первым и основным инструментом для администрирования сервера. Хотя журналы сервера не так важны для мониторинга производительности, они могут оказать существенную помощь в решении проблем с производительностью.

Команды SQL

Все команды SQL, относящиеся к мониторингу, являются вариантами команды `SHOW`, отображающей внутреннюю информацию о системе и ее подсистемах. Существует много вариантов этой команды. Ниже приведены наиболее часто используемые для мониторинга серверов MySQL.

- SHOW INDEX FROM <*таблица*> Показывает статистику по количеству элементов индекса для указанной таблицы. Эти данные используются оптимизатором для оценки избирательности объединения. Эта команда может быть очень полезной при диагностировании плохо работающих запросов, в частности, для определения того, предусмотрено ли в запросе использование индексов.
- SHOW PLUGINS Показывает список всех известных подключаемых модулей. В список включается имя модуля и его текущее состояние. Механизмы хранилища в последних выпусках MySQL реализованы как подключаемые модули. Используйте эту команду для получения информации о текущих доступных подключаемых модулях и их состоянии.
- SHOW [*FULL*] PROCESSLIST Показывает информацию обо всех потоках (включая подключения), запущенных в системе. Эта команда напоминает команды ОС для работы с процессами. Отображаемые данные включают сведения о подключении, выполняемую команду, длитель-

ность ее выполнения и текущее состояние. Как и команды ОС, эта команда позволяет диагностировать плохой отклик (слишком много потоков), зомби-процесс (долго работающий или не отвечающий) и даже проблемы с подключением. Обнаружив плохо функционирующие или не отвечающие процессы, используйте команду KILL для их уничтожения. По умолчанию показываются процессы текущего пользователя. Чтобы отобразить все процессы, добавьте ключевое слово FULL.

Для просмотра всех процессов, запущенных в системе, необходима глобальная привилегия SUPER.

- SHOW [*GLOBAL* | *SESSION*] STATUS Отображает значения всех системных переменных. Обычно эта команда используется чаще других. Используйте эту команду для получения всей статистической информации, доступной на сервере. Объединяя эту команду с ключевым словом GLOBAL или SESSION, можно отображать только глобальную статистику или только статистику сеанса.
- SHOW TABLE [FROM *<имя_БД>*] STATUS Показывает подробную информацию о таблицах указанной базы данных. Отображается информация о механизме хранилища, сопоставлении и индексе, а также сведения о создании и статистика строк. Эту команду можно использовать вместе с командой SHOW INDEX для проверки таблиц при диагностировании запросов.
- SHOW [*GLOBAL* | *SESSION*] VARIABLES Отображает системные переменные. Обычно это параметры конфигурации сервера, которые не содержат статистической информации, но иногда необходимы, чтобы определить, изменилась ли текущая конфигурация и заданы ли определенные параметры. Некоторые переменные доступны только для чтения и могут изменяться только в файле конфигурации или командной строке при загрузке, тогда как другие можно изменять глобально или устанавливать локально. Объединяя эту команду с ключевым словом GLOBAL или SESSION, можно отображать только глобальные переменные или только переменные сеанса.

Ограничение выходных данных команд SHOW

Команды SHOW предоставляют очень широкие возможности. Однако часто они отображают слишком много информации. Особенно это относится к командам SHOW STATUS и SHOW VARIABLES.

Чтобы отображалось меньше информации, можно использовать выражение LIKE <образец>, позволяющее просматривать только те строки, которые соответствуют указанному образцу. Наиболее распространенный пример — использование выражения LIKE для просмотра только переменных из определенного подмножества, например, связанных с репликацией или ведением журналов. В выражении LIKE можно использовать стандартные элементы и символы шаблонов MySQL, как и в запросах SELECT.

Приведем пример отображения переменных состояния, в имени которых встречается текст «log»:

```
mysql> SHOW SESSION STATUS LIKE '%log%';

+--------------------------+-------+
| Variable name            | Value |
+--------------------------+-------+
| Binlog cache disk use    | 0     |
| Binlog cache use         | 0     |
| Com binlog               | 0     |
| Com purge bup log        | 0     |
| Com show binlog events   | 0     |
| Com show binlogs         | 0     |
| Com show engine logs     | 0     |
| Com show relaylog events | 0     |
| Tc log max pages used    | 0     |
| Tc log page size         | 0     |
| Tc log page waits        | 0     |
+--------------------------+-------+
11 rows in set (0.11 sec)
```

К механизмам БД имеют отношение следующие команды:

- SHOW ENGINE <*имя_механизма*> LOGS Показывает информацию о журнале для указанного механизма БД. Отображаемые сведения зависят от механизма БД и могут быть очень полезными для настройки. Некоторые механизмы хранилища не предоставляют такой информации.
- SHOW ENGINE <*имя_механизма*> STATUS Показывает информацию о состоянии указанного механизма БД. Отображаемые сведения зависят от механизма БД. Некоторые механизмы хранилища показывают больше или меньше информации, чем другие. Например, для механизма InnoDB выводятся десятки переменных состояния, тогда как для NDB всего несколько, а для MyISAM не отображается ничего. Эта команда является основным средством для просмотра статистической информации о механизмах БД и может быть очень полезной при настройке некоторых из них (например, InnoDB).

Старые синонимы команды SHOW ENGINE (SHOW <механизм> LOGS и SHOW <механизм> STATUS) больше не используются.

- SHOW ENGINES Показывает список всех известных механизмов БД для текущей версии MySQL и их состояний (т.е., включен ли этот механизм БД). Это может быть полезным для определения того, какой механизм БД использовать для конкретной базы данных или в репликации, когда требуется выяснить, доступен ли выбранный механизм БД на главном и подчиненном серверах.

К репликации MySQL имеют отношение следующие команды:

```
SHOW BINLOG EVENTS [IN '<файл журнала>'] [FROM <позиция>] [LIMIT [<смещение>,] <число строк >]
```

Показывает список событий в том виде, в каком они были записаны в двоичный журнал. Можно указать, какой файл журнала должен проверяться (если не указать его, будет использоваться текущий файл журнала), и ограничить выходные данные последними событиями с определенной позиции или указанным числом строк после смещения в файле. Это основная команда для диагностирования проблем репликации. Ее очень удобно применять, когда происходит событие, прерывающее репликацию или вызывающее ошибку во время репликации.

Если не использовать выражение LIMIT на сервере, длительное время записывающем события в журнал, можно получить очень много выходных данных. Когда требуется проверить большое число событий, лучше использовать утилиту mysqlbinlog.

- SHOW BINARY LOGS Отображает список двоичных журналов сервера. Используется для получения прошлых и текущих имен файлов журналов. Также отображается размер каждого файла. Эта команда тоже полезна для диагностирования проблем репликации, так как позволяет указать файл двоичного журнала для команды SHOW BINLOG EVENTS, тем самым снизив объем данных, которые нужно изучить, чтобы разобраться в проблеме. Синоним — команда SHOW MASTER LOGS.

  ```
  SHOW RELAYLOG EVENTS [IN '<файл журнала>'] [FROM <позиция>] [LIMIT [<смещение>,] <число строк>]
  ```

  Эта команда (доступна в MySQL 5.5.0) делает то же, что и команда SHOW BINLOG EVENTS, но с журналами ретрансляции на подчиненном сервере. Если не указать им файла журнала, будут показаны события из первого журнала ретрансляции. При запуске на главном сервере эта команда не имеет никакого эффекта.
- SHOW MASTER STATUS Показывает текущую конфигурацию главного сервера. Отображается текущий файл двоичного журнала, текущая позиция в файле и все включающие или исключающие параметры репликации. Используйте эту команду для подключения или переподключения подчиненных серверов.
- SHOW SLAVE HOSTS Показывает список подчиненных серверов, подключенных к главному, если тот использует параметр --report-host. Используйте эту информацию, чтобы определить, какие подчиненные серверы подключены к главному.
- SHOW SLAVE STATUS Показывает информацию о состоянии системы, играющей в репликации роль подчиненного сервера. Это основная команда для слежения за производительностью и состоянием подчиненных серверов. Она выводит большой объем информации, жизненно важной для поддержания работоспособности подчиненного сервера. Подробнее об этой команде рассказывается в главе 2.

Две самых важных команды из этого списка — SHOW VARIABLES и SHOW STATUS. Переменных очень много (только переменных состояния больше 290), поэтому следует использовать выражение LIKE для ограничения результатов теми аспектами системы, мониторинг которых хотите выполнить.

Переменные обычно выводятся в алфавитном порядке и часто группируются по функциональности. Однако иногда переменные не сгруппированы. В этом случае их можно найти по ключевому слову. Например, команда SHOW STATUS LIKE '%thread%' отображает все переменные состояния, относящиеся к выполнению потоков. На лист. 8-1 показан результат выполнения этой команды в недавнем бета-выпуске MySQL.

Лист. 8-1. Отображение переменных состояния потока

```
mysql> SHOW VARIABLES LIKE '%thread%';
+---------------------------+---------------------------+
| Variable_name             | Value                     |
+---------------------------+---------------------------+
| innodb_file_io_threads    | 4                         |
| innodb_read_io_threads    | 4                         |
| innodb_thread_concurrency | 0                         |
| innodb_thread_sleep_delay | 10000                     |
| innodb_write_io_threads   | 4                         |
| max_delayed_threads       | 20                        |
| max_insert_delayed_threads| 20                        |
| myisam_repair_threads     | 1                         |
| pseudo_thread_id          | 1                         |
| thread_cache_size         | 0                         |
| thread_handling           | one-thread-per-connection |
| thread_stack              |  262144                   |
+---------------------------+---------------------------+
12 rows in set (0.00 sec)
```

В этом примере показаны не только переменные состояния для управления потоком, но и элемент управления потоком для механизма БД InnoDB. Хотя иногда выдается больше информации, чем вы ожидаете, выражение LIKE помогает найти нужные переменные.

Самой сложной частью в мониторинге MySQL может быть определение того, какие переменные нужно изменять, а за какими наблюдать. Как уже говорилось, по этой теме можно найти много ценной информации в онлайновой документации MySQL Reference Manual.

Чтобы продемонстрировать элементы сервера MySQL, мониторинг которых можно выполнять, рассмотрим переменные, управляющие кэшем запросов. Кэш запросов — один из наиболее важных для производительности элементов сервера, если для данных приложений используется механизм БД MyISAM. Этот кэш позволяет серверу записывать в буфер часто используемые запросы и их результаты. Таким образом, чем чаще выполняется запрос,

тем более вероятно, что его результаты можно найти в кэше, а не обращаться к структурам индекса и таблицам для извлечения данных. Понятно, что чтение результатов из оперативной памяти выполняется гораздо быстрее чтения с диска. Это может повысить производительность, если данные считываются чаще, чем записываются (обновляются).

Каждый выполняемый запрос вводится в кэш и имеет время жизни, определяемое по тому, как давно он использовался (старые запросы удаляются из кэша первыми), и сколько памяти выделено для кэша запросов. Кроме того, запрос может удаляться и по другим причинам, неполный список которых приведен ниже:

- частые изменения (в данных или индексах);
- разные формы запроса, приводящие к неудачным обращениям к кэшу. Поэтому для часто используемых данных важно использовать стандартизованные запросы. Далее будет показано, как представления могут помочь с этим;
- получение запросом данных из временных таблиц;
- события транзакций, делающие запросы в кэше недействительными (например, COMMIT).

Чтобы определить, настроен ли кэш запросов и доступен ли в текущей установке MySQL, проверьте переменную have_query_cache. Это системная переменная с глобальной областью действия, но доступная только для чтения. Кэш запросов контролируется при помощи одной из нескольких переменных, перечисленных На лист. 8-2.

Лист. 8-2. Переменные кэша запросов на сервере

```
mysql> SHOW VARIABLES LIKE '%query_cache%';
+------------------------------+----------+
| Variable_name                | Value    |
+------------------------------+----------+
| have_query_cache             | YES      |
| query_cache_limit            | 1048576  |
| query_cache_min_res_unit     | 4096     |
| query_cache_size             | 33554432 |
| query_cache_type             | ON       |
| query_cache_wlock_invalidate | OFF      |
+------------------------------+----------+
6 rows in set (0.00 sec)
```

Как видите, на кэш запросов влияют различные параметры. Следует отметить возможность временно отключить кэш запросов, задав нулевое значение переменной query_cache_size, определяющей объем памяти, выделенной для кэша запросов. Это не связано с переменной have_query_cache, которая просто указывает, доступен ли кэш запросов вообще. Подробнее о настройке кэша запросов см. в разделе «Query Cache Configuration» онлайнового руководства MySQL Reference Manual.

За производительностью кэша запросов можно наблюдать, проверяя переменные состояния, перечисленные на лист. 8-3.

Лист. 8-3. Переменные состояния кэша запросов

```
mysql> SHOW STATUS LIKE '%Qcache%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Qcache_free_blocks      | 0     |
| Qcache_free_memory      | 0     |
| Qcache_hits             | 0     |
| Qcache_inserts          | 0     |
| Qcache_lowmem_prunes    | 0     |
| Qcache_not_cached       | 0     |
| Qcache_queries_in_cache | 0     |
| Qcache_total_blocks     | 0     |
+-------------------------+-------+
8 rows in set (0.00 sec)
```

Здесь мы видим одну из некритичных несогласованностей сервера MySQL. Для управления кэшем запросов используются переменные, имена которых начинаются с query_cache, но имена переменных состояния начинаются с Qcache. Хотя эта несогласованность была создана преднамеренно (для отличия переменных сервера от переменных состояния), подобные странности могут затруднить поиск нужного элемента.

Следует периодически выполнять дефрагментацию кэша запросов при помощи команды FLUSH QUERY CACHE. Результаты из кэша не удаляются, а выполняется внутренняя реорганизация, позволяющая лучше использовать память.

Кэш запросов имеет много параметров, позволяющих управлять им, настраивать его и выполнять мониторинг производительности. Это делает кэш запросов отличным примером, демонстрирующим сложность мониторинга серверов MySQL. Чтобы полностью раскрыть эту тему, потребуется не один том. Поэтому в этой главе приводятся общие рекомендации, применимые к любым функциям сервера MySQL. Однако за конкретными деталями следует обращаться к онлайновой документации MySQL Reference Manual.

Еще одна пара команд, которые могут быть очень полезными для мониторинга репликации — SHOW MASTER STATUS и SHOW SLAVE STATUS. Подробнее об этих командах рассказывается далее в этой главе.

Утилита mysqladmin

Утилита командной строки mysqladmin — «рабочая лошадка» среди средств командной строки. Эта утилита может работать со многими параметрами и командами. В руководстве MySQL Reference Manual она описывается весьма коротко. В этом разделе мы рассмотрим параметры и команды для мониторинга сервера MySQL.

Так как эта утилита запускается из командной строки, можно писать сценарии для выполнения наборов операций гораздо более простым способом, чем при помощи команд SQL. Некоторые сторонние утилиты для мониторинга используют комбинацию mysqladmin и команд SQL для сбора и отображения информации.

Необходимо указать информацию для подключения (имя пользователя, пароль, имя хоста и т. д.), чтобы подключиться к запущенному серверу. Ниже приведен список часто используемых команд. Как вы увидите, большинство из них имеет эквивалентные команды SQL, позволяющие получить ту же информацию.

- *status* Показывает краткое описание состояния сервера, включая время работы, число потоков (подключений), число запросов и общие статистические данные. Эта команда предоставляет быстрый отчет о работоспособности сервера.
- *extended-status* Показывает полную статистическую информацию о системе. Аналогична команде SQL SHOW STATUS.
- *processlist* Показывает список текущих процессов. Работает аналогично команде SQL SHOW PROCESSLIST.
- *kill <thread id>* Позволяет завершить указанный процесс. Можно использовать вместе с командой processlist для управления вышедшими из под контроля и зависшими процессами.
- *variables* Показывает системные переменные сервера и их значения. Эквивалентна команде SQL SHOW VARIABLES.

Существует много параметров и других команд, не перечисленных здесь, включая команды для запуска и остановки подчиненных серверов во время репликации и управления различными системными журналами.

Одна из лучших возможностей утилиты mysqladmin — сравнение информации, полученной в разные моменты времени. Если указать параметр --sleep n, утилита будет выполнять указанную команду каждые n секунд. Например, чтобы просматривать список процессов локальной системы, обновляемый каждые 3 секунды, используйте следующую команду:

```
mysqladmin -uroot --password --socket=<sock> processlist --sleep 3
```

Эта команда будет выполняться до тех пор, пока не будет отменена нажатием Ctrl+C.

Наверно, одна из самых полезных возможностей — сравнение результатов команды extended-status. Чтобы сравнить значения, полученные во время предыдущего запуска, с текущими значениями, используйте параметр --relative. Например, чтобы сравнить предыдущие и текущие значения переменных состояния системы, выполните следующую команду:

```
mysqladmin -uroot --password --socket=<sock> extended-status -relative
--sleep 3
```

Можно также комбинировать команды, чтобы получать несколько отчетов одновременно. Например, чтобы просмотреть список процессов вместе с информацией о состоянии, выполните следующую команду:

```
mysqladmin --root ... processlist status
```

Утилита mysqladmin имеет много других способов применения. С ее помощью можно завершать работу сервера, очищать журналы, опрашивать сервер, запускать и останавливать подчиненные серверы репликации, а также обновлять таблицы привилегий. Подробнее об этой утилите можно узнать в разделе «mysqladmin—Client for Administering a MySQL Server» руководства MySQL Reference Manual. На рис. 8-1 показан пример выходных данных, полученных в системе без нагрузки.

```
+----+-------------+-----------+----+---------+------+---------------------------------+------------------+
| Id | User        | Host      | db | Command | Time | State                           | Info             |
+----+-------------+-----------+----+---------+------+---------------------------------+------------------+
| 1  | system user |           |    | Daemon  | 0    | Waiting for event from ndbcluster |                  |
| 18 | root        | localhost |    | Query   | 0    |                                 | show processlist |
+----+-------------+-----------+----+---------+------+---------------------------------+------------------+
Uptime: 6325  Threads: 1  Questions: 139  Slow queries: 0  Opens: 17  Flush tables: 2  Open tables: 1  Queries
per second avg: 0.21
```

Рис. 8-1. Пример отчета mysqladmin о процессах и состоянии

Утилиты MySQL GUI

Утилиты MySQL GUI в настоящий момент объединены в один пакет, который можно скачать с веб-сайта MySQL. Доступны пакеты для нескольких популярных операционных систем:

- MySQL Administrator 1.2;
- MySQL Query Browser 1.2;
- MySQL Migration Toolkit 1.1.

MySQL Administrator и MySQL Query Browser мы рассмотрим в следующих разделах. Пакет MySQL Migration Toolkit предназначен для автоматизации переноса схемы базы данных и информации из других СУБД. Он может быть очень удобным инструментом для быстрой адаптации данных для MySQL.

Пакет MySQL Migration Toolkit не доступен на платформе Mac OS X.

MySQL Administrator

MySQL Administrator — «мастер на все руки». Или почти на все. Этот инструмент предоставляет средства для просмотра и изменения системных переменных, управления файлами конфигурации, проверки журналов сервера, мониторинга переменных состояния и даже просмотра графических представлений сведений о производительности ряда ключевых элементов. Также он имеет полный набор административных параметров, позволяющих управлять пользователями и просматривать конфигурации баз данных. Хотя этот инструмент был создан для замены утилиты mysqladmin, попу-

лярность у пользователей говорит о том, что в обозримом будущем актуальными останутся оба инструмента.

MySQL Administrator можно использовать на любой платформе и обращаться к одному или нескольким серверам, подключенным к клиенту. Это облегчает мониторинг нескольких серверов по сети.

Уникальная функция MySQL Administrator позволяет анализировать отключенный сервер. Хотя большинство функций отключено (т. е. нет индикаторов производительности, работающих в реальном времени), все равно можно просматривать конфигурацию и проверять журналы. Это может быть очень полезно для диагностирования сбоев сервера.

Чтобы подключиться к отключенному серверу (это называется режимом настройки и обслуживания), запустите MySQL Administrator и нажмите клавишу Ctrl (или клавишу Apple в Mac OS X). Обратите внимание на то, что кнопка Connect превратилась в кнопку Skip. Нажмите кнопку Skip, и утилита переключится в режим настройки и обслуживания. Теперь можно запуска и останавливать сервер, изменять переменные загрузки и просматривать журналы.

Функции MySQL Administrator разделены на 10 групп или утилит, каждая из которых представлена вкладками приложения. Нам интересны утилиты, показывающие значения переменных состояния. В MySQL Administrator есть динамическая функция, аналогичная утилите mysqladmin, но расширяющая ее возможности, показывая график значений. Эти утилиты сгруппированы на вкладке Health. Если щелкнуть на ярлыке вкладки Connection Health, будет показан график, аналогичный показанным на рис. 8-2, 8-3 и 8-4.

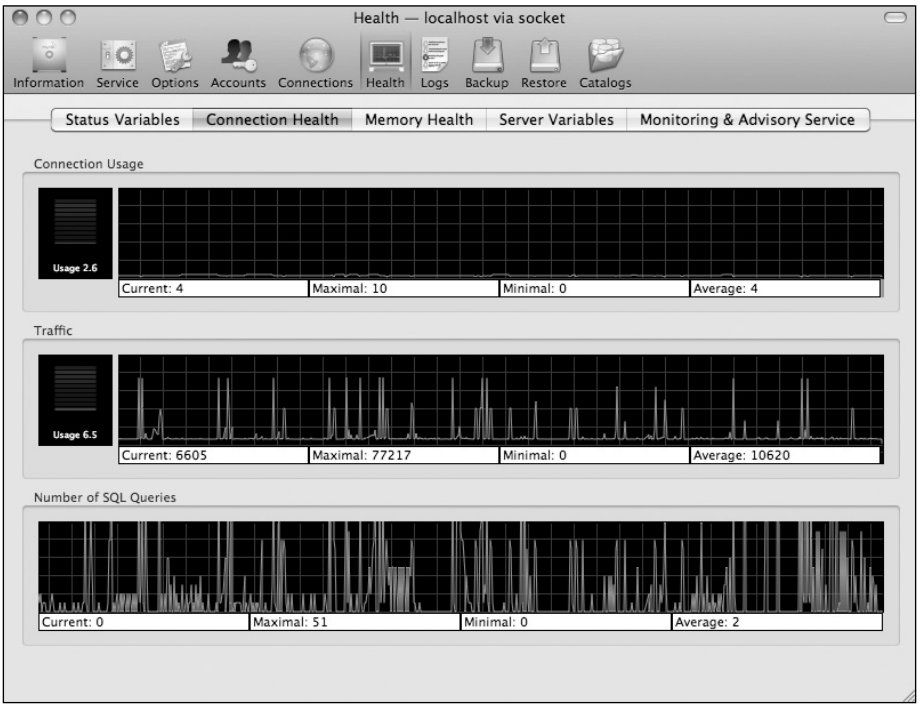


Рис. 8-2. Вкладка Connection Health (Mac OS X)

Этот график имеет три секции. В первой представлены данные об использовании подключения (сколько подключений установлено на сервере в текущий момент). Это может быть очень полезным, когда требуется определить, не перегружен ли сервер (слишком много подключений), или когда требуется решить единичные проблемы с подключениями.

Снимки экрана, представленные на рис. 8-2, 8-3 и 8-4 получены на системах Mac OS X, Windows 7 и Linux. Как видите, внешний вид приложения немного отличается на каждой платформе. В Mac OS X различается расположение меню, но в остальном набор функций один и тот же.

На графике Traffic (рис. 8-2) показан сетевой трафик. Это может быть полезным при определении потенциальных проблем с сетью.

Наиболее интересен нижний график, показывающий запросы SQL, выполненные в некоторый промежуток времени. Это может быть очень полезным, когда требуется определить, не перегружена ли система запросами.

По рисункам из примеров можно понять, что система работает с очень небольшим числом подключений, минимальным сетевым трафиком и средней нагрузкой запросами, проявляющейся всплесками.

Графическое представление переменных состояния — одна из лучших возможностей MySQL Administrator. Хотя сами переменные состояния на графиках не видны, а отображаются только максимальные и минимальные значения за некоторый период, это может быть очень полезным для визуального определения потенциальных проблем.

Долгое использование подобных утилит может добавить небольшую нагрузку и немного исказить результаты. Однако это не будет проблемой для системы, выполняющей большое число операций.

На рис. 8-3 показана система, которая практически не используется. Мы добавили этот рисунок, чтобы показать, как выглядит окно MySQL Administrator на платформе Windows. Кроме того, этот пример важно рассмотреть с точки зрения репликации. Если подчиненный сервер демонстрирует такое поведение, то, определенно, возникли какие-то серьезные проблемы. Также это может указывать на то, что схема масштабирования не работает так, как должна, или стратегия балансировки нагрузки не сработала, и запросы не передаются на этот сервер.

На рис. 8-4 показан интерфейс MySQL Administrator в системе Linux. Это еще один пример системы с умеренной нагрузкой. Обратите внимание на то, что для использования подключения и сетевого трафика имеются индикаторы, показывающие процент загруженности. Это позволяет быстро оценивать уровень нагрузки.

На вкладке Memory Health показаны динамические графики состояния кэша запросов и кэша ключей. На рис. 8-5 показан пример графических отчетов для системы с умеренной нагрузкой. В этом случае система выполняет дополнительные диагностические тесты продукта MySQL Cluster.

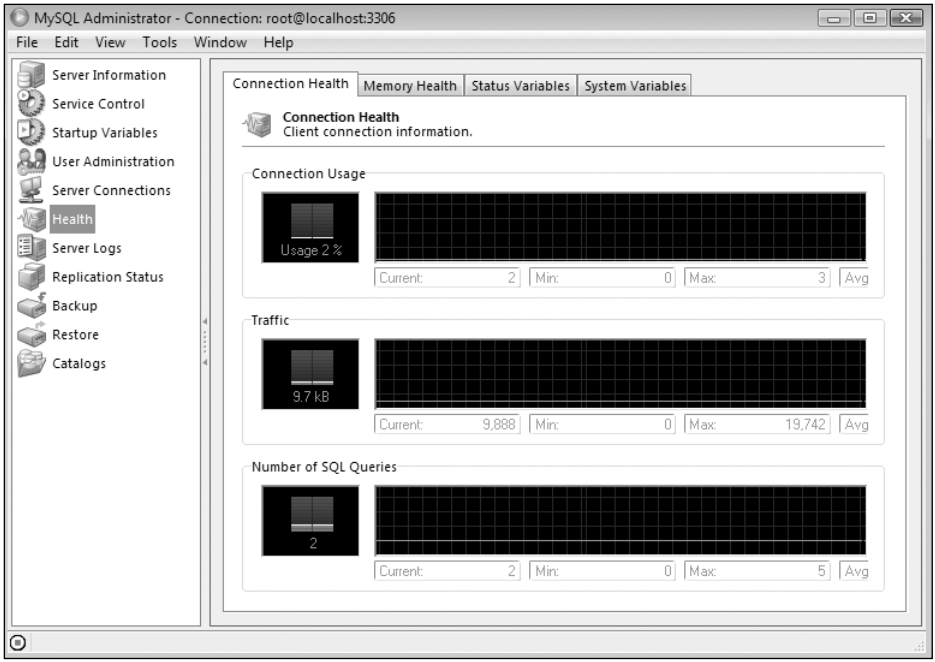


Рис. 8-3. Вкладка Connection Health (Windows 7)

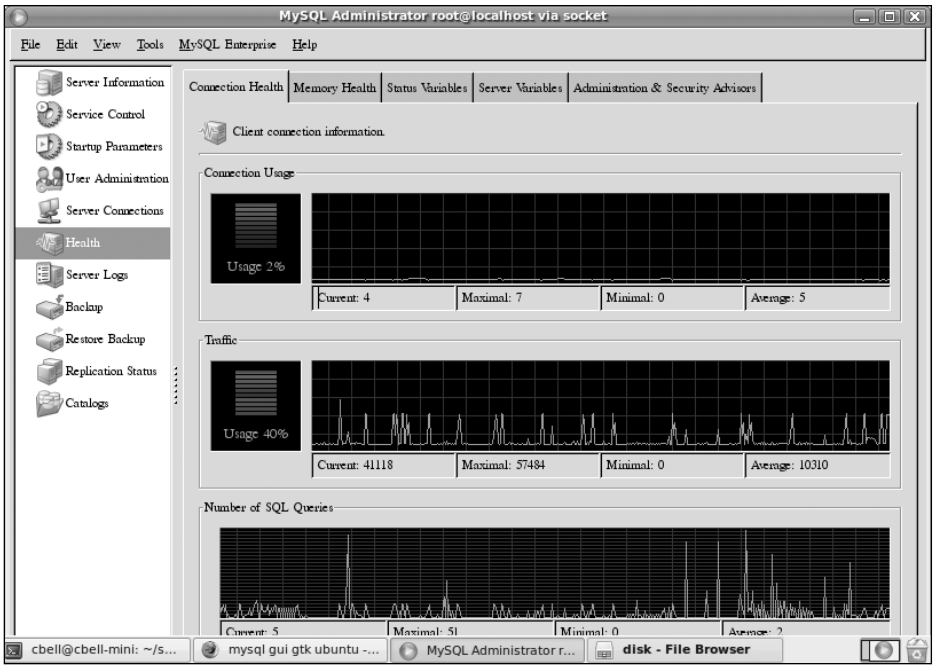


Рис. 8-4. Вкладка Connection Health (Linux)

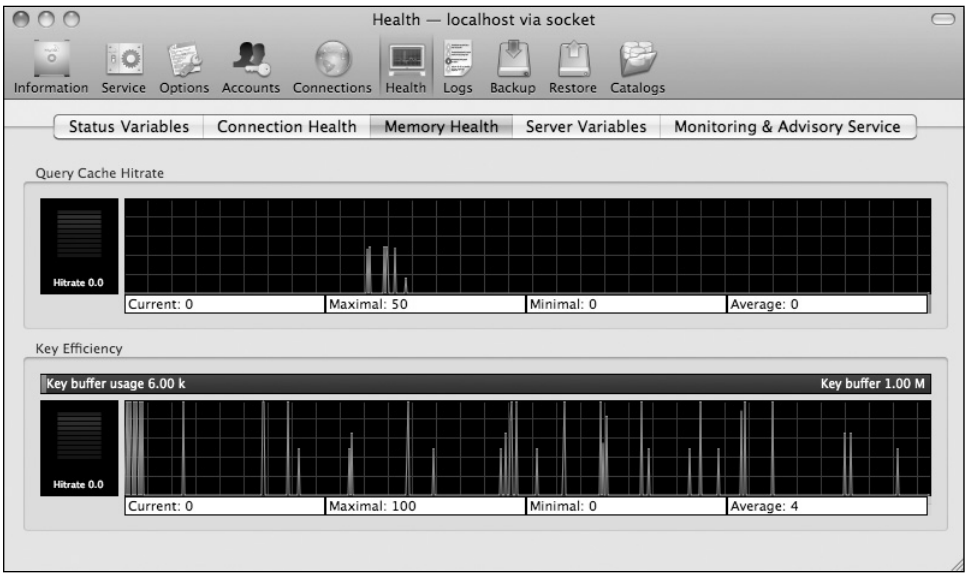


Рис. 8-5. Вкладка Memory Health

По сравнению с запросами переменных состояния кэша запросов, выполняемыми вручную, здесь гораздо проще следить за общим уровнем производительности кэша запросов. Этот график, как и некоторые другие динамические графики, имеет индикатор пиков, помогающий следить за внезапными скачками значений.

Принцип неопределенности в MySQL

Возможно, вы знакомы с принципом неопределенности Гейзенберга, который гласит, что чем точнее определены координаты частицы, тем менее точен ее импульс, и наоборот. Мониторинг MySQL включает выполнение запросов для получения значений переменных состояний, поэтому каждый раз, когда вы собираете данные, вы увеличиваете значение некоторых переменных, за которыми наблюдаете. Таким образом, мониторинг добавляет дополнительную нагрузку, и слишком активное измерение может сделать измеряемые значения менее ценными. Конечно, этот принцип применяется к любым измерениям данных.

Планируя частоту получения статистики, следует учитывать, что частый замер значений может привести к тому, что система потратит больше ресурсов на измерения, чем на саму обработку данных. На первом графике показана только частота успешных обращений к кэшу запросов. Здесь представлены те же значения, что возвращаются переменной состояния Qcache_hits, которая является счетчиком успешных извлечений результатов запросов из кэша. Понятно, что чем выше эти значения, тем лучше производительность, но обратное утверждение не обязательно верно. Низкая частота попаданий говорит о том, что сервер не использует кэш запросов, а не о том, что он плохо

функционирует. Однако, если ваша цель — повысить производительность, используя стандартизированные запросы, а вы видите низкую частоту попаданий, это может указывать на то, что требуется настройка кэша запросов, или что какие-то события делают данные из кэша недействительными.

Значение Qcache_hits монотонно возрастает, и его необходимо периодически сбрасывать, чтобы оценивать периоды активности. Для построения графика Query Cache Hitrate (рис. 8-5) MySQL Administrator выполняет команду FLUSH STATUS, чтобы сбросить значение на 0 между обращениями за данными для текущего сеанса. Это обычный прием для большинства средств мониторинга, выполняющих выборку данных.

График Key Efficiency (рис. 8-5) динамически отображает информацию об использовании буфера ключей. Точнее, это кэш ключей MyISAM, и на графике показано, как часто используется кэш ключей или сколько запросов выполнено для обращения к этому кэшу. Здесь используется переменная состояния key_read_requestst.

Кэш ключей — основной элемент, используемый механизмом хранилища MyISAM для повышения производительности. В него записываются наиболее часто используемые блоки индекса (указатели на данные), что позволяет быстрее искать индексированные значения, а значит, быстрее извлекать данные. Подробнее о кэше ключей рассказывается в гл. 9.

При помощи этого графика можно определять, когда следует увеличить значение переменной key_cache_size для обеспечения лучшей производительности. Т.е., если отображаются высокие значения (большой процент), можно увеличить значение. В общем, чем выше значение, тем более эффективным становится кэш. И наоборот, если вы видите низкие значения, когда выполняется много запросов к таблицам MyISAM, можно уменьшить размер кэша ключей.

Остальные вкладки представляют более традиционные числовые данные. Прежде чем мы рассмотрим эти вкладки, следует отметить возможность создания собственных графиков состояния. Чтобы добавить такой график, щелкните правой кнопкой мыши в пустом месте рабочей области, выберите команду New page и введите название страницы. Это название появится на панели вкладок. Откройте новую страницу, щелкните правой кнопкой мыши на пустой форме и выберите команду New group, чтобы дать имя группе. После этого можно щелкнуть правой кнопкой мыши в созданной группе и добавить новый график. На рис. 8-6 показано диалоговое окно для создания графика.

Здесь можно указать переменные состояния, за которыми вы хотите наблюдать, вычисления, которые требуется выполнять, и способ представления данных. Попробуйте сделать это самостоятельно. Обратите внимание, на рис. 8-6 выбран линейный график (Line Graph), а перед именем переменной, заключенным в скобки, стоит знак ^. В диалоговом окне даются подсказки о том, как ввести формуле, если хотите получить более сложный отчет.

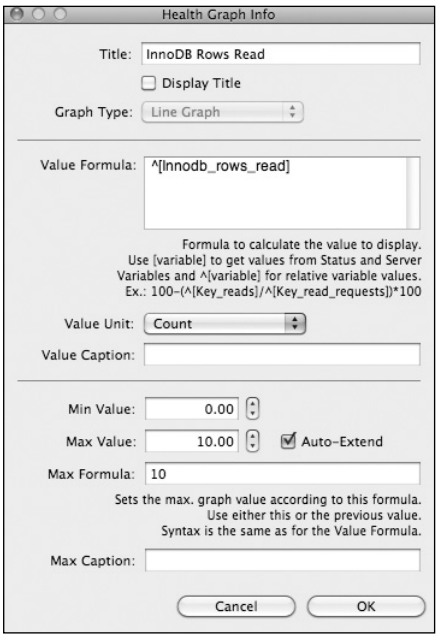


Рис. 8-6. Диалоговое окно для создания настраиваемого графика

При необходимости можно также указать единицы измерения значения, подпись, минимальное и максимальное значения. Если указать имя для значения подписи, появится небольшой пиковый индикатор, имеющийся на других графиках. Пример созданного пользователем графика показан на рис. 8-7.

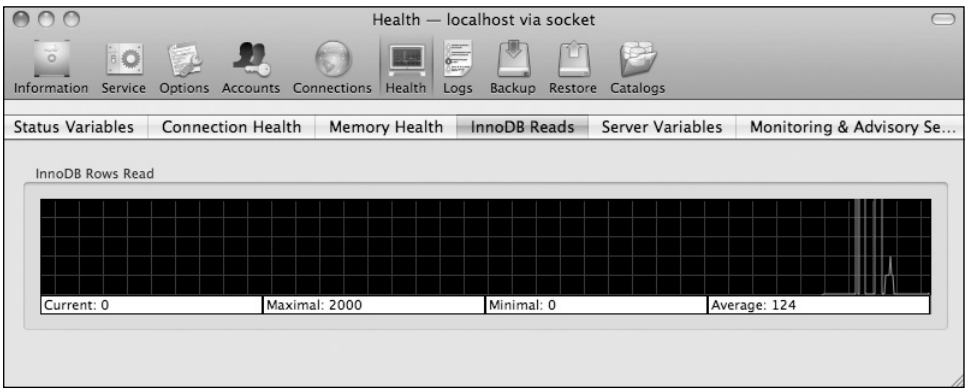


Рис. 8-7. Пользовательский график

Можно щелкнуть правой кнопкой мыши на любом из стандартных графиков, и изменить его. Эту возможность можно использовать, чтобы посмотреть, как вычисляются значения для графиков.

Как уже говорилось, вместе с информацией о состоянии можно просматривать переменные сервера. Щелкните на ярлыке вкладки Server Variables,

и содержимое окна изменится, став похожим на электронную таблицу. В левой части окна находится дерево, в котором все системные переменные объединены в категории и подкатегории.

В центральной части окна отображаются сведения о выбранной категории. Чтобы раскрыть категорию, дважды щелкните на ее названии. Например, если дважды щелкнуть на названии категории General, появится список ее подкатегорий. Щелкните на названии подкатегории Performance, чтобы просмотреть список системных переменных, имеющих отношение к настройке производительности.

Хотя это неполный список, он включает все основные переменные производительности. Чтобы просмотреть другие переменные, выберите любую другую группу. На рис. 8-8 показаны основные переменные производительности в стандартной установке.

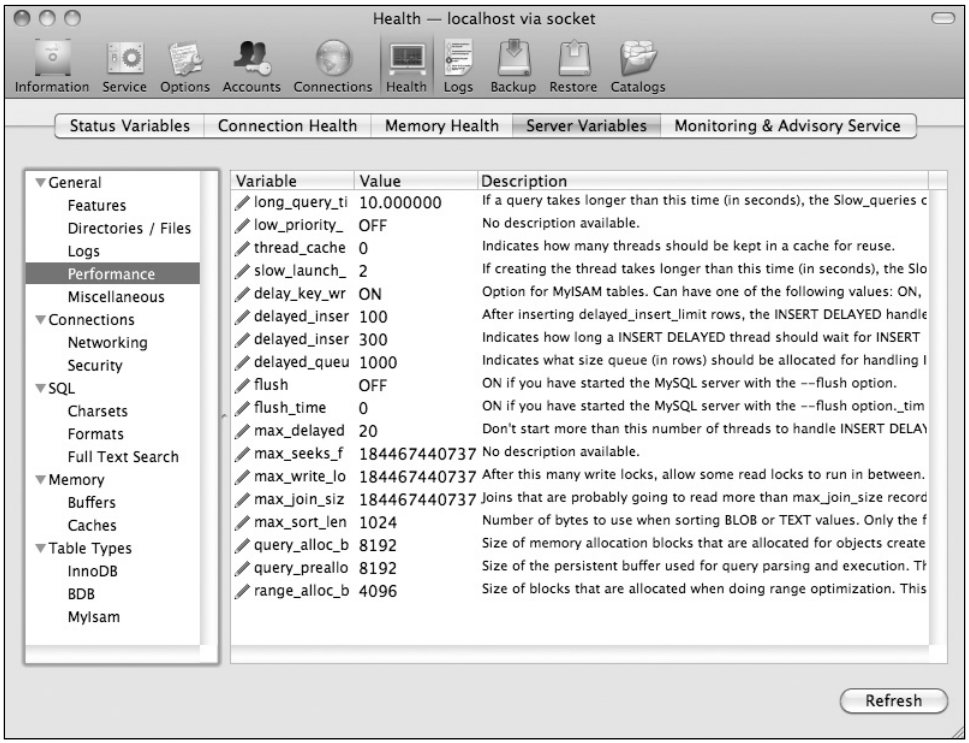


Рис. 8-8. Окно Server Variables

Одной из самых полезных возможностей этого инструмента является то, что можно динамически изменять значения. Например, если требуется увеличить время, в течение которого система считает медленные потоки, просто щелкните на значении переменной slow_launch_time и введите новое значение. Измененные значения автоматически обновляются на сервере. Это очень полезная возможность, благодаря которой использовать MySQL Administrator гораздо удобнее, чем клиентское окно.

Определить, какие переменные могут быть изменены (не являются доступными только для чтения), можно по небольшому значку в виде карандаша рядом с именем. Если такого значка нет, переменная доступна только для чтения.

Когда вы объединяете возможность извлекать данные и быстро изменять системные переменные с возможностью просматривать изменения в системе на динамических графиках, вы используете MySQL Administrator как профессиональный инструмент для настройки производительности. Но не забывайте золотое правило настройки: изменяйте только один параметр за раз и наблюдайте за результатом.

На вкладке Status Variables показаны переменные состояния системы, сгруппированные по функциональности. Например, чтобы просмотреть все переменные состояния, относящиеся к сетевому трафику, щелкните на названии вкладки Networking и выберите подгруппу Traffic. Окно Status Variables показано на рис. 8-9.

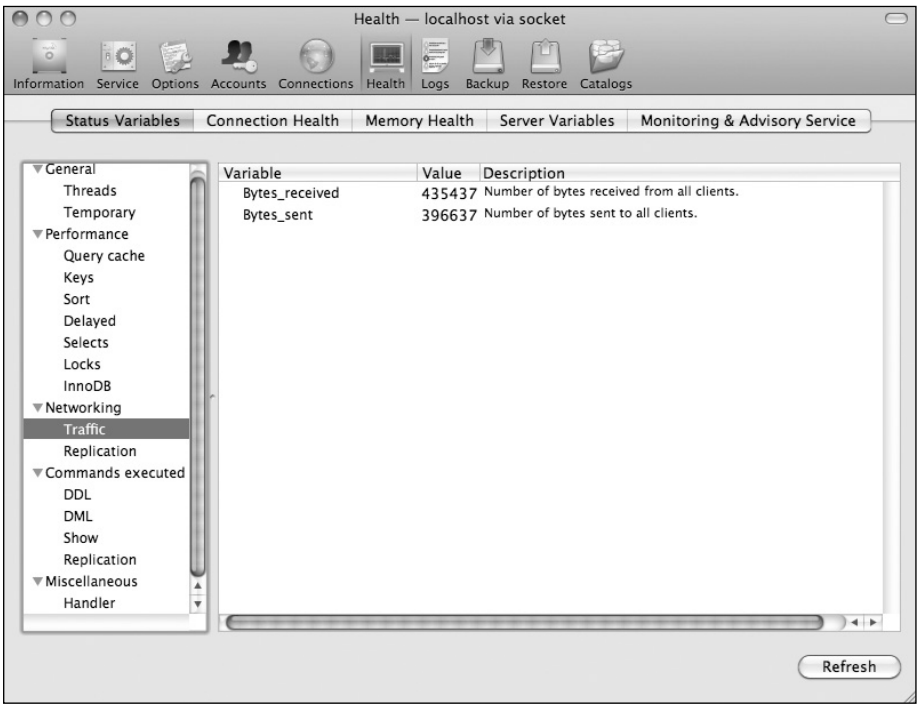


Рис. 8-9. Окно Status Variables

Как и в переменные сервера, все переменные состояния системы можно просматривать. Отображаются значения, полученные в момент выбора группы переменных. Автоматически они не обновляются. Чтобы получить новые значения, щелкните на кнопке Refresh. В отличие от динамических графиков состояния, во время обновления получаются значения в том виде, в каком они считаны с сервера при помощи команды SHOW STATUS.

Помните об этом, используя эту утилиту: она показывает только статические одномоментные значения, а не тенденции и списки истории.

Для каждой переменной состояния имеется краткое описание. Это может быть очень полезно для определения того, за какими переменными нужно наблюдать при решении проблем с производительностью. Это позволяет сэкономить много времени по сравнению с выполнением команд SHOW STATUS и попытками угадать, какое ключевое слово нужно использовать (см. рис. 8-5).

Следующий инструмент, который мы рассмотрим, позволяет просматривать мгновенные снимки журналов сервера. О журналах сервера рассказывается далее в этой главе, поэтому здесь мы приведем только краткое представление.

В журналы сервера записываются значимые события, произошедшие во время работы системы. Эта информация может включать ошибки операций или подключений, выполненные на сервере запросы и даже указания на то, какие запросы выполняются медленнее всех.

Эти журналы можно просматривать при помощи MySQL Administrator. Хотя доступ предоставляется только к журналу ошибок, общему журналу и журналу медленных операций, это все равно очень полезно.

Чтобы просмотреть журналы сервера, к которому вы подключены, щелкните на ярлыке вкладки Logs на панели инструментов. Откроется страница Logs (рис. 8-10).

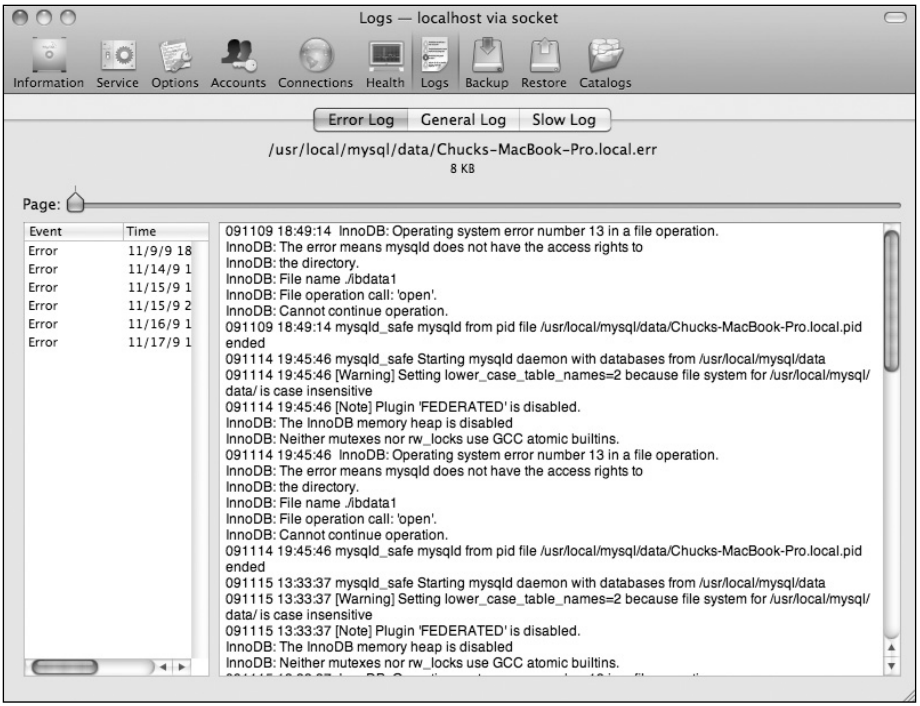


Рис. 8-10. Окно Logs

На некоторых платформах для просмотра и взаимодействия с журналами сервера при помощи MySQL Administrator могут потребоваться учетные данные администратора.

В верхней части окна находится ползунок Page, который можно перетаскивать для перехода по страницам журнала.

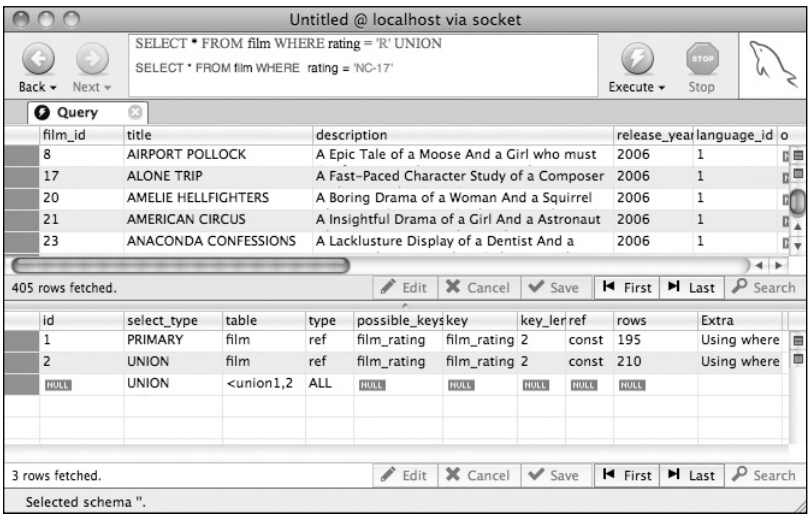


Рис. 8-11. MySQL Query Browser

Ниже находятся два поля, в которых перечислены значимые события из журнала (слева) и сведения о выбранном событии (справа).

Как видите, в списке на рисунке есть несколько ошибок. Чтобы просмотреть информацию об ошибке, выберите ее в списке слева. Это может значительно сэкономить время при изучении большого журнала в поисках событий, произошедших в определенный день в определенное время.

Мы рассмотрели средства MySQL Administrator, позволяющие выполнять подробную диагностику, мониторинг производительности и настройку. Надеемся, вы будете использовать их, когда перейдете к практике на собственных серверах. MySQL Administrator избавляет от рутинной работы по запуску команд SQL для получения одних и тех же данных. Новые возможности, такие как настраиваемые графики состояний, значительно облегчают настройку производительности.

MySQL Query Browser

MySQL Query Browser — еще один инструмент с графическим интерфейсом. Он используется для построения запросов в графической форме и их выполнения. Возвращаемые наборы результатов отображаются в диалоговом окне, похожем на электронную таблицу. Результаты можно вертикально и горизонтально прокручивать и изменять размер столбцов для удобства просмотра. Многие пользователи находят этот инструмент более удобным и

легким в использовании, чем старую клиентскую утилиту командной строки (mysql).

К функциональности, относящейся к производительности и представляющей дополнительную ценность для администраторов, относится графическое отображение результатов команды EXPLAIN для любого указанного запроса. На рис. 8-11 показан пример из базы данных sakila. Подробнее об этом будет рассказано далее в этой главе.

Показанный пример дает представление об утилитарном использовании GUI. Можно ввести любой запрос и посмотреть информацию о его выполнении, сначала выполнив его, а затем выбрав в меню Query команду Explain Query.

Обратите внимание, результаты разделены на две части. В нижней отображаются результаты команды EXPLAIN и возвращенные строки. Полосы прокрутки позволяют просматривать данные, не выполняя запрос повторно.

Почему мы относим этот инструмент к средствам настройки производительности? Он позволяет написать запрос, выполнить команду EXPLAIN, просмотреть результаты, после его либо переписать запрос, либо настроить индексы, затем еще раз выполнить запрос и просмотреть изменения в графическом интерфейсе. Если вы думали, что утилиты для выполнения запросов предназначены только для пользователей, то насчет этого инструмента вы ошиблись.

Но это еще не все. В MySQL Query Browser имеются расширенные средства редактирования, такие как цветная подсветка кода (см. рис. 8-11). Информацию обо всех функциях и способах применения MySQL Query Browser можно найти в документации по утилитам GUI MySQL (*http://dev.mysql.com/doc*).

В Windows можно установить MySQL System Tray Monitor для быстрого просмотра сведений о состоянии сервера. Зеленый значок указывает на то, что сервер работает, красный — сервер остановлен. Кроме того, с помощью этой программы можно быстро выполнять общие функции, такие как завершение работы и запуск MySQL Administrator или MySQL Query Browser.

Журналы сервера

Опытные администраторы Linux и Unix знакомы с журналированием и понимают его важность. Технология MySQL вышла из той же среды. Поэтому серверы MySQL имеют несколько журналов, которые содержат важную информацию об ошибках, событиях и изменениях данных.

В этом разделе мы рассмотрим различные журналы серверов MySQL и опишем значения каждого из них для мониторинга и повышения производительности. Файлы журналов могут предоставить много информации о прошлых событиях (в этом-то и суть).

Журналы сервера MySQL разделяются на несколько типов. В зависимости от типа, в журнал могут записываться выполненные команды SQL, долго выполняющиеся (медленные) запросы, изменения данных и результаты команд резервного копирования и восстановления. Используя параметры загрузки, можно включить и отключить любой журнал. В большинстве установок включен как минимум журнал ошибок. На серверах MySQL используются следующие журналы:

- общий журнал запросов (General Log);
- журнал медленных запросов (Slow Log);
- журнал ошибок (Error Log);
- двоичный журнал (Binary Log);
- журналы резервного копирования (Backup Log).

Общий журнал запросов, как подразумевает его название, содержит информацию о том, что делает сервер, включая сведения о подключениях клиентов и копии команд, отправленных серверу. Как можно догадаться, этот журнал растет очень быстро. Проверяйте общий журнал, когда выполняете диагностику ошибок, связанных с клиентами, или когда хотите определить, какие клиенты отправляют команды определенного типа.

В общий журнал запросов команды записываются в том порядке, в каком они были *получены* от клиентов, и могут не отражать действительный порядок выполнения.

Чтобы включить общий журнал, укажите параметр загрузки --log. Имя файла журнала можно указать при помощи параметра загрузки --log-output. Эти параметры имеют эквивалентные динамические переменные. Например, выражение SET GLOBAL log_output = FILE; указывает, что выходные данные журнала запущенного сервера должны записываться в файл. Значения этих переменных можно считывать при помощи команды SHOW VARIABLES.

В журнал медленных запросов записываются копии долго выполняющихся запросов. Этот журнал имеет такой же формат, что и общий журнал, и его так же можно контролировать при помощи параметра загрузки --log-slow-queries. Переменная сервера, определяющая, какие запросы будут записаны в журнал медленных запросов, — log_query_time (значение в секундах). Можете изменять это значение в соответствии с требованиями для вашего сервера и приложений, чтобы определить, когда запросы работают медленнее, чем ожидается. Записи журналов можно отправить в файл и/или таблицу, используя параметры FILE, TABLE или BOTH.

Журнал медленных запросов может быть очень эффективным инструментом, позволяющим обнаруживать проблемы с запросами раньше, чем на них начнут жаловаться пользователи. Конечно, лучше, чтобы этот журнал содержал мало записей или был совсем пустым. Это не значит, что следует задать переменной очень большое значение. Наоборот, следует использовать значение, соответствующее вашим ожиданиям, и изменять его при изменении обстоятельств.

Подчиненные серверы не ведут журналов медленных запросов. Но если использовать параметр --log-slow-slave-statements, они будут записывать медленно выполняющиеся события в свои журналы.

Журнал ошибок содержит информацию, полученную при запуске или остановке сервера MySQL, и ошибки, сгенерированные во время работы сервера. Журнал ошибок — первое место, куда следует заглянуть, начиная поиск причин сбоев сервера MySQL. В некоторых ОС журнал ошибок может содержать также записи из стека (или дамп памяти).

Журнал ошибок можно включить или отключить, используя параметр загрузки --log-error. По умолчанию журнал ошибок имеет имя хоста с расширением .err и сохраняется в базовом каталоге (там же, где находится каталог данных).

Если запустить сервер с параметром --console, ошибки будут выводиться не только в журнал, но и в стандартный выход ошибок.

Двоичный журнал хранит все изменения, внесенные в данные на сервере, а также статистическую информацию о выполнении исходных команд на сервере.

В онлайновом руководстве MySQL Reference Manual говорится, что двоичные журналы используются для резервного копирования. Однако практика показывает, что более популярным способом применения двоичных журналов является репликация.

Уникальный формат двоичного журнала позволяет использовать его для выполнения инкрементного резервного копирования, сохраняя файлы двоичного журнала, созданные между операциями резервного копирования. Для этого следует очищать и менять двоичные журналы (закрывать журнал и открывать новый). Это позволяет сохранить набор изменений, произошедших с момента последнего резервного копирования. Этот же прием позволяет выполнять восстановление до состояния в определенный момент времени (PITR). Для этого нужно восстановить данные из резервной копии и применить двоичный журнал до определенного времени. Подробнее о двоичных журналах см. в гл. 3. Подробнее о восстановлении PITR см. в гл. 12.

Так как в двоичный журнал записываются копии всех изменений данных, это добавляет некоторую нагрузку на сервер, но получаемые преимущества оправдывают небольшое снижение производительности.

Затраты на ведение двоичного журнала могут быть гораздо выше, в зависимости от параметров дисков. Если двоичный журнал включен и используется механизм БД InnoDB, параллельное подтверждение не выполняется. Это может быть проблемой при большом количестве операций записи.

Чтобы включить двоичный журнал, используйте параметр загрузки --log-bin, указав имя корневого файла двоичного журнала. Сервер добавляет в конец имени файла последовательный номер, предоставляя возможность автоматической и ручной смены журнала. Хотя обычно это не требуется, можно

также сменить имя индекса двоичных журналов, указав параметр загрузки --log-bin-index. Смена журнала выполняется командой FLUSH LOGS.

Кроме этого, можно определять, что будет записываться в журнал (включающее журналирование), а что не будет (исключающее журналирование), используя параметры --binlog-do-db и --binlog-ignore-db, соответственно.

Сторонние утилиты

Существует несколько весьма полезных утилит сторонних производителей. Наиболее популярные из них: MySAR, mytop, InnoTop и MONyog. Все они — утилиты командной строки, которые можно запускать в любом окне консоли и подключать к любому доступному по сети серверу MySQL. Вкратце обсудим каждую из этих утилит.

MySAR

MySAR — система отчетов об активности, напоминающая команду sar в Linux; т. е. это команда sar для MySQL. MySAR аккумулирует выходные данные команд SHOW STATUS, SHOW VARIABLES и SHOW FULL PROCESSLIST и сохраняет их в базе данных mysar на сервере. Параметры сбора данных можно настраивать различными способами, включая ограничение собираемых данных. Можно удалять старые данные, чтобы утилита MySAR продолжала работать все время, и не беспокоиться о заполнении дисков данными о состоянии.

MySAR — open source-программа, имеющая открытую лицензию GNU второй версии (GPL v2). Скачать MySAR можно по адресу *https://launchpad.net/mysar*.

mytop

Утилита mytop позволяет выполнять мониторинг статистики потоков и общей статистики производительности MySQL. Она позволяет просматривать общие сведения, такие как имя хоста, версия сервера, число выполненных запросов, среднее время выполнения запросов, общее число потоков и т. д. Это аналог команды top в Linux.

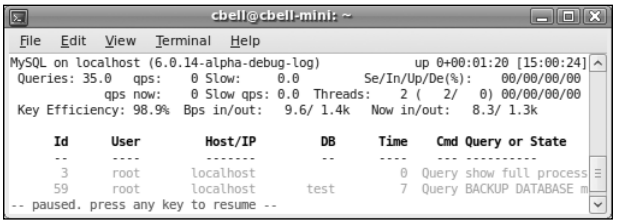


Рис. 8-12. Утилита mytop

Эта утилита периодически выполняет команды SHOW PROCESSLIST и SHOW STATUS и отображает информацию в виде листинга, как команда

top. Автор mytop — Джереми Д. Заводны (Jeremy D. Zawodny), поддержкой же занимается не только он, но и сообщество MySQL. На рис. 8-12 показан пример работы mytop.

Утилита mytop имеет открытый исходный текст и лицензию GPL v2. Скачать ее можно по адресу *http://jeremy.zawodny.com/mysql/mytop*.

InnoTop

InnoTop — еще одна система отчетов об активности, напоминающая средства Linux. В данном случае за основу взята команда top, а источником вдохновения стала утилита mytop. InnoTop во многом повторяет возможности mytop, но предназначена для мониторинга производительности InnoDB, а также сервера MySQL. Она позволяет наблюдать за ключевой статистикой, касающейся транзакций, мертвых блокировок, внешних ключей, активности запросов, активности репликации, системных переменных и прочего. InnoTop широко используется, и многие считают, что это стандартный инструмент для мониторинга производительности. Эта утилита имеет много функций, позволяющих выполнять динамический мониторинг системы. Если в качестве основного (или стандартного) механизма БД вы используете InnoDB и хотите иметь многофункциональный инструмент мониторинга, работающий в текстовом режиме, можете не искать ничего другого, кроме InnoTop (рис. 8-13).

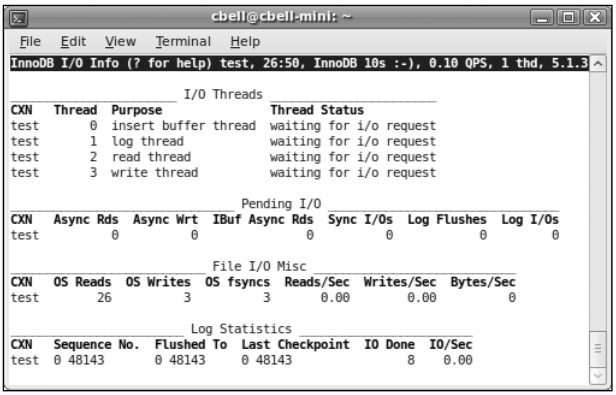


Рис. 8-13. Утилита InnoTop

Утилита InnoTop имеет лицензию GPL v2. Скачать ее можно по адресу *http://innotop.sourceforge.net/*.

MONyog

MySQL Monitor and Advisor (MONyog) — еще один хороший инструмент для мониторинга MySQL. Это решение для мониторинга и профилактики, позволяющее задавать параметры ключевых компонентов, влияющих на безопасность и производительность, и включающее средства для настройки серверов для получения наилучшей производительности. Можно опреде-

лять события для мониторинга определенных параметров и получать оповещения, когда значения достигнут указанных величин. Основные возможности MONyog:

- мониторинг ресурсов сервера;
- выявление плохо выполняющихся выражений SQL;
- мониторинг журналов сервера (например, журнала ошибок);
- мониторинг производительности запросов в реальном времени и выявление долго выполняющихся запросов;
- оповещение о значимых событиях.

Скачать MONyog можно по адресу *http://www.webyog.com/en*.

Пакет MySQL Benchmark Suite

Нагрузочное тестирование позволяет определить, как работает система при определенной нагрузке. Способы тестирования широко варьируются и в чем-то напоминают искусство. Их цель — оценить и записать статистические данные о поведении системы во время выполнения четко определенных наборов тестов с легкой, средней и высокой нагрузкой. Такие испытания нужны, чтобы определить, какой производительности системы можно ожидать.

Это важно, так как позволяет узнать, когда серверы работают не так, как ожидается. Например, если в какой-то период времени пользователи сообщают о снижении производительности сервера, как вы узнаете, что сервер функционирует не так, как должен? Допустим, вы проверили все, что обычно проверяют в таких случаях: память, диски и т.д., и все это работает нормально, без ошибок и отклонений. Как тогда определить, что сервер работает медленнее, чем обычно? Для этого применяются оценочные испытания. Можно снова запустить тест и, если полученные значения намного ниже обычных (или выше, в зависимости от того, что измеряется), становится ясно, что производительность системы ниже ожидаемой.

Для проведения собственных испытаний можно использовать пакет MySQL Benchmark Suite, расположенный в папке *sql-bench*. Обычно он включается в дистрибутив исходного кода. Тесты написаны на oPerl и используют модуль Perl DBI для обращения к серверу. Если у вас не установлен модуль Perl DBI или Perl, обратитесь к разделу «Installing Perl on Unix» онлайновой документации MySQL Reference Manual.

Для запуска оценочных испытаний используйте следующую команду:

```
./run-all-tests --server=mysql --cmp=mysql --user=root --socket=<socket>
```

Эта команда запускает полный набор стандартных тестов, записывает текущие результаты и сравнивает их с известными результатами выполнения этих тестов на сервере MySQL. На лист. 8-4 показан фрагмент результатов выполнения этой команды в системе с ограниченными ресурсами.

Лист. 8-4. Результаты тестирования сервера MySQL

```
cbell@cbell-mini:~/source/bzr/mysql-6.0-review/sql-bench$
Benchmark DBD suite: 2.15
Date of test:        2009-12-01 19:54:19
Running tests on:    Linux 2.6.28-16-generic i686
Arguments:           --socket=../mysql-test/var/tmp/mysqld.1.sock
Comments:
Limits from:         mysql
Server version:      MySQL 6.0.14 alpha debug log
Optimization:        None
Hardware:

alter-table: Total time: 77 wallclock secs
  ( 0.12 usr 0.05 sys + 0.00 cusr 0.00 csys = 0.17 CPU)
ATIS: Total time: 150 wallclock secs
  (20.22 usr 0.56 sys + 0.00 cusr 0.00 csys = 20.78 CPU)
big-tables: Total time: 135 wallclock secs
  (45.73 usr 1.16 sys + 0.00 cusr 0.00 csys = 46.89 CPU)
connect: Total time: 1359 wallclock secs
  (200.70 usr 30.51 sys + 0.00 cusr 0.00 csys = 231.21 CPU)
...
```

Функция benchmark

MySQL имеет встроенную функцию benchmark(), которую можно использовать для выполнения простых выражений и получения оценочных результатов. Лучше всего она подходит для оценки других функций и выражений, чтобы определить, являются ли они причиной задержек. Эта функция принимает два параметра: счетчик циклов и выражение для тестирования. Ниже показаны результаты выполнения 10 млн. итераций функции concat:

```
mysql> SELECT BENCHMARK(10000000, "SELECT CONCAT('te','s',' t')");
+--------------------------------------------------+
| BENCHMARK(10000000, "SELECT CONCAT('te','s',' t')") |
+--------------------------------------------------+
|                                                0 |
+--------------------------------------------------+
1 row in set (0.06 sec)
```

Возвращаемое значение — время, затраченное на выполнение оценочных тестов. В этом примере для выполнения итераций потребовалось 0,06 сек. Эту команду можно использовать для тестирования частей сложного запроса. Может оказаться, что проблема вызвана какой-то частью запроса и не связана с индексами.

Пакет для нагрузочного тестирования может оказаться очень полезным для диагностики сервера. Следует использовать их на серверах для создания базовой статистики о производительности, которую можно сравнивать

с примерами, включенными в пакет. Подробнее об этом см. в руководстве MySQL Reference Manual.

Мы обсудили различные инструменты для мониторинга MySQL и дали некоторые рекомендации по их применению. Теперь перейдем к рассмотрению одного из более сложных компонентов сервера MySQL — механизмов БД.

Производительность базы данных

Мониторинг производительности отдельной базы данных — одна из немногих областей, в которой специалисты из сообщества MySQL и сторонние разработчики значительно улучшили стандартную функциональность MySQL. В состав пакетов MySQL входят некоторые базовые инструменты, с помощью которых можно повышать производительность, но они предоставляют не так много возможностей. Из-за этого большинство администраторов баз данных MySQL используют различные приемы оптимизации реляционных запросов. Мы можем порекомендовать несколько отличных руководств, в которых подробно рассказывается о производительности баз данных и хорошо разъясняются основы их оптимизации:

- Refactoring SQL Applications, Stephane Faroult, Pascal L'Hermite, O'Reilly Media (*http://oreilly.com/catalog/9780596514976*);
- SQL and Relational Theory: How to Write Accurate SQL Code, C.J. Date, O'Reilly Media (*http://oreilly.com/catalog/9780596523084*);
- SQL Cookbook, Anthony Mollinaro, O'Reilly Media (*http://oreilly.com/catalog/ 9780596009762*).

Чтобы не повторять написанное о приемах оптимизации запросов, мы расскажем об инструментах, с помощью которых можно оптимизировать базы данных MySQL. Для демонстрации применения команды для оптимизации запросов мы будем использовать известную базу-образец. В следующем разделе мы приведем рекомендации по повышению производительности баз данных.

Оценка производительности базы данных

Обычно в состав СУБД входят средства профилирования и индексирования, предоставляющие статистическую информацию, которую можно использовать для настройки индексов. Хотя в MySQL есть некоторые базовые элементы, с помощью которых можно повышать производительность баз данных, бесплатных развитых инструментов профилирования нет.

Хотя базовая установка MySQL не включает специальные средства мониторинга усовершенствования баз данных, в пакете MySQL Enterprise Manager есть семейство функций мониторинга. Подробнее об этом рассказывается в гл. 13.

К счастью, MySQL предоставляет несколько простых инструментов, помогающих определить, в оптимальном ли состоянии находятся таблицы

и запросы. Это команды SQL EXPLAIN, ANALYZE TABLE и OPTIMIZE TABLE. Рассмотрим эти команды подробнее.

EXPLAIN

Команда EXPLAIN предоставляет информацию о том, как может быть выполнено выражение SELECT (для других выражений она не работает). Эта команда, являющаяся синонимом команды DESCRIBE из других СУБД, имеет следующий синтаксис:

```
[EXPLAIN | DESCRIBE] [EXTENDED] SELECT <параметры выборки>
```

Команды EXPLAIN и DESCRIBE можно также использовать для просмотра информации о столбцах или разделах таблицы. Ниже показан синтаксис для этой версии команды.

```
[EXPLAIN | DESCRIBE] [PARTITIONS SELECT * FROM] <имя_таблицы>
```

Команда EXPLAIN <имя_таблицы> имеет синоним SHOW COLUMNS FROM <имя_таблицы>.

Мы обсудим первый способ применения команды EXPLAIN — изучение команды SELECT с целью понять, как оптимизатор MySQL выполняет выражение. В результате мы получим поэтапный список операторов объединения, которые, по прогнозу оптимизатора, потребуются для выполнения выражения.

Обработка запросов с выражениями order-by и group-by пошагово не представляется.

Лучше всего использовать эту команду, чтобы определить, имеются ли для таблиц правильные индексы, позволяющие более точно находить строки-кандидаты. Кроме того, результаты можно использовать для тестирования различных параметров замещения, указанных для оптимизатора. Хотя это сложный и не рекомендуемый прием, в некоторых случаях запрос выполняется быстрее с определенными параметрами оптимизатора. Пример этого мы приведем далее в этом разделе.

Теперь рассмотрим примеры команды EXPLAIN в действии. Приведенные ниже примеры — это запросы к базе-примеру *sakila*, предназначенной для разработки и экспериментов. Скачать эту БД можно по адресу *http://downloads.mysql.com/docs/sakila-db.zip*. Диаграмму сущностей и связей, полученную при помощи MySQL Forge, можно скачать по адресу *http://forge.mysql.com/wiki/Image:SakilaSampleDB-0.8.png*.

Для ясности мы будем использовать параметр /G или вертикальное отображение.

Начнем с простого и, казалось бы, безобидного запроса. Допустим, мы хотим просмотреть все фильмы с рейтингом выше PG. Результаты будут содержать одну строку со следующими столбцами:

- *id* — номер выражения в порядке выполнения;
- *select_type* — тип выполненного выражения;
- *table* — таблица, используемая на данном шаге;
- *type* — тип использованного объединения;

Если этот столбец содержит значение ALL, выполняется полное сканирование таблиц. Старайтесь избегать таких операций, добавляя индексы или переписывая запрос. Если этот столбец содержит значение INDEX, выполняется полное сканирование индекса, что очень неэффективно. Подробнее о типах объединения и их последствиях см. в онлайновой документации MySQL Reference Manual.

- *possible_keys* — список доступных столбцов, если имеются доступные индексы;
- *key* — ключ, выбранный оптимизатором;
- *key_len* — длина ключа или используемая часть ключа;
- *ref* — ограничения или столбцы для сравнения;
- *rows* — ожидаемое число строк для обработки;
- *extra* — дополнительная информация от оптимизатора.

На лист. 8-5 показано, как оптимизатор MySQL выполняет это выражение.

Лист. 8-5. Простое выражение SELECT

```
mysql> EXPLAIN SELECT * FROM film WHERE rating > 'PG' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 892
        Extra: Using where
1 row in set (0.01 sec)
```

По этим выходным данным видно, что оптимизатор имеет только один этап для выполнения и не использует никакие индексы. Это понятно, так как мы не используем столбцы с индексами. Более того, даже несмотря на то, что есть выражение WHERE, оптимизатору все равно приходится выполнять полное сканирование таблиц. Это может быть правильным выбором, если учесть используемые столбцы и отсутствие индексов. Однако если выполнять этот запрос сотни тысяч раз, полное сканирование таблиц будет очень нерациональной тратой времени. В таком случае, глядя на результаты, мы видим, что добавление индекса пойдет на пользу. Добавим индекс и попробуем снова. На лист. 8-6 показан улучшенный план запроса.

Лист. 8-6. Улучшенный план запроса

```
mysql> ALTER TABLE film ADD INDEX film_rating (rating);
Query OK, 0 rows affected (0.42 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM film WHERE rating > 'PG' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ALL
possible_keys: film_rating
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 892
        Extra: Using where
1 row in set (0.00 sec)
```

Если вы уже заметили проблему, наберитесь терпения, скоро мы до нее дойдем.

Здесь мы видим, что запрос идентифицировал индекс (possible_keys), но все еще его не использует, так как поле key имеет значение NULL. Что мы можем сделать? В этом простом примере ожидаемое число строк для считывания — всего 892. Общее число строк — 1000, а результирующий набор будет содержать только 418 строк. Ясно, что запрос выполнится гораздо быстрее, если будет считывать только 42% строк.

Теперь посмотрим, сможем ли мы получить от оптимизатора какую-либо дополнительную информацию, используя ключевое слово EXTENDED. Это ключевое слово позволяет увидеть больше информации, используя команду SHOW WARNINGS. Эту команду следует выполнить сразу после команды EXPLAIN. Текст предупреждений описывает, как оптимизатор идентифицирует имена таблиц и столбцов в выражении, внутреннюю перезапись запроса, примененные правила оптимизатора и любые дополнительные сведения о выполнении. На лист. 8-7 показаны результаты использования ключевого слова EXTENDED.

Лист. 8-7. Использование ключевого слова EXTENDED для получения дополнительной информации

```
mysql> EXPLAIN EXTENDED SELECT * FROM film WHERE rating > 'PG' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ALL
possible_keys: film_rating
```

```
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 892
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS \G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select `sakila`.`film`.`film_id` AS `film_id`,
`sakila`.`film`.`title` AS `title`,`sakila`.`film`.`description` AS
`description`, `sakila`.`film`.`release_year` AS `release_year`,
`sakila`.`film`.`language_id` AS `language_id`, `sakila`.`film`.`original_
language_id` AS `original_language_id`, `sakila`.`film`.`rental_duration`
AS `rental_duration`, `sakila`.`film`.`rental_rate` AS `rental_rate`,
`sakila`.`film`.`length` AS `length`, `sakila`.`film`.`replacement_
cost` AS `replacement_cost`, `sakila`.`film`.`rating` AS `rating`,
`sakila`.`film`.`special_features` AS `special_features`,
`sakila`.`film`.`last_update` AS `last_update`
from `sakila`.`film` where (`sakila`.`film`.`rating` > 'PG')
1 row in set (0.00 sec)
```

В этот раз показано одно предупреждение (содержащее информацию от оптимизатора), показывающее переписанную форму запроса, включающую все столбцы и явную ссылку на столбец в выражении WHERE. К сожалению, чтобы сделать этот запрос еще более эффективным, требуются дополнительные изменения. Весьма вероятно, что вы будете встречать подобные запросы, требующие переделки, пересмотра частоты использования или (скорее всего) изменения таблицы для поддержки более эффективного индекса.

Посмотрим, что произойдет, когда мы выполним запрос определенного рейтинга вместо использования запроса диапазона. Мы увидим оптимизацию с индексом и без него. Результаты показаны на лист. 8-8.

Лист. 8-8. Удаление запроса диапазона

```
mysql> EXPLAIN SELECT * FROM film WHERE rating = 'R' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ref
possible_keys: film_rating
          key: film_rating
      key_len: 2
```

```
          ref: const
         rows: 195
        Extra: Using where
1 row in set (0.00 sec)

mysql> ALTER TABLE film DROP INDEX film_rating;Query OK, 0 rows affected
(0.37 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM film WHERE rating = 'R' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 892
        Extra: Using where
1 row in set (0.00 sec)
```

Теперь мы видим небольшое улучшение. Обратите внимание, первый план запроса на самом деле использует индекс и дает значительное улучшение по сравнению с планом, не имеющим индекса. Остается вопрос, почему оптимизатор не использует индекс? В этом случае мы использовали неуникальный индекс по полю перечисления. То, что казалось очень хорошей идеей, в действительности не дает никаких преимуществ для запросов диапазона перечислимых значений. Однако мы можем переписать запрос (даже несколькими способами), чтобы повысить производительность. Рассмотрим этот запрос еще раз.

Мы хотим получить список фильмов с рейтингом выше PG-13. Мы предполагаем, что рейтинг упорядочен, и перечислимое поле отражает этот порядок. В базе данных *sakila* поле рейтинга определено как имеющее значения (G, PG, PG-13, R, NC-17). Таким образом, похоже, что порядок сохранится, если мы примем индекс перечисления для каждого значения, соответствующего порядку (например, G = 1, PG = 2 и т. д.). Но вдруг порядок неверный или (как в нашем примере) список значений неполный?

В приведенном примере в список фильмов с рейтингом выше PG-13 должны входить фильмы с рейтингом R и NC-17. Вместо использования запроса диапазона давайте посмотрим, что сделает оптимизатор, если мы перечислим эти значения.

Помните, что мы удалили индекс, поэтому мы сначала попробуем выполнить запрос без индекса, а потом добавим индекс и посмотрим, будет ли какое-то улучшение. На лист. 8-9 показан улучшенный запрос.

Лист. 8-9. Улучшенный запрос без диапазона

```
mysql> EXPLAIN SELECT * FROM film WHERE rating = 'R' OR rating = 'NC-17' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 892
        Extra: Using where
1 row in set (0.00 sec)

mysql> ALTER TABLE film ADD INDEX film_rating (rating);
Query OK, 0 rows affected (0.40 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM film WHERE rating = 'R' OR rating = 'NC-17' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film
         type: ALL
possible_keys: film_rating
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 892
        Extra: Using where
1 row in set (0.00 sec)
```

Увы, такой вариант тоже не работает. Мы снова выполняем запрос по столбцу, имеющему индекс, который не может использоваться оптимизатором. Мы знаем, что оптимизатор может и будет использовать индекс для простой проверки равенства. Попробуем переписать запрос, используя объединение двух запросов (листинг 8-10).

Лист. 8-10. Запрос с использованием выражения UNION

```
mysql> EXPLAIN SELECT * FROM film WHERE rating = 'R' UNION
SELECT * FROM film WHERE rating = 'NC-17' \G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: film
         type: ref
possible_keys: film_rating
```

```
          key: film_rating
      key_len: 2
          ref: const
         rows: 195
        Extra: Using where
*************************** 2. row ***************************
           id: 2
  select_type: UNION
        table: film
         type: ref
possible_keys: film_rating
          key: film_rating
      key_len: 2
          ref: const
         rows: 210
        Extra: Using where
*************************** 3. row ***************************
           id: NULL
  select_type: UNION RESULT
        table: <union1,2>
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra:
3 rows in set (0.00 sec)
```

Получилось! Теперь мы имеем план запроса, использующий индекс и обрабатывающий гораздо меньше строк. Из результатов команды EXPLAIN мы видим, что оптимизатор выполняет каждый запрос в отдельности (этапы выполнения показаны в строках с 1 по n) и объединяет результаты на последнем этапе.

В MySQL есть переменная состояния сеанса last_query_cost, показывающая стоимость выполнения последнего запроса. Используйте это значение для сравнения разных планов для одного и того же запроса. Например, проверяйте это значения после каждого выполнения команды EXPLAIN. Запрос с наименьшей стоимостью считается наиболее эффективным (требующим меньше времени на выполнение). Значение 0 говорит о том, что запрос не выполнялся.

Может показаться, что в этом упражнении затрачено много труда, а преимущество совсем мало, но представьте, что приложения выполняют множество таких запросов, и никто не видит, что их эффективность низкая. Обычно мы замечаем запросы такого типа, только когда число строк достаточно велико. В базе данных *sakila* всего 1000 строк, но если бы их было несколько десятков миллионов?

Кроме команды EXPLAIN, в стандартном дистрибутиве MySQL нет других инструментов для профилирования запросов. В главе «Optimization» онлайнового руководства MySQL Reference Manual приведено много советов для опытных администраторов баз данных по повышению производительности запросов различного типа.

ANALYZE TABLE

Оптимизатор MySQL, как и большинство традиционных оптимизаторов, использует статистическую информацию о таблицах для анализа оптимального плана выполнения запроса. Эта статистика, кроме прочего, включает сведения об индексах, распределении значений и структуре таблицы.

Команда ANALYZE TABLE пересчитывает распределение ключей для одной или нескольких таблиц. Полученная информация определяет порядок объединения. Эта команда имеет следующий синтаксис:

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE <список_таблиц>
```

Можно обновлять распределение ключей для таблиц MyISAM и InnoDB. Это очень важно отметить, так как эта команда работает не со всеми механизмами хранилища. Однако все механизмы хранилища, поддерживающие индексы, должны сообщать оптимизатору статистику по количеству элементов. Некоторые механизмы хранилища имеют собственную специфическую статистику. На лист. 8-11 показано типичное выполнение команды. Если выполнить команду для таблицы без индексов, никакого эффекта не будет, но сообщение об ошибке не появится.

Лист. 8-11. Анализ таблицы для обновления распределения ключей

```
mysql> ANALYZE TABLE film;
+-------------+---------+----------+----------+
| Table       | Op      | Msg_type | Msg_text |
+-------------+---------+----------+----------+
| sakila.film | analyze | status   | OK       |
+-------------+---------+----------+----------+
1 row in set (0.00 sec)
```

В этом примере мы видим, что анализ завершен, и никаких необычных условий нет. Если бы во время выполнения команды произошли какие-либо необычные события, в поле Msg_type содержало бы значение info, error или warning, а в поле Msg_text были бы указаны дополнительные сведения о событии. Всегда выясняйте, в чем дело, если получили любые результаты, кроме status и OK.

Состояние индексов можно просмотреть при помощи команды SHOW INDEX. На лист. 8-12 показаны выходные данные для таблицы film. В данном случае нам интересно кардинальное число каждого индекса, которое является ожидаемым числом уникальных значений индекса. Для краткости мы опустили другие столбцы. Подробнее об этой команде см. в руководстве MySQL Reference Manual.

Лист. 8-12. Индексы для таблицы film

```
mysql> SHOW INDEX FROM film \G
*************************** 1. row ***************************
       Table: film
  Non_unique: 0 Key_name: PRIMARY
Seq_in_index: 1
 Column_name: film_id
   Collation: A
 Cardinality: 1028
...
*************************** 2. row ***************************
       Table: film
  Non_unique: 1
    Key_name: idx_title
Seq_in_index: 1 Column_name: title
   Collation: A
 Cardinality: 1028
...
*************************** 3. row ***************************
       Table: film
  Non_unique: 1
    Key_name: idx_fk_language_id
Seq_in_index: 1
 Column_name: language_id
   Collation: A
 Cardinality: 2
...
*************************** 4. row ***************************
       Table: film
  Non_unique: 1
    Key_name: idx_fk_original_language_id
Seq_in_index: 1
 Column_name: original_language_id
   Collation: A
 Cardinality: 2
...
*************************** 5. row ***************************
       Table: film
  Non_unique: 1
    Key_name: film_rating
Seq_in_index: 1
 Column_name: rating
   Collation: A
 Cardinality: 11
    Sub_part: NULL
      Packed: NULL
        Null: YES
```

```
  Index_type: BTREE
     Comment:
5 rows in set (0.00 sec)
```

Ключевые слова LOCAL и NO_WRITE_TO_BINLOG предотвращают запись команды в двоичный журнал (а значит, и ее распространение в топологии репликации). Это может быть очень полезно, если вы хотите поэкспериментировать или выполнить настройку, когда включена репликация данных, или если хотите исключить этот этап из двоичного журнала и не воспроизводить во время восстановления до состояния на определенный момент времени (PITR).

Эту команду следует выполнять, когда в таблице произошли значительные изменения (например, массивная загрузка данных). Система должна иметь блокировку чтения таблицы на время выполнения этой операции.

OPTIMIZE TABLE

Таблицы, часто обновляющиеся вставками и удалениями, могут быстро фрагментироваться и, в зависимости от механизма БД, в них могут появляться промежутки неиспользуемого пространства или неоптимальные структуры хранения. Значительная фрагментация может ухудшить производительность, особенно при сканировании таблиц.

Команда OPTIMIZE TABLE реструктурирует структуры данных в одной или нескольких таблицах. Это дает особенно заметные преимущества для форматов строк с переменной длиной полей (строк). Синтаксис команды OPTIMIZE TABLE показан ниже:

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE <список_таблиц>
```

Эту команду можно использовать для таблиц MyISAM и InnoDB. Это очень важно отметить, так как эта команда работает не со всеми механизмами хранилища. Если таблицу нельзя реорганизовать (например, нет записей переменной длины или нет фрагментации), эта команда воссоздаст таблицу и обновит статистику. На лист. 8-13 показаны выходные данные этой команды.

Лист. 8-13. Команда OPTIMIZE TABLE

```
mysql> OPTIMIZE TABLE film \G
*************************** 1. row ***************************
   Table: sakila.film
      Op: optimize
Msg_type: note
Msg_text: Table does not support optimize, doing recreate + analyze instead
*************************** 2. row ***************************
   Table: sakila.film
      Op: optimize
Msg_type: status
Msg_text: OK
2 rows in set (0.44 sec)
```

Здесь мы видим две строки результатов. В первой сообщается, что команда OPTIMIZE TABLE не может быть выполнена, и вместо этого будет воссоздана таблица и выполнена команда ANALYZE TABLE. Во второй строке приведены результаты команды ANALYZE TABLE.

Как и для команды ANALYZE TABLE, описанной выше, значения info, error или warning в поле Msg_type указывают на любые необычные события, произошедшие во время выполнения команды. В поле Msg_text приводится дополнительная информация о событии. Всегда выясняйте, в чем дело, если получили любые результаты, кроме status и OK.

Ключевые слова LOCAL и NO_WRITE_TO_BINLOG предотвращают запись команды в двоичный журнал (а значит, и ее распространение в топологии репликации). Это может быть очень полезно, если вы хотите поэкспериментировать или выполнить настройку, когда включена репликация данных, или если хотите исключить этот этап из двоичного журнала и не воспроизводить во время восстановления до состояния на определенный момент времени (PITR).

Эту команду следует выполнять, когда в таблице произошли значительные изменения (например, большое количество вставок или удалений). Эта операция предназначена для перегруппировки данных для получения оптимальной структуры и может выполняться дольше, чем ожидается. Эту операцию лучше выполнять во время низкой загруженности системы.

При использовании InnoDB, особенно когда есть вторичные индексы (которые обычно становятся фрагментированными), вы можете не заметить никаких улучшений или столкнуться с долгим выполнением этой операции, если не используете параметр InnoDB «быстрое создание индекса» (fast index create).

Рекомендации по оптимизации баз данных

Как говорилось ранее, существует много замечательных примеров, приемов и рекомендаций от лучших в мире специалистов по производительности баз данных. Вместо того, чтобы судить или предлагать конкретные инструменты и приемы, мы обсудим общие рекомендации по повышению производительности БД. Более подробную информацию можно найти в текстах, на которые мы ссылались выше.

Используйте индексы редко, но эффективно

Специалисты по базам данных понимают важность индексов и их влияние на повышение производительности. Использование команды EXPLAIN часто бывает лучшим способом определения необходимых индексов. Тогда как проблема с нехваткой индексов понятна, следует отметить, что слишком большое число индексов тоже может снижать производительность.

Как вы видели при изучении команды EXPLAIN, можно иметь слишком много индексов или создать неиспользуемые индексы. Каждый индекс уве-

личивает нагрузку для каждой операции вставки и удаления. В некоторых случаях наличие слишком большого количества индексов с широким распределением может значительно снизить производительность вставки и удаления. Кроме того, это может замедлять репликацию и восстановление.

Следует периодически проверять индексы, выясняя, используются ли они, и удалять все неиспользуемые, редко используемые и имеющие широкое распределение. Часто некоторые проблемы с широким распределением можно решить при помощи нормализации.

Выполняйте нормализацию, но не чрезмерно

Те, кто изучал информатику и связанные дисциплины, наверное, помнят, чего стоило запомнить нормальные формы, описанные К. Дж. Дейтом и другими. Мы не будем пересматривать этот материал, а обсудим, что бывает, если следовать этим принципам слишком усердно.

Нормализация (во всяком случае, до третьей нормальной формы) — известный стандартный прием. Однако бывают ситуации, в которых лучше нарушить правила нормализации.

Побочным следствием нормализации часто является использование таблиц поиска. Т. е. создается специальная таблица, содержащая связанную информацию, часто используемую в других таблицах. Однако частое обращение к таблицам поиска с ограниченным распределением (всего несколько строк или ограниченное число строк с небольшими значениями) может снизить производительность. В этом случае каждый раз, когда запрашивается информация, для получения всех данных необходимо выполнить объединение. Объединение — затратная операция, и ее частое выполнение может увеличивать нагрузку. Для решения этой проблемы производительности можно хранить данные в перечислимых полях, а не в таблицах поиска. Например, вместо создания таблицы, хранящей цвета волос (кто-то может не согласиться, но в действительности цветов волос не так много), можно использовать перечислимое поле и избежать выполнения объединений.

Еще одна потенциальная проблема — вычисляемые поля. Обычно мы не храним данные, получаемые на основе других данных, такие как налог с продаж или сумма нескольких столбцов. Вместо этого выполняются вычисления, либо во время получения данных, либо при просмотре в приложении. Это не проблема, если вычисления простые или редко выполняются. Но что будет, если вычисления сложные и выполняются часто? Возможно, что в этом случае будет тратиться много времени на вычисления. Один из способов решения этой проблемы — использовать триггер для вычисления значения и сохранять его в таблице. Хотя технически это дублирование данных (громкое «нет» теоретикам нормализации!), это может повысить производительность, когда требуется много вычислений.

Используйте подходящий механизм БД

Одна из самых полезных особенностей MySQL — поддержка различных механизмов БДа. Механизмы хранилища управляют хранением и извлечением данных. MySQL поддерживает несколько механизмов, каждый из которых имеет уникальный набор функций и особенностей. Это позволяет разработчикам БД настраивать производительность, выбирая механизм БД, наиболее подходящий для конкретного приложения. Например, для среды, требующей контроля транзакций для активно использующихся баз данных следует выбрать механизм БД, поддерживающий транзакции (да, не все механизмы хранилища MySQL поддерживают транзакции). Если представления или таблицы часто запрашиваются, но почти никогда не обновляются (например, таблицы поиска), можно использовать механизм БД, хранящий данные в памяти для быстрого доступа.

Недавние изменения в MySQL позволили превратить некоторые механизмы хранилища в подключаемые модули, и в некоторых дистрибутивах MySQL по умолчанию включены только определенные механизмы хранилища. Чтобы выяснить, какие механизмы хранилища включены, выполните команду SHOW ENGINES. На лист. 8-14 показано, какие механизмы хранилища включены в типичной установке.

Лист. 8-14. Механизмы хранилища

```
mysql> SHOW ENGINES \G
*************************** 1. row ***************************
      Engine: InnoDB
     Support: YES
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 2. row ***************************
      Engine: MyISAM
     Support: DEFAULT
     Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it
disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
```

```
*************************** 4. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
8 rows in set (0.00 sec)
```

Здесь перечислены все известные механизмы хранилища, указано, установлены ли они (установлены, если Support = Yes), какие особенности имеют и поддерживают ли транзакции, распределенные транзакции (XA) и *точки сохранения*.

Точка сохранения — именованное событие, которое можно использовать как транзакцию. Можно задать точку сохранения и либо освободить ее (удалить), либо откатить изменения до этой точки. Подробнее см. в онлайновом руководстве MySQL Reference Manual.

Такой большой выбор механизмов БД может смутить при проектировании базы данных. Вкратце опишем эти механизмы, указав, для каких задач их лучше применять. Механизм БД для таблицы можно выбрать при помощи параметра ENGINE выражения CREATE, а изменить — при помощи команды ALTER TABLE:

```
CREATE TABLE t1 (a int) ENGINE=InnoDB;
ALTER TABLE t1 ENGINE=MEMORY;
```

Механизм InnoDB — основной механизм, поддерживающий транзакции. Когда требуется поддержка транзакции, всегда следует выбирать его; в настоящий момент это единственный механизм БД, имеющий такую поддержку. Существуют механизмы БД сторонних производителей, находящиеся на различных стадиях разработки, которые могут поддерживать транзакции, но единственный «стандартный» вариант — InnoDB. Интересно, что все индексы в InnoDB являются B-деревьями, в которых записи индекса хранятся на страницах-листьях дерева. InnoDB — выбор для высоконадежных систем и систем, использующих транзакции.

Механизмом БД по умолчанию является MyISAM. Этот механизм будет использоваться, если в выражении CREATE не указать параметр ENGINE. MyISAM часто используется в архивах, электронной коммерции и промышленных приложениях. MyISAM использует сложные механизмы кэширования и индексации, повышающие производительность извлечения и индексации данных. MyISAM — отличный выбор, когда нужно создать хранилище для приложений, требующих быстрого извлечения данных и не использующих транзакции.

Blackhole — очень интересный механизм БД. Он вообще ничего не хранит. В действительности, он является тем, чем называется (blackhole — черная дыра): данные уходят в него, но никогда не возвращаются. Если без шуток, то этот механизм БД удовлетворяет очень специфическую потребность. Если включено ведение двоичного журнала, команды SQL записываются в журналы, а Blackhole служит агентом ретрансляции (или прокси) в топологии репликации. Агент ретрансляции обрабатывает данные с главного сервера и передает их подчиненным серверам, но ничего не сохраняет. Этот механизм БД удобен, когда требуется проверить, записывает ли приложение данные, но в действительности записывать что-либо на диск нежелательно.

Механизм БД CSV может создавать, читать и записывать файлы со значениями, разделенными запятой (comma-separated value, CSV), как таблицы. Он лучше всего подходит для быстрого экспорта структурированных данных в электронные таблицы. Этот механизм БД не предоставляет какого-либо механизма индексации и имеет некоторые проблемы с хранением и преобразованием значений даты-времени (несоответствие региональным параметрам во время запросов). Механизм БД CSV лучше всего использовать, когда требуется разрешить другим приложениям совместно использовать данные или обмениваться информацией в общем формате. Учитывая,

что этот механизм не особенно эффективен при хранении данных, не используйте его часто.

Механизм БД CSV используется для записи файлов журналов. Например, журналы резервного копирования являются файлами CSV, которые можно открыть в других приложениях, использующих протокол CSV (но не тогда, когда сервер работает).

Механизм БД Memory (иногда его называют HEAP) хранит данные в оперативной памяти и использует механизм хэширования для извлечения часто используемых данных. Это значительно ускоряет получение данных. Обращение к данным происходит так же, как и в других механизмах БД, но эти данные хранятся в памяти и действительны только в течение сеанса MySQL, а при завершении работы удаляются. Такой механизм БД хорошо подходит для ситуаций, когда статические данные часто запрашиваются и редко изменяются (например, таблицы поиска). В качестве примеров можно привести почтовые индексы, названия стран и регионов, списки категорий и т.д. Механизм БД Memory можно также использовать для баз данных, использующих мгновенные снимки для доступа к распределенным или историческим данным.

Механизм БД Federated создает единую табличную ссылку из нескольких систем баз данных. Этот механизм позволяет связывать таблицы, расположенные на разных серверах баз данных. Он аналогичен таблицам связанных данных, доступным в других СУБД. Его лучше всего использовать в распределенных средах и киосках данных. Наиболее интересная особенность этого механизма БД — он не перемещает данные и не требует, чтобы удаленные таблицы использовали одинаковые механизмы хранилища.

Механизм БД Federated в настоящий момент отключен в большинстве дистрибутивов MySQL. Подробнее см. в онлайновой документации MySQL Reference Manual.

Механизм БД Archive может хранить большие объемы данных в сжатом виде. Его лучше всего использовать для хранения и извлечения больших объемов редко запрашиваемых архивных или исторических данных. Индексы не поддерживаются, и единственный метод доступа — сканирование таблиц. Таким образом, механизм БД Archive не следует использовать для обычных задач хранения и извлечения данных.

Механизм БД Merge (MRG_MYISAM) может инкапсулировать набор таблиц MyISAM с одинаковой структурой (схемой или макетом таблиц) и ссылаться на него как на одну таблицу. Таблицы разделяются по расположению, но никакой другой механизм разделения не используется. Все таблицы должны находиться на одном сервере (но не обязательно в одной БД).

Когда для объединенной таблицы выполняется команда DROP, удаляется только спецификация объединения, а исходные таблицы остаются.

Главное достоинство механизма БД Merge — скорость. Он позволяет разделять крупную таблицу на несколько маленьких таблиц на разных дисках,

объединять их, используя спецификацию объединенной таблицы, и обращаться к ним одновременно. Операции поиска и сортировки будут выполняться гораздо быстрее, так как каждая таблица содержит меньше данных для обработки. Также более эффективным становится восстановление данных, так как быстрее и проще восстановить несколько маленьких таблиц, чем одну большую. К сожалению, такая конфигурация имеет несколько недостатков:

- для формирования объединенной таблицы необходимо использовать идентичные таблицы MyISAM;
- операция замены не разрешена;
- индексы гораздо менее эффективны, чем для одной таблицы.

Механизм БД Merge лучше всего использовать для приложений, использующих очень крупные базы данных (very large database, VLDB), таких как хранилища данных, хранящие данные в нескольких таблицах одной или нескольких баз данных. Кроме того, его можно использовать для решения проблем с разделением, когда требуется выполнить горизонтальное разделение, но вы не хотите связываться со сложностью параметров разделения таблиц.

Понятно, что при таком большом выборе можно случайно выбрать механизм, ухудшающий производительность или не позволяющий реализовать некоторые решения. Например, если не указать механизм БД при создании таблицы, будет использоваться механизм БД, выбранный по умолчанию. Если вручную не указать механизм БД, используемый по умолчанию, он может изменяться на разных платформах (на некоторых платформах это MyISAM). Из-за этого может пропасть возможность оптимизации таблиц поиска или функции приложения будут ограничены из-за отсутствия поддержки транзакций. Поэтому при проектировании и настройке баз данных следует выделять время на анализ доступных механизмов БД.

Используйте представления для ускорения получения результатов при помощи кэша запросов

Представления — очень удобный способ инкапсуляции сложных запросов, облегчающий работу с данными. Предоставления можно использовать для ограничения данных как по вертикали (меньше столбцов), так и по горизонтали (выражение WHERE в команде SELECT). Оба способа очень удобны и, конечно, сложные представления используют их оба для ограничения возвращаемого пользователю набора данных, для сокрытия определенных таблиц или для эффективного выполнения объединений.

Использование представлений для ограничения возвращаемых столбцов может помочь в тех случаях, о которых вы не думали. Это не только снижает объем обрабатываемых данных, но и помогает избежать затратных операций SELECT *, которые пользователи любят выполнять, особо не задумываясь. Если выполняется много таких операций, приложение обрабатывает гораздо больше данных, чем требуется, что может снизить производительность не

только приложения, но и сервера, и, что важнее, может снизить пропускную способность сети. Для решения таких проблем можно ограничивать данные и скрывать доступ к базовым таблицам, чтобы избавить пользователей от искушения выполнить обращение к базовой таблице напрямую.

Представления, ограничивающие число возвращаемых строк, также помогают снизить нагрузку на сеть и могут повысить производительность приложений. Запросы такого типа также защищают от выполнения пользователями запросов SELECT *. Такой способ использования представлений требует немного больше планирования, так как цель здесь — создать значимое подмножество данных. Для этого необходимо проверить требования к базе данных и понять, какие запросы выполняются для формирования корректных выражений `WHERE`.

Чуть сложнее создавать представления с комбинациями вертикальных и горизонтальных ограничений, но это может гарантировать, что приложение оперирует только необходимыми данными. Чем меньше данных запрашивается, тем больше полезных данных может обработать приложение за то же время.

Возможно, лучший способ использования представлений — устранение плохо сформированных объединений. Это особенно актуально, когда таблицы имеют сложную нормализованную схему. Пользователи могут не понимать, как комбинировать таблицы для получения нужного набора данных. В действительности, большая часть работы, выполняемой администраторами баз данных в попытках повышения производительности, — это исправление плохо сформированных объединений. Иногда это может быть очень просто, например, как устранение некоторых строк из операции объединения, но чаще всего время отклика повышается значительно.

Кроме этого, представления могут быть полезны при использовании кэша запросов MySQL. Кэш запросов хранит результаты часто используемых запросов. Использование представлений, предоставляющих стандартизированные наборы результатов, может повысить вероятность того, что результаты будут кэшированы и, соответственно, быстрее извлечены при повторном обращении.

Потратив некоторые усилия на проектирование и используя представления, можно повысить производительность. Выясните, какой объем данных перемещается (число столбцов и число строк), и проверьте приложение на наличие запросов, использующих объединения. Сформируйте представления, ограничивающие данные, и найдите наиболее эффективные способы объединения, также включив их в представления. Представьте, насколько спокойнее вам будет, если пользователи будут выполнять эффективные объединения.

Используйте ограничения

Использование ограничений — еще один инструмент в вашем арсенале для решения проблем производительности. Мы не будем размышлять о недо-

статках ограничений, а посоветуем вам относиться к установке ограничений как к стандартной практике, а не возможному позднему дополнению.

В MySQL доступны следующие типы ограничений:

- уникальные индексы;
- первичные ключи;
- внешние ключи;
- перечислимые значения;
- наборы;
- значения по умолчанию;
- параметр NOT NULL.

Мы уже обсуждали использование индексов и проблемы при чрезмерном их использовании. Индексы помогают ускорить извлечение информации, позволяя системе быстрее сохранять и находить данные.

Уникальный индекс — это индекс по одному полю таблицы, гарантирующий, что в таблице нет дубликатов, если используется совместно с ограничением NOT NULL, требующим указания значения. Таким образом, только одна строка может иметь одно значение из индекса. Используйте уникальные индексы для полей, дублирование которых хотите запретить, например, для последовательных и порядковых номеров, номеров социального страхования и т. д. Таблица может иметь один или несколько уникальных индексов.

Первичный ключ также считается уникальным индексом, но он уникально идентифицирует каждую строку таблицы и запрещает дублирование. Первичные ключи создаются при нормализации и формируют столбцы, по которым выполняется объединение таблиц.

Один из наиболее распространенных первичных ключей — автоматически генерируемый последовательный номер (называемый суррогатным), уникально идентифицирующий строку. В MySQL есть параметр AUTO_INCREMENT, сообщающий системе, что требуется генерировать уникальные значения. Некоторые теоретики считают использование суррогатных ключевых значений нежелательным, так как первичный ключ должен состоять из существующих полей, а не создаваться искусственно. Мы не будем говорить, что суррогатные ключи никогда не должны использоваться, но все же посоветуем делать это как можно реже. Если вы используете AUTO_INCREMENT почти в каждой таблице, вы, возможно, злоупотребляете этой функцией.

Внешние ключи также создаются при нормализации. Они позволяют формирование отношений родитель-дитя или главный-дополнительный, в которых строка из одной таблицы является главной, а одна или несколько строк из другой таблицы содержит дополнительные сведения о главной записи. Внешние ключи позволяют выполнять каскадные операции, при которых удаление главной строки ведет к удалению всех дополнительных строк.

В настоящий момент внешние ключи поддерживает только InnoDB.

Мы уже обсуждали использование *перечислимых значений* для замены небольших таблиц поиска. Однако перечислимые значения могут быть и средством для настройки производительности, так как текст таких значений сохраняется только дин раз — в структурах заголовков таблиц. То, что сохраняется в строках, является числовыми ссылочными значениями, формирующими индекс (массив индекса) перечислимого значения. Таким образом, списки перечислимых значений могут экономить пространство и немного повышать эффективность переходов по данным. Тип перечислимого поля допускает одно и только одно значение.

Использование *наборов* в MySQL аналогично использованию перечислимых значений. Однако этот тип поля позволяет хранение одного или нескольких значений из набора. Наборы можно использовать для хранения информации, представляющей атрибуты данных, вместо использования отношений главный-дополнительный. Это не только экономит место в таблице (наборы значений — это побитовые комбинации), но и избавляет от необходимости обращения к другой таблицы для получения значений.

Использование параметра DEFAULT — отличный способ решить проблемы с некорректно сконструированными данными. Например, если есть числовое поле, представляющее значения, используемые для вычислений, можно сделать так, что, когда поле неизвестно, будет задаваться значение по умолчанию. Значения по умолчанию можно указывать для большинства типов данных. Также можно использовать значения по умолчанию для полей даты и времени, чтобы избежать проблем с обработкой недействительных значений даты и времени. Еще важнее то, что значения по умолчанию позволяют приложению не указывать значения (и не использовать менее надежный способ — запрос значений у пользователей), снизив объем данных, отправляемый на сервер.

Следует также учитывать возможность использования параметра NOT NULL при определении полей, которые обязательно должны иметь значения. Если попытаться ввести в столбец с примененным параметром NOT NULL данные, не содержащие значений, команда INSERT завершится с ошибкой. Это избавляет от проблем с целостностью данных и гарантирует, то все важные поля имеют значения.

Используйте команды EXPLAIN, ANALYZE и OPTIMIZE

Мы уже обсуждали возможности этих команд. Здесь мы говорим о них, чтобы напомнить о том, что они жизненно важны для диагностики и настройки. Может использовать их как угодно часто, но учитывайте предназначение каждой команды. Команды ANALYZE и OPTIMIZE используйте, когда это нужно, но не рутинно. Мы встречали администраторов, выполняющих эти команды каждую ночь, и, хотя в некоторых случаях это может быть обоснованным решением, чаще всего это ведет только к созданию ненужных копий таблиц (как мы видели в приведенных примерах). Понятно, что регулярное

копирование данных может быть напрасной тратой времени и снижать доступность данных во время выполнения других операций.

Мы рассказали, как выполнять мониторинг и повышать производительность баз данных. Далее мы перейдем к более подробному рассмотрению одной из наиболее полезных и популярных функций — репликации. В следующем разделе мы обсудим мониторинг и повышение производительности репликации в MySQL. Мы поставили эту тему ближе к концу, поскольку, как вы убедитесь, прежде чем приступать к повышению производительности репликации, необходимо иметь хорошо функционирующий сервер с хорошо работающими базами данных и запросами.

Рекомендации по повышению производительности

Подробное описание способов диагностики и повышения производительности баз данных описывается в посвященных этому работах и занимает немало страниц.

Для полноты и в качестве общей справки перечислим рекомендации по исправлению аномалий производительности. Можете использовать эти сведения в качестве руководства для дальнейших поисков. Рекомендации сгруппированы по симптомам.

Все работает медленно

Когда система в целом функционирует плохо, следует сфокусировать усилия на том, как она работает, начиная с операционной системы. Для поиска и решения проблем выполняйте следующее:

- проверьте оборудование на наличие проблем;
- модернизируйте оборудование (например, добавьте памяти);
- рассмотрите возможность переноса данных на изолированные диски;
- проверьте правильность настройки операционной системы;
- рассмотрите возможность переноса некоторых приложений на другие серверы;
- рассмотрите возможность масштабирования при помощи репликации;
- настройте сервер для повышения производительности.

Медленные запросы

Запросы, оказывающиеся в журнале медленных запросов, или запросы, на которые жалуются пользователи или разработчики, можно улучшить, выполнив следующее:

- нормализуйте схему базы данных;
- используйте команду EXPLAIN для выявления недостающих или некорректных индексов;

- используйте функцию benchmark() для тестирования частей запроса;
- попробуйте переписать запрос;
- используйте представления для стандартизации запросов;
- включите кэш запросов.

Подчиненные серверы репликации не записывают реплицированные запросы в журнал медленных запросов, независимо от того, записаны ли запросы в этот журнал на главном сервере.

Медленная работа приложений

Если приложение показывает признаки плохой производительности, следует проверить его компоненты, чтобы локализовать проблему. Возможно, источником проблемы является только один модуль, но иногда дело может быть серьезнее. Для выявления и решения проблем с производительностью приложений выполняйте следующее:

- включите кэш запросов;
- пересмотрите выбор механизма БД;
- убедитесь, что проблема не в сервере и операционной системе;
- выполните оценочные испытания приложений и сравните результаты с базовыми;
- проверьте внутренние (написанные в приложении) запросы и повысьте их производительность;
- «разделяйте и властвуйте» — проверяйте по одному компоненту за раз;
- используйте разбиение на разделы для распределения данных;
- проверьте фрагментацию индексов.

Медленная репликация

Как говорилось ранее, проблемы производительности, связанные с репликацией, обычно сводятся к проблемам с базой данных и сервером. Для диагностики производительности репликации выполняйте следующее:

- убедитесь, что сеть работает на максимуме производительности;
- убедитесь, что серверы должным образом настроены;
- оптимизируйте базы данных;
- ограничьте обновления на главном сервере;
- разделите операции чтения по нескольким подчиненным серверам;
- регулярно выполняйте обслуживание журналов (двоичного журнала и журнала ретрансляций);
- используйте сжатие данных, передаваемых по сети, если пропускная способность недостаточна;
- используйте включающее и исключающее журналирование, чтобы уменьшить объем реплицируемых данных.

Заключение

Сервер MySQL имеет много компонентов, мониторинг которых можно выполнять. Мы обсудили базовые команды SQL, применяемые для мониторинга сервера, рассмотрели утилиту командной строки mysqladmin, средства для проведения оценочных испытаний, а также утилиты MySQL Administrator и MySQL Query Browser. Кроме того, мы привели некоторые рекомендации по повышению производительности баз данных.

Теперь вы знаете, как выполнять мониторинг операционной системы и серверов MySQL, настраивать базы данных и проводить оценочные испытания, и имеете все необходимые инструменты и знания, чтобы успешно настраивать серверы для получения оптимальной производительности.

Джоэл блаженно улыбнулся: он только что закончил отчет о проблеме с вложенным запросом, возникшей у Сьюзан. Чтобы найти причину, потребовалось несколько часов копаться в файлах журналов, но когда он объяснил разработчикам, сколько нагрузки генерирует этот запрос, те согласились переписать его и использовать таблицу поиска, хранящуюся в оперативной памяти. Джоэл чувствал, что его босс будет рад его находчивости. Только он отправил сообщение, как появился г-н Саммерсон.

— Джоэл!

Джоэл подпрыгнул, несмотря на то, что ждал Саммерсона, и выпалил:

— Я решил проблему в отделе маркетинга, сэр!

— Замечательно! С нетерпением жду отчета о том, как ты это сделал.

Джоэл не был уверен, поймет ли Саммерсон техническую часть отчета, и знал, что начальник будет дотошно расспрашивать, если сразу все ему не объяснить.

Г-н Саммерсон кивнул и вышел. Джоэл же открыл сообщение от Фила из Сиэтла (тот жаловался на репликацию), и скоро понял, что там проблема гораздо серьезнее, чем на сервере, с которым он только что работал.

ГЛАВА 9

Мониторинг механизмов БД

Джоэл наслаждался утренним кофе, когда зазвонил телефон. Это озадачило его, так как до сих пор телефон всегда молчал. Он поднял трубку и услышал звук мотора. Джоэл подумал, что ошиблись номером, и неуверенно сказал:

— Алло?

— Джоэл! Рад, что застал тебя, — г-н Саммерсон звонил из машины.

— Да, сэр.

— Я еду в аэропорт, лечу в Сиэтл на встречу с менеджерами по продажам. Хочу, чтобы ты взглянул на новое приложение для работы с базами данных. Разработчики из Сиэтла говорят, что нам надо будет менять конфигурацию для повышения производительности.

Джоэл ожидал чего-то вроде этого. Он знал немного о MyISAM и InnoDB, но не был знаком с мониторингом и настройкой производительности этих механизмов.

— Я могу посмотреть.

— Отлично! Благодарю, Джоэл. Я напишу позже.

Связь оборвалась прежде, чем Джоэл смог ответить. Он допил кофе и начал читать о механизмах БД в MySQL

Вы уже знаете, как оценить производительность серверов, но как оценить производительность механизмов БД? При развертывании баз данных, использующих транзакции, или при необходимости настроить хранилище для оптимальной работы с быстрыми запросами потребуется провести мониторинг БД. В этой главе мы поговорим о мониторинге, уделив основное внимание мониторингу и повышению производительности двух наиболее популярных механизмов БД: MyISAM и InnoDB.

Наличие нескольких взаимозаменяемых механизмов БД — уникальная особенность MySQL, предоставляющая широкие возможности. Несмотря на то что встроенных средств или даже каких-либо стандартизированных способов мониторинга механизмов БД не существует, мониторинг и настройку для оптимальной производительности наиболее популярных механизмов все-таки можно выполнять.

В этом разделе мы рассмотрим механизмов БД MyISAM и InnoDB, расскажем, как проводить мониторинг каждого из них и дадим практические советы по повышению производительности.

MyISAM

В механизме БД MyISAM не так много компонентов, пригодных для мониторинга. Причина этого в том, что этот механизм создавался для веб-приложений, и основное его предназначение — обработка быстрых запросов, поэтому единственное, что можно настраивать на сервере — кэш ключей. Это не значит, что для повышения производительности больше ничего нельзя сделать. Наоборот, сделать можно многое. Большинство доступных возможностей разделяется на три группы: оптимизация БД на диске, эффективное использование памяти посредством мониторинга и настройки кэша ключей и настройка таблиц для максимальной производительности.

Мы не будем обсуждать общие аспекты этих возможностей, а опишем стратегию повышения производительности, разбив ее на следующие этапы:

- оптимизация дискового файла БД;
- настройка таблиц;
- использование утилит MyISAM;
- хранение таблицы в порядке индексации;
- сжатие таблиц;
- дефрагментация таблиц;
- мониторинг кэша ключей;
- предварительная загрузка кэшей ключей;
- использование нескольких кэшей ключей;

мдругие действия.

В следующих разделах мы вкратце обсудим каждый из этих этапов.

Оптимизация дискового файла БД

Оптимизация дискового пространства для MyISAM относится больше к настройке системы, чем к настройкам параметров MyISAM. Каждая таблица MyISAM хранится в отдельном файле данных с расширением *.myd* и одном или нескольких файлах индекса *.myi*. Эти файлы хранятся вместе с файлом. *frm* в папке с именем базы данных, расположенной в каталоге данных, указанном в параметре загрузки --datadir. Таким образом, оптимизация дискового пространства для MyISAM выполняется так же, как оптимизация дискового пространства для сервера. Т.е. можно повысить производительность, переместив каталог данных на отдельный диск, производительность которого можно повышать, используя массивы RAID и другие возможности по обеспечению высокой доступности.

Настройка таблиц

Существует несколько команд SQL, позволяющих поддерживать таблицы в оптимальном состоянии. Это команды ANALYZE TABLE, OPTIMIZE TABLE и REPAIR TABLE.

Команда ANALYZE TABLE проверяет и реорганизует распределение ключей для таблицы. Сервер MySQL использует распределение ключей для определения порядка объединения при выполнении объединения по полю, не являющемуся константой. Распределение ключей также определяет, какие индексы будут применяться для запроса. Подробнее эта команда обсуждается в разделе «Использование команды ANALYZE TABLE».

Команда REPAIR TABLE, в общем-то, не является инструментом для повышения производительности, она применяется для исправления поврежденных таблиц, используемых механизмами БД MyISAM, Archive и CSV. Используйте эту команду для восстановления таблиц, в которых возникли повреждения или которые стали очень плохо функционировать (что обычно говорит о том, что таблица «испортилась» и требует реорганизации или восстановления).

Команда REPAIR TABLE аналогична команде myisamchk --recover <имя таблицы> (см. следующий раздел).

Команда OPTIMIZE TABLE используется для восстановления удаленных блоков и реорганизации таблицы для повышения производительности. Эту команду можно использовать для таблиц MyISAM, BDB и InnoDB.

Эти команды весьма полезны, но есть инструменты, предоставляющие больше возможностей для управления таблицами MyISAM.

Использование утилит MyISAM

В комплектацию MySQL входит несколько специальных утилит, предназначенных для работы с таблицами MyISAM:

- *myisam_ftdump* позволяет просматривать информацию о полнотекстовых индексах;
- *myisamchk* позволяет выполнять анализ таблиц MyISAM;
- *myisamlog* позволяет просматривать изменения журналов таблиц MyISAM;
- *myisampack* позволяет сжимать таблицу для уменьшения занимаемого дискового пространства.

Чаще всего используют утилиту myisamchk. С ее помощью можно просматривать информацию о таблицах MyISAM, анализировать, восстанавливать и оптимизировать их. Можно проверять одну или несколько таблиц сразу, но делать это можно только в автономном режиме (закрыть таблицы и завершить работу сервера).

Прежде чем запускать эту утилиту для восстановления или оптимизации, убедитесь, что имеете резервную копию таблиц. В редких случаях эта утилита повреждает таблицы, и их не удается восстановить.

Ниже описаны параметры, относящиеся к повышению производительности. Полный список доступных параметров можно найти в онлайновой документации MySQL Reference Manual.

- *analyze* Анализирует распределение ключей индексов для повышения производительности запросов.
- *backup* Создает копию таблиц (файлов .myd) перед их изменением.
- *check* Выполняет проверку таблиц на ошибки и только создает отчет.
- *extended-check* Выполняет полную проверку таблиц на ошибки, включая все индексы, и только создает отчет.
- *force* Выполняет восстановление при обнаружении любых ошибок.
- *information* Показывает статистическую информацию о таблице. Используется перед выполнением восстановления (recover) для проверки состояния таблицы.
- *medium-check* Выполняет более полную проверку таблицы, чем check, но менее полную, чем extended-check, и только создает отчет.
- *recover* Выполняет восстановление таблицы, восстанавливая структуры данных. Восстанавливает все, кроме дублированных уникальных ключей.
- *safe-recover* Выполняет более старую версию восстановления, при которой считываются все строки по порядку и обновляются все индексы.
- *sort index* Сортирует дерево индексов в порядке от верхних к нижним. Это может уменьшить время поиска в структурах индекса и ускорить доступ к индексам.
- *sort records* Сортирует записи в порядке, указанном индексом. Это может повысить производительность некоторых запросов на основе индекса.

В листинге 9-1 показаны результаты выполнения команды myisamchk для отображения информации о таблице MyISAM.

Лист. 9-1. Использование утилиты myisamchk

```
MyISAM file:       /usr/local/mysql/data/employees/employees
Record format:     Packed
Character set:     latin1_swedish_ci (8)
File-version:      1
Creation time:     2009-12-03 20:52:12
Status:            changed
Data records:           297024 Deleted blocks:           3000
Datafile parts:         300024 Deleted data:            95712
Datafile pointer (bytes):    6 Keyfile pointer (bytes):     6
Datafile length:   9561268 Keyfile length:            3086336
Max datafile length: 281474976710654
Max keyfile length: 288230376151710719
Recordlength:              44

table description:
Key Start Len Index  Type       Rec/key      Root Blocksize
1   1     4  unique  long             1   2931712      1024
```

Хранение таблицы в порядке индексации

Эффективность выборки крупных объемов данных при помощи запросов с условиями типа WHERE a > 5 AND a < 15 можно повысить, храня данные из таблиц в том же порядке, в каком они хранятся в индексе. Такой способ хранения позволяет запросам извлекать данные по порядку, не ища данные на разных дисковых страницах. Чтобы отсортировать таблицу в порядке индексации, используйте утилиту myisamchk с параметром –R и укажите номер индекса, который хотите использовать (первый индекс имеет номер 1). Следующая команда сортирует таблицу table1 из базы данных test по порядку второго индекса:

```
myisamchk -R 2 /usr/local/mysql/data/test/table1
```

То же самое можно сделать при помощи команд ALTER TABLE и ORDER BY.

Такая сортировка таблицы не гарантирует, что порядок индекса сохранится после добавления новых строк. Удаления не изменяют порядок, но при добавлении новых строк таблица может стать менее упорядоченной, что приведет к снижению производительности. Если вы используете этот прием для часто изменяющихся таблиц, вам следует периодически выполнять указанную команду, чтобы обеспечить оптимальный порядок.

Сжатие таблиц

Сжатие данных позволяет сэкономить место. В дополнение к методам сжатия данных средствами MySQL, механизм БД MyISAM позволяет сжимать (паковать) таблицы, доступные только для чтения, чтобы сэкономить дисковое пространство. Таблицы должны быть доступны только для чтения, так как MyISAM не имеет средств для распаковки, упорядочивания и сжатия дополнений (или удалений). Чтобы сжать таблицу, используйте утилиту myisampack:

```
myisampack -b /usr/local/mysql/data/test/table1
```

Всегда добавляйте параметр –b, чтобы создать резервную копию таблицы перед сжатием. Это позволит разрешить запись в таблицу, не выполняя повторно команду myisampack.

Есть две причины для сжатия таблиц, доступных только для чтения. Первая и главная — это может сэкономить много дискового пространства, занимаемого таблицами, содержащими легко сжимаемые данные (например, текст). Вторая причина — когда происходит обращение к сжатой таблице, а запрос использует первичный ключ или уникальный индекс для поиска строки, распаковывается только одна строка данных, прежде чем выполняются дополнительные сравнения.

Команда myisampack имеет множество параметров, о которых можно узнать из онлайновой документации MySQL Reference Manual.

Дефрагментация таблиц

Если в таблицах MyISAM происходит много изменений в виде удалений и вставок, физическое хранилище может фрагментироваться. Часто в физическом хранилище возникают небольшие промежутки, представляющие удаленные данные, а записи хранятся не в исходном порядке. Чтобы оптимизировать таблицу и реорганизовать ее, вернув желаемый порядок и форму, используйте команду OPTIMIZE TABLE или утилиту myisamchk.

Эти команды следует периодически выполнять для тех таблиц, для которых указан порядок сортировки, чтобы они хранились в оптимальном виде. Также следует выполнить одну из этих команд, если данные претерпели много изменений за некоторый период времени.

Мониторинг кэша ключей

Кэш ключей MySQL — очень эффективная структура, предназначенная для хранения часто используемых данных индекса. Он используется только механизмом MyISAM и хранит ключи с использованием быстрого механизма поиска (обычно применяется B-дерево). Индексы хранятся в памяти как связанные списки, и поиск в них можно выполнять очень быстро. Кэш ключей создается автоматически при первом открытии таблицы MyISAM для чтения. Когда запрос обращается к таблице MyISAM, сначала проверяется кэш ключей. Если индекс обнаружен в кэше, выполняется поиск в памяти, что гораздо быстрее поиска на диске. Кэш ключей — секретное оружие, благодаря которому MyISAM настолько быстрее обрабатывает некоторые запросы, по сравнению с другими механизмами БД.

Есть несколько переменных, управляющих кэшем ключей, мониторинг каждой из которых можно выполнять при помощи команд SHOW VARIABLES и SHOW STATUS или средствами MySQL Administrator. О мониторинге кэша ключей при помощи MySQL Administrator мы говорили в гл. 8 (см. рис. 8-5). Переменные, мониторинг которых можно выполнять командами SHOW, показаны в листинге 9-2.

Лист. 9-2. Переменные системы и состояния кэша ключей

```
mysql> SHOW STATUS LIKE 'Key%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Key_blocks_not_flushed | 0     |
| Key_blocks_unused      | 6694  |
| Key_blocks_used        | 0     |
| Key_read_requests      | 0     |
| Key_reads              | 0     |
| Key_write_requests     | 0     |
| Key_writes             | 0     |
+------------------------+-------+
```

```
7 rows in set (0.00 sec)
mysql> SHOW VARIABLES LIKE 'key%';
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| key_buffer_size         | 8384512 |
| key_cache_age_threshold | 300     |
| key_cache_block_size    | 1024    |
| key_cache_division_limit | 100    |
+-------------------------+---------+
4 rows in set (0.01 sec)
```

Как можно представить, кэш ключей может быть очень сложным механизмом. А значит, настройка кэша ключей может быть сложной задачей. Мы рекомендуем выполнять мониторинг использования и изменять только размер кэша ключей, не вмешиваясь в его работу, так как он работает очень хорошо при настройках по умолчанию.

Если хотите повысить производительность кэша, используйте один из двух описанных ниже приемов: предварительная загрузка кэша и использование нескольких кэшей с добавлением памяти кэшу, используемому по умолчанию.

Предварительная загрузка в кэш

Можно заранее загружать индексы в кэш ключей. Это обеспечивает более быстрое выполнение запросов, так как индекс уже загружен в кэш ключей и загружен последовательно (а не произвольно, что происходит, когда кэш ключей загружается во время текущих операций). Однако необходимо убедиться, что в кэше достаточно места для хранения индекса. Предварительная загрузка может быть очень эффективной для ускорения обработки запросов для определенных приложений и режимов использования. Например, если вы знаете, что во время работы приложения будет много запросов к определенной таблице, можете загрузить связанные с ней индексы в кэш ключей, тем самым повысив производительность обработки этих запросов. Чтобы выполнить предварительную загрузку, используйте команду LOAD INDEX, как показано в листинге 9-3.

Лист. 9-3. Предварительная загрузка индексов в кэш ключей

```
mysql> LOAD INDEX INTO CACHE salaries IGNORE LEAVES;
+--------------------+--------------+----------+----------+
| Table              | Op           | Msg_type | Msg_text |
+--------------------+--------------+----------+----------+
| employees.salaries | preload_keys | status   | OK       |
+--------------------+--------------+----------+----------+
1 row in set (1.49 sec)
```

В этом листинге в кэш ключей загружается индекс для таблицы *salary*. Выражение IGNORE LEAVES выполняет предварительную загрузку только блоков только узлов, не являющихся листами дерева индекса. Хотя нет специальной команды для очистки кэша ключей, можно принудительно удалить из него индекс, изменив таблицу; например, реорганизовав индекс или просто сбросив и воссоздав его.

Использование нескольких кэшей ключей

Одним из малоизвестных преимуществ MySQL является создание нескольких кэшей ключей и пользовательских кэшей ключей для снижения нагрузки на кэш, используемый по умолчанию. Эта возможность позволяет загружать индекс одной или нескольких таблиц в специальный настраиваемый кэш. Как можно догадаться, это означает выделение памяти для задачи, а значит, требует тщательного планирования. Однако повышение производительности может быть существенным, когда выполняется много запросов к набору таблиц с частым обращениям к индексам.

Чтобы создать вторичный кэш ключей, сначала определите его командой SET, выделив память, а затем выполните одну или несколько команд CACHE INDEX, чтобы загрузить индексы для одной или нескольких таблиц. В отличие от стандартного кэша ключей, вторичный кэш можно очистить или удалить, задав ему нулевой размер. В листинге 9-4 показано, как создать вторичный кэш ключей и добавить в него индекс.

Лист. 9-4. Использование вторичных кэшей ключей

```
mysql> SET GLOBAL emp_cache.key_buffer_size=128*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CACHE INDEX salaries IN emp_cache;
+--------------------+--------------------+----------+----------+
| Table              | Op                 | Msg_type | Msg_text |
+--------------------+--------------------+----------+----------+
| employees.salaries | assign_to_keycache | status   | OK       |
+--------------------+--------------------+----------+----------+
1 row in set (0.00 sec)

mysql> SET GLOBAL emp_cache.key_buffer_size=0;
Query OK, 0 rows affected (0.00 sec)
```

Заметьте, что вторичный кэш требует определения новой переменной emp_cache и установки ее размера в 128 Кб. Это специальный синтаксис команды SET, создающий новую глобальную пользовательскую переменную (а не системную переменную, как может показаться). Определить, существует ли вторичный кэш, и узнать его размер можно следующим образом:

```
mysql> select @@global.emp_cache.key_buffer_size;
+-----------------------------------+
| @@global.emp_cache.key_buffer_size |
+-----------------------------------+
|                            131072 |
+-----------------------------------+
1 row in set (0.00 sec)
```

Вторичные кэши ключей являются глобальными и поэтому существуют до тех пор, пока их не очистить, задав нулевой размер, или до перезагрузки сервера.

Можно сохранить конфигурацию нескольких кэшей ключей, записав выражения в файл и используя команду init-file=<путь_к_файлу> в секции [mysql] файла параметров MySQL для выполнения этих выражений при загрузке.

Другие параметры

Существуют также некоторые другие параметры, влияющие на производительность. Помните: изменяйте только один параметр за раз и только в том случае, если имеете для того веские основания. Никогда не следует изменять конфигурацию такой сложной системы, как механизм БД, не имея на то серьезных причин, и без понимания последствий.

- *myisam_data_pointer_size* Стандартный размер указателя в байтах (2-7) используется командой CREATE TABLE, если не указано значение MAX_ROWS (максимальное число строк, хранящихся в таблице). По умолчанию установлено значение 6.
- *myisam_max_sort_file_size* Максимальный размер временного файла, используемого при сортировке данных. Повышение этого значения может ускорить операции восстановления и реорганизации индекса.
- *myisam_recover_options* Режим восстановления MyISAM. Этот параметр можно также использовать с командой OPTIMIZE TABLE. Существуют следующие режимы: стандартный (default), резервного копирования (backup), принудительный (force) и быстрый (quick). В стандартном режиме восстановление выполняется без резервного копирования, принудительного продолжения или быстрой проверки. В режиме резервного копирования перед восстановлением создается резервная копия. В принудительном режиме восстановление продолжается даже в том случае, если данные потеряны (более одной строки). В быстром режиме строки таблицы не проверяются на наличие блоков, помеченных как удаленные. Выбирая режим, учитывайте серьезность восстановления.
- *myisam_repair_threads* Если задать значение больше 1, операции восстановления и сортировки будут выполняться параллельно, что позволяет завершить их немного быстрее. В противном случае они будут выполняться последовательно.

- *myisam_sort_buffer_size* Размер буфера для операций сортировки. Повышение этого значения может помочь в сортировке индексов. Однако значения более 4 Гб будут работать только в 64-разрядных системах.
- *myisam_stats_method* Определяет, как сервер считает значения NULL в распределении значений индекса в статистических операциях. Этот параметр может повлиять на оптимизатор, так что его следует изменять с осторожностью.
- *myisam_use_map* Включает использование карты памяти для чтения и записи таблиц MyISAM. Это может быть полезным, когда выполняется много небольших операций записи, конкурирующих с запросами на чтение, извлекающих большие наборы данных.

Мы обсудили некоторые стратегии мониторинга и повышения производительности MyISAM. Хотя обсуждение получилось весьма кратким, оно охватывает наиболее важные аспекты эффективного использования MyISAM. Подробнее о кэшах ключей и механизме БД MyISAM можно узнать из онлайновой документации MySQL Reference Manual.

MySQL, репликация и высокая доступность

Вероятность повреждения данных MyISAM выше, чем данных InnoDB, в результате чего при использовании MyISAM выше суммарное время на восстановление. Кроме того, так как MyISAM не поддерживает транзакции, события выполняются по одному за раз, что может привести к частичному выполнению выражений и незавершенным транзакциям. Добавьте к этому тот факт, что подчиненные серверы — однопоточные, что может привести к их длительному отставанию во время обработки долгих запросов. Таким образом, использование MyISAM на подчиненных серверах, когда требуется обеспечить высокую доступность и использовать транзакции, может приводить к возникновению проблем.

InnoDB

Существует много параметров, которые можно настраивать для механизма БД InnoDB, и для их полного описания может потребоваться отдельная книга. Например, есть 50 переменных, управляющих поведением InnoDB и более 40 переменных состояния, взаимодействующих с метаданными, относящимися к производительности и состояниям. В этом разделе мы расскажем о мониторинге механизма БД InnoDB и обсудим некоторые ключевые области, связанные с повышением производительности.

Мы не будем обсуждать общие аспекты, а опишем стратегию повышения производительности, разбив ее на следующие группы:

- использование команды SHOW ENGINE;
- использование мониторов InnoDB;
- мониторинг файлов журнала;

- мониторинг пула буферов;
- мониторинг табличного пространства;
- использование таблиц INFORMATION_SCHEMA;
- другие параметры.

В следующих разделах мы вкратце обсудим каждую из этих тем. Однако сначала сделаем обзор особенностей архитектуры InnoDB.

Механизм БД InnoDB использует очень сложную архитектуру, спроектированную для поддержки транзакций и большой параллельной нагрузки. Этот механизм имеет некоторые дополнительные функции, которые следует изучить, прежде чем пытаться повышать производительность. Здесь мы рассмотрим те функции, которые можно настраивать. Сюда относятся индексы, пул буферов, файлы журналов и табличные пространства.

Индексы таблиц InnoDB используют кластеризованные индексы. Кластеризованный индекс — структура данных, хранящая не только индекс, но и сами данные. Это значит, что как только значение найдено в индексе, можно извлекать данные, не выполняя дополнительный поиск по диску. Естественно, индекс первичного ключа и первый уникальный индекс таблицы создаются как кластеризованные индексы.

При создании вторичного индекса ключ из кластеризованного индекса (первичный ключ, уникальный ключ или ID строки) сохраняется вместе со значением для вторичного индекса. Это позволяет очень быстро переадресовывать запросы и обращаться к исходным данным из кластеризованного индекса. Это также означает, что можно использовать столбцы первичных ключей при сканировании вторичного индекса, чтобы позволить запросу использовать только вторичный индекс для получения данных.

Пул буферов — механизм кэширования для управления транзакциями и чтения/записи данных с дисков и на диски, который при правильной настройке может ускорить доступ к дискам. Кроме того, пул буферов жизненно важен для восстановления после сбоев, так как периодически записывается на диск (например, во время завершения работы). Так как это компонент, находящийся в памяти, необходимо выполнять мониторинг его эффективности, чтобы следить за его правильной настройкой.

Кроме этого, InnoDB использует пул буферов для хранения изменений данных и транзакций. Механизм БД кэширует изменения, сохраняя их в странице (блоке) данных в пуле буферов. При каждом обращении к странице она помещается в пул буферов, а при изменении помечается как модифицированная. Затем изменения записываются на диск, обновляя данные и копию, записанную в журнал повтора. Файлы журналов хранятся в виде файлов с именами *ib_logfile0* и *ib_logfile1*. Эти файлы можно увидеть в каталоге данных сервера MySQL.

Для хранения данных механизм БД InnoDB использует два механизма, основанных на дисковой подсистеме: файлы журналов и табличные пространства. Также используются журналы для воссоздания (повтора) из-

менений данных, внесенных до завершения работы или сбоя. Во время загрузки InnoDB считывает журналы и автоматически записывает модифицированные страницы на диск, тем самым восстанавливая буферизованные изменения, внесенные до сбоя.

Табличные пространства — организационный инструмент, используемый InnoDB в качестве файла, независимого от оборудования, содержащего и данные, и индексы, а также в качестве механизма отката (для отката транзакций). По умолчанию все таблицы совместно используют одно табличное пространство. Можно сохранять таблицы в собственных табличных пространствах. Табличные пространства содержат и данные, и индексы таблиц. Они автоматически расширяются, занимая несколько файлов, позволяя хранить в таблицах больше данных, чем допускает операционная система. Табличное пространство можно разделить на несколько файлов и разместить эти файлы на разных дисках.

Для создания отдельного табличного пространства для каждой таблицы используйте параметр innoDB_file_per_table. Все таблицы, созданные до применения этого параметра, останутся в общем табличном пространстве. Новые настройки будут действовать только для новых таблиц.

Использование команды SHOW ENGINE

Команда SHOW ENGINE INNODB STATUS отображает статистические данные и сведения о конфигурации, относящиеся к состоянию механизма БД InnoDB. Это стандартный способ просмотра информации о InnoDB. Листинг отображаемых статистических данных длинный и очень сложный. В листинге 9-5 показан фрагмент выходных данных этой команды, выполненной в стандартной установке MySQL.

Лист. 9-5. Команда SHOW ENGINE INNODB STATUS

```
mysql> SHOW ENGINE INNODB STATUS \G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
091205 18:31:10 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 1 seconds


----------
BACKGROUND THREAD
----------
srv_master_thread loops: 233 1_second, 34 sleeps, 23 10_second,
                         3 background, 3 flush
srv_master_thread log flush and writes: 44 log writes only: 448
```

```
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 882, signal count 901
Mutex spin waits 2501, rounds 21869, OS waits 388
RW-shared spins 165, OS waits 144; RW-excl spins 0, OS waits 335
Spin rounds per wait: 8.74 mutex, 26.70 RW-shared, 10301.00 RW-excl


------------
TRANSACTIONS
------------
Trx id counter 2969
Purge done for trx's n:o < 2519 undo n:o < 0
History list length 3
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0, not started, OS thread id 4491317248
MySQL thread id 4, query id 152 localhost root
SHOW ENGINE INNODB STATUS
---TRANSACTION 2968, ACTIVE 0 sec, OS thread id 4548612096 inserting
mysql tables in use 1, locked 1
7 lock struct(s), heap size 1216, 1171 row lock(s), undo log entries 11375
MySQL thread id 3, query id 151 localhost root update
INSERT INTO `salaries` VALUES (204383,71223,'1998-11-14','1999-09-07'),
...


--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (read thread)
I/O thread 4 state: waiting for i/o request (read thread)
I/O thread 5 state: waiting for i/o request (read thread)
I/O thread 6 state: waiting for i/o request (write thread)
I/O thread 7 state: waiting for i/o request (write thread)
I/O thread 8 state: waiting for i/o request (write thread)
I/O thread 9 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 1
31 OS file reads, 3979 OS file writes, 1593 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 146.85 writes/s, 78.92 fsyncs/s
```

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2,
0 inserts, 0 merged recs, 0 merges
Hash table size 276707, node heap has 444 buffer(s)
92903.10 hash searches/s, 459.54 non-hash searches/s
---
LOG
---
Log sequence number 219912928
Log flushed up to  218951284
Last checkpoint at 217528539
0 pending log writes, 0 pending chkp writes
1074 log i/o's done, 46.95 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 138805248; in additional pool allocated 0
Dictionary memory allocated 70560
Buffer pool size  8192
Free buffers      760
Database pages    6988
Modified db pages 113
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 21, created 6968, written 10043
0.00 reads/s, 89.91 creates/s, 125.87 writes/s
Buffer pool hit rate 1000 / 1000
LRU len: 6988, unzip_LRU len: 0
I/O sum[9786]:cur[259], unzip sum[0]:cur[0]
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread id 4528414720, state: sleeping
Number of rows inserted 2078594, updated 0, deleted 0, read 0
31059.94 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

Команда SHOW ENGINE INNODB MUTEX отображает информацию о мьютексах InnoDB и может быть очень полезной для настройки потоков механизма БД. В листинге 9-6 показан фрагмент выходных данных этой команды, выполненной в стандартной установке MySQL.

Лист. 9-6. Команда SHOW ENGINE INNODB MUTEX

```
mysql> SHOW ENGINE INNODB MUTEX;
+--------+--------------------+--------------+
| Type   | Name               | Status       |
+--------+--------------------+--------------+
| InnoDB | trx/trx0rseg.c:167 | os_waits=1   |
| InnoDB | trx/trx0sys.c:181  | os_waits=7   |
| InnoDB | log/log0log.c:777  | os_waits=1003 |
| InnoDB | buf/buf0buf.c:936  | os_waits=8   |
| InnoDB | fil/fil0fil.c:1487 | os_waits=2   |
| InnoDB | srv/srv0srv.c:953  | os_waits=101 |
| InnoDB | log/log0log.c:833  | os_waits=323 |
+--------+--------------------+--------------+
7 rows in set (0.00 sec)
```

Столбец Name содержит имя исходного файла и номер строки, где создан мьютекс. В столбце Status показано, сколько раз мьютекс ожидал операционную систему (например, os_waits=5). Если исходный код был скомпилирован с директивой UNIV_DEBUG, этот столбец может содержать одно из следующих значений:

- *count* — число запросов мьютекса;
- *spin_waits* — число выполнений операции spinlock;
- *os_waits* — число ожиданий мьютекса операционной системой;
- *os_yields* — показывает, сколько раз поток оставил свой временной интервал и вернулся к операционной системе;
- *os_wait_times* — промежуток времени, в течение которого мьютекс ожидал операционную систему мьютексом.

Команда SHOW ENGINE INNODB STATUS отображает много информации, полученной от механизма БД InnoDB. Эта информация не отформатирована (не разделена на строки и столбцы), но существуют утилиты, использующие ее и представляющие в удобном для чтения виде. Например, так работает команда InnoTop.

Использование мониторов InnoDB

InnoDB — единственный механизм БД, напрямую поддерживающий мониторинг. InnoDB имеет специальный механизм, называемый *монитором*, который собирает статистическую информацию и отправляет ее серверу и клиентским утилитам. Все перечисленные ниже элементы (и большинство сторонних утилит) взаимодействуют со средствами мониторинга InnoDB, а значит, InnoDB выполняет мониторинг этих элементов посредством сервера MySQL:

- блокировки таблиц и записей;
- ожидания блокировок;

- ожидания семафоров;
- запросы файлового ввода-вывода;
- пул буферов;
- операции слияния буфера с очисткой и вставкой.

Мониторы InnoDB автоматически запускаются командой SHOW ENGINE INNODB STATUS и генерируют отображаемую информацию. Эту информацию можно получить и напрямую от мониторов InnoDB, создав в MySQL специальный набор таблиц. Схема и местоположение таблиц не важно (если используется выражение ENGINE = INNODB). После создания каждая из этих таблиц указывает InnoDB, что данные следует выгружать в stderr. Эту информацию можно просматривать в журнале ошибок или в консоли, запустив MySQL с параметром --console. Чтобы включить мониторы InnoDB создайте в любой базе данных следующие таблицы:

```
mysql> SHOW TABLES LIKE 'innodb%';
+---------------------------+
| Tables_in_test (innodb%)  |
+---------------------------+
| innodb_lock_monitor       |
| innodb_monitor            |
| innodb_table_monitor      |
| innodb_tablespace_monitor |
+---------------------------+
4 rows in set (0.00 sec)
```

Чтобы отключить мониторы, просто удалите таблицу. Мониторы автоматически генерируют данные каждые 15 секунд.

Каждый монитор представляет следующие данные:

- *innodb_monitor* Стандартный монитор, выводящий ту же информацию, что и SQL-команда status. Выходные данные этого монитора показаны в листинге 9-5. Единственное различие в выходных данных команды SQL и монитора innodb_monitor состоит в том, что выходные данные в stderr отформатированы так же, как в клиенте MySQL при использовании вертикального отображения.
- *innodb_lock_monitor* Монитор блокировки отображает ту же информацию, что и команда SQL, но включает дополнительные сведения о блокировках.
- *innodb_table_monitor* Монитор таблиц создает подробный отчет о внутреннем словаре данных. Фрагмент сгенерированного отчета показан в листинге 9-7 (отформатирован для читаемости). Обратите внимание, насколько подробна информация о каждой таблице: указаны определения столбцов, индексы, Примерное число строк, внешние ключи и другое. Этот отчет можно использовать для диагностики проблем с таблицами или для получения подробных сведений об индексах.

Лист. 9-7. Отчет монитора таблиц InnoDB

```
=========================================
091208 21:10:06 INNODB TABLE MONITOR OUTPUT
=========================================
-------------------------------------
TABLE: name sakila/address, id 0 14, flags 1,
       columns 11, indexes 2, appr.rows 628
COLUMNS: address_id: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE
                     DATA_NOT_NULL len 2;
         address: type 12 DATA_NOT_NULL len 150;
         address2: type 12 len 150;
         district: type 12 DATA_NOT_NULL len 60;
         city_id: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE
                  DATA_NOT_NULL len 2;
...
   INDEX: name PRIMARY, id 0 17, fields 1/10, uniq 1, type 3
    root page 52, appr.key vals 628, leaf pages 4, size pages 5
    FIELDS: address_id DB_TRX_ID DB_ROLL_PTR address address2
            district city_id postal_code phone last_update
   INDEX: name idx_fk_city_id, id 0 18, fields 1/2, uniq 2,
          type 0 root page 53, appr.key vals 599, leaf pages 1,
          size pages 1
    FIELDS: city_id address_id
   FOREIGN KEY CONSTRAINT sakila/fk_address_city:
                          sakila/address ( city_id )
            REFERENCES sakila/city ( city_id )
...
---------------------------------
END OF INNODB TABLE MONITOR OUTPUT
=================================
```

- *innodb_tablespace_monitor* Отображает расширенную информацию об общем табличном пространстве, включая список сегментов файлов, и подтверждает распределение структур данных табличного пространства. Отчет может быть весьма подробным и очень длинным, так как содержит все данные о табличном пространстве. В листинге 9-8 показан фрагмент такого отчета.

Лист. 9-8. Отчет монитора табличного пространства InnoDB

```
============================================
091208 21:14:19 INNODB TABLESPACE MONITOR OUTPUT
============================================
FILE SPACE INFO: id 0
size 16000, free limit 15424, free extents 2
not full frag extents 3: used pages 144, full frag extents 41
first seg id not used 0 714
SEGMENT id 0 1 space 0; page 2; res 2 used 2; full ext 0
```

```
fragm pages 2; free extents 0; not full extents 0: pages 0

...

SEGMENT id 0 411 space 0; page 209; res 29 used 29; full ext 0
fragm pages 29; free extents 0; not full extents 0: pages 0
SEGMENT id 0 412 space 0; page 209; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 413 space 0; page 209; res 96 used 60; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 28
SEGMENT id 0 414 space 0; page 209; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 415 space 0; page 209; res 96 used 33; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 1
NUMBER of file segments: 275
Validating tablespace
Validation ok
---------------------------------------
END OF INNODB TABLESPACE MONITOR OUTPUT
=======================================
```

Как видите, мониторы InnoDB предоставляют весьма подробную информацию. Если долго держать их включенными, в файлы журналов может быть добавлен значительный объем данных.

Мониторинг файлов журналов

Так как файлы журналов InnoDB играют роль буфера между данными и операционной системой, их хорошая работа обеспечивает хорошую производительность. Выполнять мониторинг файлов журнала можно напрямую, наблюдая за следующими системными переменными:

```
mysql> SHOW STATUS LIKE 'Innodb%log%';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Innodb_log_waits              | 0     |
| Innodb_log_write_requests     | 0     |
| Innodb_log_writes             | 2     |
| Innodb_os_log_fsyncs          | 5     |
| Innodb_os_log_pending_fsyncs  | 0     |
| Innodb_os_log_pending_writes  | 0     |
| Innodb_os_log_written         | 1024  |
+-------------------------------+-------+
```

Мы уже видели некоторые из этих данных в представлении мониторов InnoDB, но можно также получить подробную информацию о файлах журналов, используя следующие переменные состояния:

- *Innodb_log_waits* Количество ситуаций, когда журнал был слишком мал (т. е. не мог вместить все данные) и операции приходилось ожидать его очистки. Если это значение начинает увеличиваться и долго остается больше нуля (исключением могут быть массовые операции), следует увеличить размер файлов журнала.
- *Innodb_log_write_requessts* Число запросов на запись в журнал.
- *Innodb_log_writes* Число операций записи данных в журнал.
- *Innodb_os_log_fsyncs* Число синхронизаций файлов операционной системы, т.е. число вызовов метода fsync().
- *Innodb_os_log_pending_fsyncs* Число ожидающих запросов на синхронизацию файлов. Если это значение начинает увеличиваться и долго остается больше нуля, возможно, возникли проблемы с дисками.
- *Innodb_os_log_pending_writes* Число ожидающих запросов на запись в журнал. Если это значение начинает увеличиваться и долго остается больше нуля, возможно, возникли проблемы с дисками.
- *Innodb_os_log_written* Общее число байтов, записанных в журнал.

Так как все эти переменные содержат числовую информацию, можно строить собственные графики при помощи MySQL Administrator, представляя данные в графическом виде.

Мониторинг пула буферов

В пуле буферов InnoDB кэширует часто запрашиваемые данные. Любые изменения, внесенные в данные из пула буферов, также кэшируются. Кроме того, пул буферов хранит информацию о текущих транзакциях. Таким образом, пул буферов — критически важный механизм, от которого зависит производительность.

Для просмотра информации о состоянии пула буферов используется команда SHOW ENGINE INNODB STATUS (см. листинг 9-5). Для удобства повторим секцию с данными о пуле буферов и памяти:

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 138805248; in additional pool allocated 0
Dictionary memory allocated 70560
Buffer pool size  8192
Free buffers      760
Database pages    6988
Modified db pages 113
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 21, created 6968, written 10043
0.00 reads/s, 89.91 creates/s, 125.87 writes/s
Buffer pool hit rate 1000 / 1000
```

```
LRU len: 6988, unzip_LRU len: 0
I/O sum[9786]:cur[259], unzip sum[0]:cur[0]
```

Перечислим самые важные элементы из этого списка (более специфичные переменные состояния обсудим чуть позже):

- *Free buffers* — число пустых и свободных для записи данных сегментов буферов;
- *Modified db pages* — число изменившихся (грязных) страниц;
- *Pending reads* — число ожидающих операций чтения;
- *Pending writes* — число ожидающих операций записи;
- *Buffer pool hit rate* — отношение числа удачных операций чтений из буфера к общему числу запросов. Этот коэффициент должен быть как можно ближе к 1:1.

Есть несколько переменных состояния, с помощью которых можно просматривать более подробную информацию. Ниже перечислены переменные состояния пула буферов InnoDB:

```
mysql> SHOW STATUS LIKE 'Innodb%buf%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Innodb_buffer_pool_pages_data     | 21    |
| Innodb_buffer_pool_pages_dirty    | 0     |
| Innodb_buffer_pool_pages_flushed  | 1     |
| Innodb_buffer_pool_pages_free     | 8171  |
| Innodb_buffer_pool_pages_misc     | 0     |
| Innodb_buffer_pool_pages_total    | 8192  |
| Innodb_buffer_pool_read_ahead_rnd | 0     |
| Innodb_buffer_pool_read_ahead_seq | 0     |
| Innodb_buffer_pool_read_requests  | 558   |
| Innodb_buffer_pool_reads          | 22    |
| Innodb_buffer_pool_wait_free      | 0     |
| Innodb_buffer_pool_write_requests | 1     |
+-----------------------------------+-------+
```

Существует несколько переменных состояния, отображающих ключевую статистическую информацию о производительности пула буферов. Можно просмотреть подробные сведения о состоянии страниц пула буферов, чтении и записи из пула и в пул, количестве ожидающих операций записи и чтения. Эти переменные состояния перечислены ниже:

- *Innodb_buffer_pool_pages_data* Число страниц, содержащих данные, включая неизмененные и измененные («грязные») страницы.
- *Innodb_buffer_pool_pages_dirty* Число измененных страниц.
- *Innodb_buffer_pool_pages_flushed* Число очисток страниц пула буферов.

- *Innodb_buffer_pool_pages_free* Число пустых (свободных) страниц.
- Innodb_buffer_pool_pages_misc Число страниц, используемых для административных нужд самим механизмом InnoDB. Это значение вычисляется следующим образом:

```
X = Innodb_buffer_pool_pages_total - Innodb_buffer_pool_pages_free -
Innodb_buffer_pool_pages_data
```

- *Innodb_buffer_pool_pages_total* Общее число страниц в пуле буферов.
- *Innodb_buffer_pool_read_ahead_rnd* Число произвольных опережающих операций чтения, выполненных InnoDB во время сканирования крупных блоков данных.
- *Innodb_buffer_pool_read_ahead_seq* Число последовательных опережающих операций чтения, выполненных в результате полного последовательного сканирования таблицы.
- *Innodb_buffer_pool_read_requests* Число логических запросов на чтение.
- *Innodb_buffer_pool_reads* Число логических операций чтения, для которых в пуле буферов не обнаружены данные, и которые выполнены напрямую с диска.
- *Innodb_buffer_pool_wait_free* Если пул буферов занят или нет свободных страниц, может потребоваться ожидание очистки страниц. Это значение показывает число таких ожиданий. Если это значение растет и остается больше нуля, это может указывать на проблемы с размером пула буферов или с дисками.
- *Innodb_buffer_pool_write_requests* Число операций записи в пул буферов InnoDB.

Так как все эти переменные содержат числовую информацию, можно строить собственные графики при помощи MySQL Administrator, представляя данные в графическом виде.

Мониторинг табличных пространств

В принципе, табличные пространства InnoDB не требуют внимания администратора, если разрешить InnoDB расширять их, когда кончается место. Можно настроить табличные пространства, чтобы они автоматически увеличивались, используя параметр autoextend переменной innodb_data_file_path. Например, в стандартной конфигурации MySQL общее табличное пространство изначально имеет размер 10 Мб и может автоматически расширяться.

```
--innodb_data_file_path=ibdata1:10M:autoextend
```

Подробнее об этом см. в разделе «InnoDB Configuration» онлайновой документации MySQL Reference Manual.

Текущую конфигурацию табличных пространств можно просмотреть при помощи команды SHOW ENGINE STATUS INNODB. Подробные сведения о табличных пространствах можно получить, включив монитор табличных пространств InnoDB (см. раздел «Using Tablespace Monitors» онлайновой документации MySQL Reference Manual).

Использование таблиц INFORMATION_SCHEMA

Если в вашей версии MySQL имеется подключаемый модуль механизма БД InnoDB (доступен в MySQL 5.1 и выше), можно получить доступ к семи специальным таблицам из базы данных INFORMATION_SCHEMA.

Таблицы INFORMATION_SCHEMA необходимо установить отдельно. Подробнее см. в документации по подключаемому модулю InnoDB (*http://www.in nodb.com/products/innodb_plugin/plugin-documentation*).

Технически эти таблицы не являются таблицами, так как данные, представленные в них, не хранятся на диске, а генерируются, когда происходит обращение к этим таблицам. Эти таблицы предоставляют еще один способ мониторинга InnoDB, позволяя администраторам получать информацию о производительности. Есть отдельные таблицы для мониторинга сжатия, транзакций и блокировок. Перечислим эти таблицы:

- *INNODB_CMP* Содержит информацию о сжатых таблицах.
- *INNODB_CMP_RESET* Содержит ту же информацию, что и *INNODB_CMP*, но имеет особенность: при запросе данных из таблицы статистика сбрасывается. Это позволяет получать периодическую статистику (по часам, дням и т. д.).
- *INNODB_CMPMEM* Содержит информацию об использовании сжатия в пуле буферов.
- *INNODB_CMPMEM_RESET* Содержит ту же информацию, что и *INNODB_CMPMEM*, но имеет особенность: при запросе данных из таблицы статистика сбрасывается. Это позволяет получать периодическую статистику (по часам, дням и т. д.).
- *INNODB_TRX* Содержит информацию обо всех транзакциях, включая состояние и текущий обрабатываемый запрос.
- *INNODB_LOCKS* Содержит информацию обо всех блокировках, запрошенных транзакциями. Сюда записываются данные о каждой блокировке, включая состояние, режим, тип и т. д.
- *INNODB_LOCK_WAITS* Содержит информацию обо всех блокировках, запрошенных транзакциями, но заблокированных. Сюда записываются данные о каждой блокировке, включая состояние, режим, тип и блокирующую транзакцию.

Полное описание каждой таблицы, включая сведения о столбцах и примеры использования, см. в документации по подключаемому модулю InnoDB.

Таблицы сжатия можно использовать для мониторинга сжатия в таблицах, включая информацию о размере страниц, количестве использованных таблиц, времени на сжатие и распаковку и многом другом. Эти данные могут быть важны, когда используется сжатие и нужно убедиться, что лишняя нагрузка не влияет на производительность сервера баз данных.

Таблицы транзакций и блокировок можно использовать для наблюдения за транзакциями. Это очень полезный инструмент для слежения за надлежащей работой баз данных, использующих транзакции. Что самое важное, можно точно определить, в каком состоянии находится каждая транзакция, и какие транзакции заблокированы. Эта информация может быть очень важной для диагностирования сложных проблем с транзакциями, таких как мертвые блокировки и плохая производительность.

Другие параметры

Механизм БД InnoDB позволяет изменять и отслеживать множество параметров. Мы обсудили только часть из них, затронув, в основном, мониторинг различных подсистем и повышение производительности. Есть еще несколько элементов, которые следует отметить.

В некоторых случаях можно повысить производительность потоков, изменив параметр innodb_thread_concurrency. По умолчанию установлено значение 0, чего обычно достаточно. Но если сервер MySQL работает на многопроцессорном компьютере, имеющем много независимых дисков (и активно использующем InnoDB), можно установить значение, равное числу процессоров плюс число независимых дисков. Это позволит InnoDB использовать достаточно потоков, чтобы поддерживать максимум параллельных операций. Если установить значение большее, чем может обеспечить сервер, никакого эффекта не будет — если нет доступных потоков, предел никогда не будет достигнут.

Если сервер MySQL запущен на компьютере, который часто или периодически выключается (например, на лэптопе), можно заметить, что при использовании InnoDB выключение выполняется довольно долго. В этом случае можно настроить InnoDB на быстрое отключение, установив параметр innodb_fast_shutdown. Это не повлияет на целостность данных и не приведет к потере данных из буфера. Скорость увеличивается за счет пропуска потенциально долгих операций очистки внутренних кэшей и слияния буферов вставки. В остальном, выполняется управляемое выключение с сохранением пулов буферов на диск.

В некоторых ранних выпусках MySQL есть проблемы с управлением параллельностью и блокировками. В таких версиях можно определять, как InnoDB будет обрабатывать мертвые блокировки, используя переменную innodb_lock_wait_timeout. Эта переменная имеет как глобальную область действия, так и область действия в пределах сеанса. Она определяет, как долго InnoDB позволит транзакции ожидать блокировки строки, прежде чем

отменит операцию. Значение по умолчанию — 50 секунд. Если вы замечаете много таймаутов, связанных с ожиданием блокировок, можете увеличить это значение и решить некоторые проблемы с параллельной обработкой.

Если вы импортируете много данных, можете уменьшить время загрузки, отсортировав входящие файлы данных в порядке, соответствующем первичному ключу. Кроме того, можно отключить автоматическое подтверждение, задав переменной AUTOCOMMIT значение 0. При этом вся операция загрузки будет подтверждена только один раз. Для ускорения массовой загрузки можно также отключить внешний ключ и ограничения уникальности.

Помните, что к настройке InnoDB следует подходить с большой осторожностью. Имея возможность изменять так много параметров, можно легко что-нибудь испортить. Изменяйте не больше одной переменной за раз (и только если имеете на то причины) и наблюдайте за результатами.

Заключение

В этой главе рассмотрены способы мониторинга и повышения производительности механизмов БД на сервере MySQL, обсудив два из наиболее популярных механизмов БД. Следующая глава посвящена более сложным вопросам мониторинга и повышения производительности репликации.

Джоэл задержал указатель мыши над кнопкой отправки. Он подготовил отчет по мониторингу InnoDB и некоторые рекомендации, и собрался отправить шефу, но не был уверен, следует ли отправлять то, о чем его не просили. Пожав плечами, он решил, что хуже не будет, и отправил сообщение.

Минуты через две в ящик «упало» новое письмо и Джоэл открыл его. Письмо оказалось от Саммерсона.

«Джоэл, хорошая работа. Я хочу, чтобы ты организовал собрание разработчиков и ребят из ИТ. Ознакомь всех со своими рекомендациями и заставь выполнять их. Буду в офисе в понедельник».

— Хорошо, — сказал Джоэл, почувствовав, как потяжелело его бремя ответственности. Он осознал, что впервые будет проводить собрание с момента поступления на работу. Он немного нервничал, и потому решил пройтись, прежде чем написать повестку и разослать приглашения.

— Что ж, вряд ли это будет труднее, чем защита диссертации...

ГЛАВА 10

Мониторинг репликации

Джоэл потратил несколько секунд, чтобы войти на подчиненный сервер репликации в Сиэтле и определить, что репликация все еще работает.

Из двери раздался знакомый голос:

— Что там с Сиэтлом, Джоэл? Ты работаешь над этим?

— Да, сэр, еще работаю. Мне нужно понять, как настроена репликация и выяснить, в чем проблема, — ответил Джоэл, а про себя добавил: «...и успеть еще почитать про мониторинг репликации».

— Хорошо, работай. Я еще зайду после обеда.

Как только г-н Саммерсон ушел, Джоэл посмотрел на часы.

— Итак, у меня всего час на то, чтобы разобраться с мониторингом репликации...

Со вздохом Джоэл снова раскрыл свою любимую книгу по MySQL, чтобы подробнее узнать о мониторинге MySQL.

— Вот уж не думал, что с репликацией может быть столько проблем, — пробормотал он.

Итак, вы уже знаете, когда серверы работают нормально, а когда нет. А как узнать, нормально ли выполняется репликация? Все может работать как надо, но как вы узнаете об этом?

В этой главе мы подробно разберем мониторинг, уделив особое внимание повышению производительности репликации.

Приступаем к работе

Есть два аспекта, влияющих на производительность топологии репликации. Чтобы репликация работала как следует, оба этих аспекта необходимо оптимизировать.

Во-первых, сеть должна обеспечивать достаточную пропускную способность для передачи реплицируемых данных. Как мы уже говорили, главный сервер делает копию изменений и отправляет ее подчиненным серверам по сети. Если сетевое подключение медленное, такой же будет и репликация данных. Мы обсудим некоторые способы настройки сети и репликации, позволяющие получить максимальную производительность в некоторых типах сетевого окружения.

Во-вторых, и это самое важное, реплицируемые БД должны быть оптимизированы. Это жизненно необходимо, так как все недостатки БД на главном сервере будут приводить к такой же медленной работе на подчиненных серверах. В первую очередь это относится к индексированию и нормализации. Тем не менее, хорошо настроенная БД — только половина уравнения. Необходимо также оптимизировать запросы. Плохо написанный запрос с главного сервера будет плохо работать и на подчиненных системах.

Если сеть работает хорошо, а БД и запросы оптимизированы, можно сосредоточиться на настройке серверов с целью получения максимальной производительности.

Настройка сервера

Еще одна очень важная вещь, которую используют для улучшения работы репликации — настройка серверов на максимальную производительность. Часто репликация работает плохо по причине медленно работающих серверов. Убедитесь, что серверы имеют достаточно памяти и выбраны подходящие устройства хранения для БД.

Иногда рекомендуют использовать в качестве подчиненных менее производительные серверы, аргументируя это их меньшей загруженностью (обычно, подчиненные серверы выполняют только запросы SELECT, а главные — модифицируют БД). Однако это неверно. В типичной топологии с одним главным и одним подчиненным сервером при репликации всех БД оба сервера испытывают примерно равную нагрузку. Но при этом подчиненный сервер обрабатывает события в одном потоке, а главный — во многих, так что даже при одинаковой нагрузке подчиненному серверу может потребоваться больше времени для обработки событий.

Вероятно, лучше рассматривать этот вопрос с точки зрения возможности восстановления после сбоя. Предположим, что потребовалось выполнить восстановление после сбоя на главном сервере. При повышении подчиненного сервера до уровня главного от первого потребуется та же производительность, что и от бывшего главного.

Фильтрация данных для репликации

В зависимости от настроек, можно реплицировать все данные (это задано по умолчанию), записывать в журнал подмножество данных, исключив из репликации определенные данные с главного сервера либо явно определив данные, которые требуется реплицировать. Использование частичной репликации помогает решать сложные задачи балансировки нагрузки и масштабирования, расширяя возможности репликации. Еще одно название этого процесса — *фильтрация*, где комбинации условий, включающих данные в репликацию и исключающие из нее данные, образуют критерии фильтрации.

Если нужно записывать в двоичный журнал только события определенной БД, используйте параметр загрузки --binlog-do-db на главном сервере. Можно указать один или несколько таких параметров, указав в каждом параметре одну БД, используя командную строку или файл конфигурации.

Можно также исключить (игнорировать) события в определенной БД, используя параметр загрузки --binlog-ignore-db. Можно указать один или несколько таких параметров, указав одну БД в каждом, используя командную строку или файл конфигурации.

Параметры --binlog-do-db и --binlog-ignore-db можно использовать совместно. Помните об этом, когда будете диагностировать проблемы с репликацией данных (например, причину отсутствия передачи данных на подчиненный сервер). Кроме того, любой из этих параметров фильтрует то, что будет записано в двоичный журнал. Это серьезно ограничивает использование PITR, так как восстанавливать можно только то, что записано в двоичный журнал.

Существует несколько параметров, с помощью которых можно задавать данные, которые будут реплицированы на подчиненные серверы. Параметры binlog, применяемые на главном сервере, имеют дополнительные параметры для установки ограничений на уровне таблицы и даже команды для переименования.

Фильтрация репликации на подчиненном сервере может и не повысить производительность репликации. Хотя объем информации, хранимый на подчиненных серверах может уменьшиться, главный сервер будет передавать им прежний объем данных, а нагрузка по фильтрации, ложащаяся на подчиненный сервер, может свести на нет все преимущества, если критерии фильтрации достаточно сложны. Если нежелательно передавать большой объем данных по сети, лучше выполнять фильтрацию на главном сервере.

При помощи параметра загрузки replicate-do-db на подчиненном сервере можно исключить только те события для определенной БД, которые берутся из журнала ретрансляции. Можно указать один или несколько таких параметров (по одной БД в каждом параметре), используя командную строку или файл конфигурации.

Можно также игнорировать события для определенной БД, используя параметр загрузки --replicate-ignore-db. Таких параметров может быть несколько (по одной БД в каждом параметре), задавать их можно через командную строку или файл конфигурации.

Параметры репликации на подчиненном сервере работают по-разному в зависимости от формата. Это особенно важно для логической репликации, так как возможна потеря данных. Например, если при логической репликации используется параметр --replicate-do-db, подчиненный сервер будет обрабатывать только те команды, за которыми следует команда USE <db>. Если выполнить команду для другой БД, не сменив БД, эта команда будет проигнорирована. Подробнее см. в электронной документации MySQL Reference Manual.

На подчиненном сервере возможна фильтрация репликации на уровне таблиц. Используйте параметры --replicate-do-table и --replicate-ignore-table

для обработки либо исключения событий заданных таблиц. Эти команды очень удобны, когда есть таблицы с данными, не нужными приложению, но важные для администрирования или выполнения каких-то особых задач. Например, в приложении, получающем от поставщика информацию о стоимости чего-либо, можно скрыть эти сведения, если товар поставляется третьей организации для продажи. Чтобы не создавать для продавцов отдельную программу, можно настроить существующее приложение так, что оно использовало подчиненный сервер, на который реплицируется все, кроме таблицы с конфиденциальной информацией.

Кроме этого, существуют варианты двух последних параметров, разрешающие использование шаблонов с символами подстановки. Эти параметры, replicate-wild-do-table и replicate-wild-ignore-table, выполняют те же функции, что и исходные параметры, но поддерживают использование символов подстановки. Например, команда --replicate-wild-do-table=tbl% выполняет события для любых таблиц, имя которых начинается с «tbl» (*tbl*, *tbl1*, *tbl_test* и т. д.) Такой способ фильтрации на подчиненном сервере — еще один инструмент для решения сложных задач репликации.

Также имеется параметр, который можно использовать на подчиненном сервере для переименования или смены БД. Он применяется только к таблицам. Синтаксис таков: --replicate-rewrite-db=»<from><to>» (кавычки необходимы). Этот параметр изменяет только имя БД при обращении к таблицам, но не изменяет имена в командах CREATE DATABASE, ALTER DATABASE и т. д. Он влияет только на те события, для которых указана БД (или выполняет перенаправление БД, используемой по умолчанию, при логической репликации). Этот параметр можно использовать несколько раз для выбора разных БД.

Хотя параметр --replicate-same-server-id нельзя отнести к параметрам фильтрации, он предотвращает бесконечный цикл в круговой репликации. Если указать для него значение 0, подчиненный сервер пропустит события с тем же server_id, а значение 1 заставит подчиненный сервер обрабатывать все события.

Потоки репликации

Прежде чем говорить о мониторинге главного и подчиненного серверов, следует изучить потоки, вовлеченные в репликацию. Репликацией управляют три потока, каждый со своей ролью. На главном сервере для каждого подключенного подчиненного сервера создается один поток с именем Binlog Dump. Этот поток отвечает за отправку событий двоичного журнала подключенным подчиненным серверам. На подчиненном сервере создается два потока: Slave IO и Slave SQL. Первый отвечает за чтение входящих событий двоичного журнала с главного сервера и запись их в журнал ретрансляции подчиненного сервера. Второй поток отвечает за чтение событий из журнала ретрансляции и их выполнение.

За текущим состоянием потока Binlog Dump можно наблюдать при помощи команды SHOW PROCESS LIST:

```
mysql> SHOW PROCESSLIST \G
*************************** 1. row ***************************
      Id: 1
    User: rpl
    Host: localhost:54197
      db: NULL
 Command: Binlog Dump
    Time: 25
   State: Master has sent all binlog to slave; waiting for binlog to be updated
    Info: NULL
```

Обратите внимание на столбец `State`, в котором описано, что выполняет главный сервер над двоичным журналом и подчиненным сервером. Пример, приведенный выше, является типичным результатом правильно работающей топологии репликации. Выходные данные включают следующие столбцы:

- *Id* — показывает ID подключения;
- *User* — показывает пользователя, выполнившего команду;
- *Host* — хост, на котором была выполнена исходная команда;
- *db* — здесь указана БД по умолчанию; если база данных по умолчанию не была указана, отображается NULL;
- *Command* — тип команды, которую выполняет поток. Подробнее см. в электронной документации MySQL Reference Manual;
- *Time* — время нахождения потока в указанном состоянии (в секундах);
- *State* — описание текущего действия или состояния (например, ожидание). Обычно это содержательное текстовое сообщение;
- *Info* — команда, которую выполняет поток. Значение NULL показывает, что в текущий момент нет выполняющихся команд. Это значение появляется, когда потоки репликации находятся в состоянии ожидания.

На подчиненном сервере тоже можно посмотреть состояние потоков. Для этого используется команда SHOW PROCESSLIST:

```
mysql> SHOW PROCESSLIST \G
*************************** 1. row ***************************
     Id: 2
   User: system user
   Host: db: NULL
Command: Connect
   Time: 127
  State: Waiting for master to send event Info: NULL
*************************** 2. row ***************************
     Id: 3
   User: system user
```

```
   Host:
     db: NULL
Command: Connect
   Time: 10
  State: Slave has read all relay log; waiting for the slave I/O thread to update it
   Info: NULL
```

Наиболее важную информацию снова содержит столбец State. Если на подчиненном сервере возникают проблемы с репликацией, выполните на нем команду SHOW PROCESSLIST и проверьте состояния потоков I/O и SQL. В приведенном примере видно, что состояние подчиненного сервера в норме: он ожидает информацию от главного сервера (поток I/O) и выполняет все события из журнала ретрансляции (поток SQL).

При устранении сбоев рекомендуется всегда проверять состояние репликации при помощи команды SHOW PROCESSLIST.

Мониторинг главного сервера

Существует несколько способов мониторинга главного сервера. Можно выполнять команды SHOW и просматривать переменные и данные состояния, а можно использовать MySQL Administrator. Основными командами SQL для мониторинга являются SHOW MASTER STATUS, SHOW BINARY LOGS и SHOW BINLOG EVENTS.

В этом разделе мы рассмотрим команды SQL, используемые для мониторинга главного сервера, и вкратце опишем доступные переменные состояния, за которыми можно следить при помощи команды SHOW STATUS или графиков, построенных в MySQL Administrator.

Команды для мониторинга главного сервера

Команда SHOW MASTER STATUS показывает информацию о двоичном журнале главного сервера, включая имя и смещение текущего файла двоичного журнала. Эта информация жизненно важна для подключения подчиненных серверов, о чем мы говорили в предыдущих главах. Также эта команда предоставляет информацию об ограничениях журналирования. В листинге 10-1 показаны типичные результаты выполнения команды SHOW MASTER STATUS.

Лист. 10-1. Команда SHOW MASTER STATUS

```
mysql> SHOW MASTER STATUS;
+------------------+---------- +--------------+------------------+
| File             | Position  | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+-----------+--------------+------------------+
| mysql-bin.000002 | 156058362 | Inventory    | Vendor_sales     |
+------------------+-----------+--------------+------------------+
1 row in set (0.00 sec)
```

Данные отображаются в следующих столбцах:

- *File* — имя текущего файла двоичного журнала;
- *Position* — текущая позиция (следующая запись) в двоичном журнале;
- *Binlog_Do_DB* — содержит имена БД, указанных загрузочным параметром --binlog-do-db, описанным выше;
- *Bnlog_Ignore_DB* — содержит имена БД, указанных загрузочным параметром --binlog-ignore-db, описанным выше.

Команда SHOW BINARY LOGS (или SHOW MASTER LOGS) показывает список файлов двоичного журнала, доступных на главном сервере, и их размер в байтах. Эта команда полезна для сравнения позиции подчиненного сервера по отношению к главному, т. е. для определения того, какой двоичный журнал в текущий момент читается с главного сервера. В листинге 10-2 показаны типичные результаты выполнения команды SHOW BINARY LOGS.

Лист. 10-2. Выполнение команды SHOW MASTER LOGS на главном сервере

```
mysql> SHOW MASTER LOGS;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| master-bin.000001 | 103648205 |
| master-bin.000002 | 2045693   |
| master-bin.000003 | 1022910   |
| master-bin.000004 |  3068436  |
+------------------+-----------+
4 rows in set (0.00 sec)
```

Файлы двоичного журнала можно менять командой FLUSH LOGS. Эта команда закрывает и повторно открывает все журналы и открывает новый журнал со следующим по порядку номером (в расширении). Следует периодически начинать новый файл журнала, чтобы ограничивать его рост со временем. Также это помогает в поиске проблем с репликацией.

Кроме этого, команду SHOW BINLOG EVENTS можно использовать для просмотра событий из двоичного журнала. Эта команда имеет следующий синтаксис:

```
SHOW BINLOG EVENTS [IN <log>] [FROM <pos>] [LIMIT [<offset>,] <rows>]
```

Используйте эту команду с осторожностью, так как она может возвращать большой объем данных. Лучше всего ее использовать для сравнения событий на главном и подчиненном серверах. На лист. 10-3 показаны события двоичного журнала в типичной конфигурации репликации.

Лист. 10-3. Команда SHOW BINLOG EVENTS (логическая репликация)

```
mysql> SHOW BINLOG EVENTS IN 'master-bin.000001' FROM 2571 LIMIT 4 \G
*************************** 1. row ***************************
   Log_name: master-bin.000001
        Pos: 2571
 Event_type: Query
  Server_id: 1
End_log_pos: 2968
       Info: use `employees`; CREATE TABLE salaries (
   emp_no      INT             NOT NULL,
   salary      INT             NOT NULL,
   from_date   DATE            NOT NULL,
   to_date     DATE            NOT NULL,
   KEY      (emp_no),
   FOREIGN KEY (emp_no) REFERENCES employees (emp_no)
   ON DELETE CASCADE,
   PRIMARY KEY (emp_no, from_date)
)
*************************** 2. row ***************************
   Log_name: master-bin.000001
        Pos: 2968
 Event_type: Query
  Server_id: 1 End_log_pos: 3041
       Info: BEGIN
*************************** 3. row ***************************
   Log_name: master-bin.000001 Pos: 3041 Event_type: Query Server_id: 1
End_log_pos: 3348
       Info: use `employees`; INSERT INTO `departments` VALUES
       ('d001','Marketing'),('d002','Finance'),('d003','Human Resources'),
       ('d004','Production'),('d005','Development'),('d006','Quality
       Management'),('d007','Sales'),('d008','Research'),('d009',
       'Customer Service')
*************************** 4. row ***************************
   Log_name: master-bin.000001
        Pos: 3348
 Event_type: Xid
  Server_id: 1
End_log_pos: 3375
       Info: COMMIT /* xid=17 */
4 rows in set (0.01 sec)
```

В этом примере используется логическая репликация. Если бы использовалась построчная репликация, события двоичного журнала выглядели бы совершенно иначе (см. лист. 10-4).

Лист. 10-4. Команда SHOW BINLOG EVENTS (построчная репликация)

```
mysql> SHOW BINLOG EVENTS IN 'master-bin.000001' FROM 2571 LIMIT 4 \G
*************************** 1. row ***************************
   Log_name: master-bin.000001
        Pos: 2571
 Event_type: Query
  Server_id: 1
End_log_pos: 2968
       Info: use `employees`; CREATE TABLE salaries (
   emp_no      INT             NOT NULL,
   salary      INT             NOT NULL,
   from_date   DATE            NOT NULL,
   to_date     DATE            NOT NULL,
   KEY         (emp_no),
   FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
   PRIMARY KEY (emp_no, from_date)
)
*************************** 2. row ***************************
   Log_name: master-bin.000001
        Pos: 2968
 Event_type: Query
  Server_id: 1
End_log_pos: 3041
       Info: BEGIN
*************************** 3. row ***************************
   Log_name: master-bin.000001
        Pos: 3041
 Event_type: Table_map
  Server_id: 1
End_log_pos: 3101
       Info: table_id: 15 (employees.departments)
*************************** 4. row ***************************
   Log_name: master-bin.000001
        Pos: 3101
 Event_type: Write_rows
  Server_id: 1
End_log_pos: 3292
       Info: table_id: 15 flags: STMT_END_F 4
rows in set (0.01 sec)
```

Обратите внимание на то, что при использовании построчной репликации в двоичном журнале гораздо меньше информации. Поэтому при анализе сложных проблем, связанных с повреждением данных или нерегулярными сбоями, полезно включать логическую репликацию. Например, так можно проверить, что именно записывается в двоичный журнал на главном сервере, и сравнить с тем, что читается из журнала ретрансляции на подчиненном

сервере. Различия в этом случае проще обнаружить при регистрации самих команд, а не результатов их исполнения, представленных лишь в машиночитаемом виде. Подробнее о форматах двоичного журнала, их преимуществах и недостатках см. в главе 2.

Переменные состояния на главном сервере

Переменных состояния для мониторинга главного сервера немного. Они ограничены счетчиками, показывающими, сколько раз на главном сервере были выполнены команды, относящиеся к главному серверу.

- *Com_change_master* Показывает, сколько раз была выполнена команда CHANGE MASTER. Если это значение часто изменяется или значительно превышает число серверов, умноженное на число запланированных перезапусков подчиненных серверов, это означает, что подчиненные серверы перезапускаются слишком часто, и может указывать на нестабильность подключения.
- *Com_show_master_status* Показывает, сколько раз была выполнена команда SHOW MASTER STATUS. Как и для предыдущей переменной, большие значения этого счетчика могут указывать на слишком высокое число запросов переподключения подчиненных серверов.

Мониторинг подчиненных серверов

Существует несколько способов мониторинга подчиненных серверов. Можно использовать команд SHOW для просмотра информации о состоянии и переменных состояния, а можно использовать MySQL Administrator. Основные команды SQL, используемые для мониторинга: SHOW SLAVE STATUS, SHOW BINARY LOGS и SHOW BINLOG EVENTS.

В этом разделе мы рассмотрим команды SQL, используемые для мониторинга подчиненных серверов, и вкратце опишем переменные состояния, за которыми можно наблюдать при помощи команды SHOW STATUS или графиков, построенных в MySQL Administrator. Об использовании MySQL Administrator рассказывается далее в разделе «Мониторинг репликации при помощи MySQL Administrator».

Команды для мониторинга подчиненных серверов

Команда SHOW SLAVE STATUS показывает информацию о двоичном журнале подчиненного сервера, его подключении к серверу и активности репликации, включая имя и позицию смещения текущего файла двоичного журнала. Эта информация жизненно важна для проверки производительности подчиненного сервера, о чем мы говорили в предыдущих главах. На лист. 10-5 показаны типичные результаты выполнения команды SHOW SLAVE STATUS на сервере MySQL 5.5.

Лист. 10-5. Команда SHOW SLAVE STATUS

```
mysql> SHOW SLAVE STATUS \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: localhost
                  Master_User: rpl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000002
          Read_Master_Log_Pos: 39016226
               Relay_Log_File: relay-bin.000004
                Relay_Log_Pos: 9353715
        Relay_Master_Log_File: mysql-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 25263417
              Relay_Log_Space: 39016668
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 66
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
1 row in set (0.00 sec)
```

Информации выводится много, но это самая важная для репликации команда. Рекомендуем подробно изучить каждый представленный здесь

элемент. Вместо того чтобы описывать элемент за элементом, мы отметим самое важное с точки зрения администратора. Данные обычно проверяются с определенной целью, поэтому мы разбили их по категориям, чтобы облегчить поиск. Эти категории включают информацию о подключении к главному серверу, производительности подчиненного сервера, самих журналах, фильтрации, производительности журналов и ошибках.

Наиболее важные сведения находятся в первом столбце. Здесь сообщается о текущем состоянии потока ввода-вывода: подключение к главному серверу, ожидание событий с главного сервера, повторное подключение к главному серверу и т. д.

Сведения о подключении к главному серверу включают текущее имя главного сервера, учетную запись, используемую для подключения, и порт подключения. Ближе к концу листинга находится информация о подключении SSL (если такое подключение используется).

Следующая категория включает информацию о двоичном журнале на главном сервере и журнале ретрансляции на подчиненном. Отображается имя файла и позиция каждого журнала. Важно просматривать эти значения при анализе проблем с репликацией. Особенно важно значение Relay_Master_Log_File. Оно показывает имя файла двоичного журнала главного сервера, из которого получено последнее обработанное событие.

В конфигурации фильтрации отражены все фильтры репликации, установленные на подчиненном сервере. Проверяйте эти данные, если не уверены в настройках фильтров.

Включены также код и описание последней ошибки подчиненного сервера, потока ввода-вывода и потока SQL. В случае ошибки эта информация приобретает первостепенное значение, наряду с данными о состоянии потоков подчиненного сервера. Эту информацию проверяют в первую очередь, до проверки журнала ошибок, так как она является наиболее свежей и обычно позволяет понять причину ошибки.

Также имеется информация о конфигурации подчиненного сервера, включая параметры счетчика пропусков и условий until. Подробнее об этом см. в электронной документации MySQL Reference Manual.

В нижней части списка приводится текущая информация об ошибках, включая ошибки потоков SQL и ввода-вывода подчиненного сервера. Если сервер функционирует должным образом, эти значения всегда должны быть равны 0.

Теперь подробнее рассмотрим некоторые из наиболее важных столбцов с данными о производительности:

- *Connect_Retry* Время в секундах между повторными попытками подключения. Это значение всегда должно быть небольшим, оно увеличивается, когда возникают проблемы с подключением подчиненного сервера к главному.

- *Exec_Master_Log_Pos* Показывает позицию последнего события, выполненного из двоичного журнала главного сервера.
- *Relay_Log_Space* Общий размер всех файлов журнала ретрансляции. Показывает, следует ли очистить журналы, если на диске осталось мало места.
- *Seconds_Behind_Master* Время в секундах между выполнением команды и записью соответствующего события в двоичный журнал главного сервера. Высокое значение может указывать на значительную задержку репликации. О задержках репликации будет рассказано далее в этой главе.

Значение Seconds_Behind_Master может стать неактуальным, если репликация остановилась из-за сбоя сети, потери heartbeat-импульсов главного сервера и т. д. Оно имеет смысл, только когда репликация работает.

Если на подчиненном сервере включена запись в двоичный журнал, команда SHOW BINARY LOGS отображает список файлов двоичного журнала, доступных на подчиненном сервере, и их размеры в байтах. В листинге 10-6 показаны типичные результаты выполнения команды SHOW BINARY LOGS.

Лист. 10-6. Команда SHOW BINARY LOGS на подчиненном сервере

```
mysql> SHOW BINARY LOGS;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| slave-bin.000001 | 5151604   |
| slave-bin.000002 | 1030108   |
| slave-bin.000003 |  1030044  |
+------------------+-----------+
3 rows in set (0.00 sec)
```

Менять журнал ретрансляции на подчиненном сервере можно командой FLUSH LOGS.

Также можно использовать команду SHOW BINLOG EVENTS для отображения событий из двоичного журнала на подчиненном сервере (если на этом сервере включено ведение двоичного журнала). Разница между отображением событий на подчиненном и на главном серверах состоит в том, что на подчиненном сервере можно указать имя файла двоичного журнала, как показано в выходных данных команды SHOW BINARY LOGS. На лист. 10-7 показаны события из двоичного журнала в типичной конфигурации репликации.

Лист. 10-7. Команда SHOW BINLOG EVENTS (логическая репликация)

```
mysql> SHOW BINLOG EVENTS IN 'slave-bin.000001' FROM 2701 LIMIT 2 \G
*************************** 1. row ***************************
   Log_name: slave-bin.000001
```

```
        Pos: 2701
 Event_type: Query
  Server_id: 1
End_log_pos: 3098
       Info: use `employees`; CREATE TABLE salaries (
   emp_no      INT              NOT NULL,
   salary      INT              NOT NULL,
   from_date   DATE             NOT NULL,
   to_date     DATE             NOT NULL,
   KEY         (emp_no),
   FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
   PRIMARY KEY (emp_no, from_date)
)
*************************** 2. row ***************************
   Log_name: slave-bin.000001
        Pos: 3098
 Event_type: Query
  Server_id: 1
End_log_pos: 3405
       Info: use `employees`; INSERT INTO `departments` VALUES
       ('d001','Marketing'),('d002','Finance'),
       ('d003','Human Resources'),('d004','Production'),
       ('d005','Development'),('d006','Quality Management'),
       ('d007','Sales'),('d008','Research'),
       ('d009','Customer Service') 2 rows in set (0.01 sec)
```

В MySQL 5.5 и выше можно также проверять журнал ретрансляции подчиненного сервера при помощи команды SHOW RELAYLOG EVENTS.

Переменные состояния на подчиненном сервере

Переменных состояния для мониторинга подчиненных серверов также немного. К ним относятся счетчики, указывающие, сколько раз на главном сервере были выполнены команды, относящиеся к подчиненным серверам, а также статистические данные для ключевых операций, связанных с подчиненными серверами. Первые четыре переменные из следующего списка являются простыми счетчиками для различных команд. Их значения должны соответствовать частоте выполнения служебных операций на подчиненных серверах. В противном случае возможно, что в топологию включено больше подчиненных серверов, чем вы думаете, или какой-то из подчиненных серверов перезапускается слишком часто.

`Com_show_slave_hosts`

Показывает, сколько раз выполнена команда SHOW SLAVE HOSTS.

`Com_show_slave_status`

Показывает, сколько раз выполнена команда SHOW SLAVE STATUS.

`Com_slave_start`

Показывает, сколько раз выполнена команда SLAVE START.

`Com_slave_stop`

Показывает, сколько раз выполнена команда SLAVE STOP.

`Slave_heartbeat_period`

Текущая конфигурация, определяющая время в секундах между проверками тактовых импульсов главного сервера.

`Slave_open_temp_tables`

Число временных таблиц, используемых потоком SQL подчиненного сервера. Высокое значение может указывать на то, что сервер перегружен.

`Slave_received_heartbeats`

Число ответов на heartbeat-импульсы, полученных от главного сервера. Это значение должно примерно соответствовать времени с момента последнего перезапуска подчиненного сервера, деленному на интервал между импульсами.

`Slave_retried_transactions`

Число попыток повторного выполнения транзакций, предпринятых потоком SQL с момента запуска подчиненного сервера.

`Slave_running`

Имеет значение ON, если подчиненный сервер подключен к главному, а потоки ввода-вывода и SQL работают без ошибок.

Мониторинг репликации при помощи MySQL Administrator

В предыдущих главах мы показали, как использовать MySQL Administrator для мониторинга сетевого трафика и механизмов БД. Эта утилита также поддерживает собранные на одной странице простые функции для мониторинга главного и подчиненных серверов в топологии репликации. На вкладке Replication Status (Состояние репликации) можно просматривать базовую информацию о репликации. Однако для получения максимальной пользы от этих данных, подчиненные серверы следует запускать с параметром загрузки --report host.

На рис. 10-1 показано окно утилиты MySQL Administrator, запущенной на главном сервере с одним подключенным подчиненным сервером. Если бы были подчиненные серверы, подключенные без параметра --report host, они бы не вошли в список.

Если запустить MySQL Administrator на подчиненном сервере, будет показана только информация об этом подчиненном сервере (рис. 10-2).

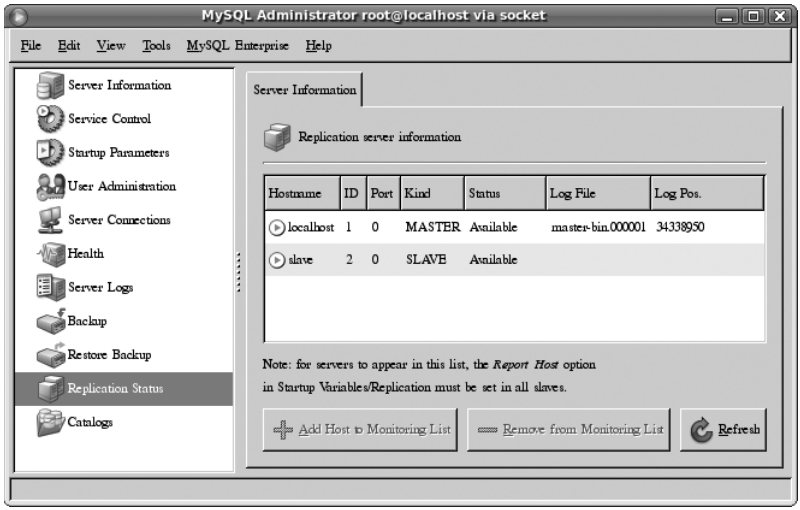


Рис. 10-1. MySQL Administrator на главном сервере

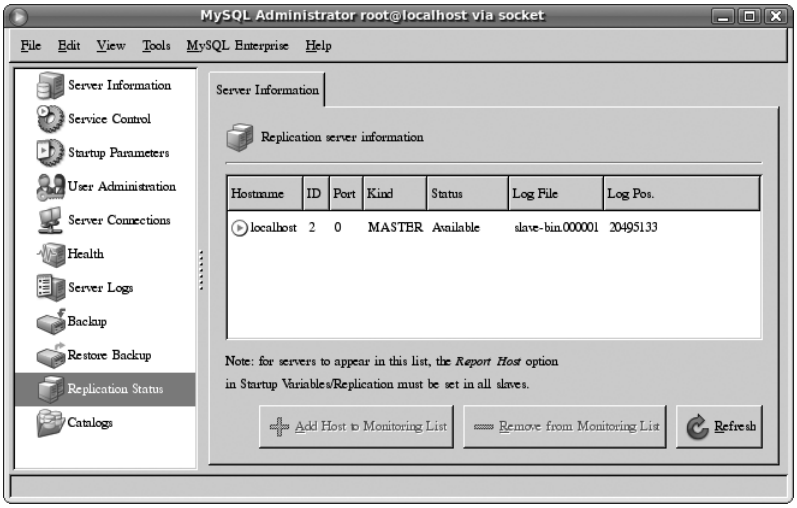


Рис. 10-2. MySQL Administrator на подчиненном сервере

Данные в окнах на рис. 10-1 и 10-2 включают имя сервера, ID сервера, порт, тип (главный или подчиненный), общее состояние, имя файла двоичного журнала и текущую позицию в журнале. На рис. 10-1 показана топология репликации с перечислением всех подключенных подчиненных серверов. Это удобно, когда нужно быстро просмотреть состояние серверов.

Прочие элементы

В этом разделе обсуждаются дополнительные вопросы мониторинга репликации, а именно некоторые нюансы работы в сети и задержки.

Вопросы, связанные с сетями

Если пропускная способность сети ограничена, сеть загружена или у вас просто очень медленное подключение, можно улучшить производительность репликации, используя сжатие. Сжатие настраивается при помощи переменной slave_compressed_protocol.

Если экономить пропускную способность не требуется, но нужно защитить данные, передающиеся от главного сервера подчиненным, можно использовать подключение SSL. Для настройки такого подключения используйте команду CHANGE MASTER. Подробнее см. в разделе «Setting Up Replication Using SSL» (Настройка репликации с использованием SSL) электронной документации MySQL Reference Manual.

Еще один важный вопрос — использование heartbeat-импульсов главного сервера. О них говорилось в описании команды SHOW SLAVE STATUS. Это механизм автоматической проверки состояния подключения подчиненного сервера к главному, позволяющий определить скорость подключения в миллисекундах. Heartbeat-импульсы главного сервера используются, когда подчиненные серверы должны синхронизироваться с главным с минимальной задержкой. Возможность определить превышение порогового времени позволяет выявить задержку до остановки репликации на подчиненном сервере.

Настроить heartbeat-импульсы можно при помощи параметра команды CHANGE MASTER со значением master heartbeat period=<*значение*> (появилось в MySQL 5.4.4), где *значение* — число секунд между отправкой импульсов. Наблюдать за состоянием импульсов можно при помощи следующих команд:

```
SHOW STATUS like 'slave_heartbeat period'
SHOW STATUS like 'slave_received_heartbeats'
```

Задержки подчиненного сервера

Массовые модификации, перегрузка подчиненных серверов или прочие события, снижающие производительность сети, могут привести к отставанию подчиненных серверов от главного. Когда это происходит, подчиненные серверы не успевают обрабатывать события из своих журналов ретрансляции вслед за изменениями, которые рассылает главный сервер.

Как вы уже видели при работе с командой SHOW SLAVE STATUS, поле Seconds Behind Master может указывать на отставание подчиненного сервера от главного. Это поле показывает, на сколько секунд поток SQL подчиненного сервера отстает от потока ввода-вывода, т. е. на сколько задерживается

обработка поступающих от главного сервера событий. Для вычисления этого значения подчиненный сервер использует временные отметки событий. Когда поток SQL на подчиненном сервере читает событие, полученное от главного сервера, он вычисляет разность временных отметок. В следующем примере подчиненный сервер на 146 секунд отстает от главного, что может привести к сбоям приложений, требующих актуальной информации.

```
mysql> SHOW SLAVE STATUS \G
...
      Seconds_Behind_Master: 146
...
```

Команда SHOW PROCESSLIST, выполненная на подчиненном сервере, тоже может показать время задержки. Здесь можно увидеть время отставания потока SQL в секундах, измеренное по разнице между временными отметками последнего реплицированного события и текущим временем подчиненного сервера. Например, если подчиненные серверы были отключены от сети на 30 минут и после были повторно подключены к главному серверу, в поле Time выходных данных команды SHOW PROCESSLIST будет значение примерно в 1800 секунд (см. пример ниже). Большие значения в этом поле указывают на значительные задержки, которые могут привести к устареванию данных на подчиненных серверах.

```
mysql> SHOW PROCESSLIST \G
...
Time: 1814
...
```

В зависимости от топологии, репликация может осуществляться и для балансировки нагрузки. В таких случаях обычно используется несколько подчиненных серверов, которые берут на себя обработку части запросов от приложений или пользователей, снижая тем самым нагрузку на главный сервер.

Причины задержки подчиненных серверов и способы их устранения

Задержки подчиненных серверов могут доставлять неприятности некоторым пользователям репликации. Основная причина задержек — однопоточность подчиненного сервера (в действительности потоков два, но только один из них обрабатывает события, что и является основной причиной задержки). Например, главный сервер с многоядерным процессором может параллельно выполнять несколько транзакций и будет работать быстрее однопоточного подчиненного сервера. Некоторые способы выявления задержек подчиненных серверов описаны выше, а в этом разделе рассказывается о типичных причинах и способах их устранения.

Существует несколько причин задержек на подчиненных серверах (например, задержка при передаче по сети). Возможно, что поток ввода-вывода

подчиненного сервера «тормозит» при чтении событий из журналов. Наиболее распространенная причина задержки проста — подчиненный сервер имеет только один поток для обработки всех событий, а главный сервер может иметь много потоков, работающих параллельно. К другим причинам относятся долго выполняющиеся запросы с неэффективными операциями объединения, операции чтения с диска, конфликты при блокировках и проблемы с параллельной работой потоков InnoDB.

Теперь, когда вы немного разобрались с причинами задержек, рассмотрим некоторые способы их минимизации.

- *Оптимизируйте данные* Производительность можно улучшить, выполнив нормализацию данных и используя шардинг (sharding). Это позволяет избежать дублирования данных, но, как вы знаете из гл. 8, дублирование может повышать производительность. Идея состоит в том, чтобы нормализовать и сегментировать данные в достаточной для повышения производительности степени, но не заходить слишком далеко. Что делать в конкретном случае, может определить только владелец данных, исходя из опыта или просто экспериментируя.
- *Разделяй и властвуй* Мы знаем, что добавление подчиненных серверов для обработки запросов (т. н. горизонтальное масштабирование) — хороший способ повышения производительности. Но задержки все равно возникают, если подчиненные серверы все еще обрабатывают большое число запросов. В критических ситуациях задержки могут возникать сразу на всех подчиненных серверах. Чтобы решить эту проблему, попробуйте изолировать данные с помощью фильтрации репликации, чтобы направить разным подчиненным серверам разные БД. Конечно, можно продолжить масштабирование, но тогда настройте промежуточный подчиненный сервер для каждой группы фильтруемых БД, а затем уже выполняйте масштабирование.
- *Выявляйте долго выполняющиеся запросы и изменяйте их* Если причиной задержки являются долго выполняющиеся запросы, подумайте, как переписать запрос, изменить операцию или само приложение, чтобы можно было обойтись более короткими запросами и транзакциями. Однако при использовании этого приема совместно с фильтрацией репликации необходимо следить за транзакциями, охватывающими разные группы фильтрации. Если разделить длинный запрос, который должен быть единой и неделимой операцией (транзакцией), между подчиненными серверами, могут возникнуть проблемы целостности данных.
- *Балансировка нагрузки* Балансировку нагрузки можно использовать для перенаправления запросов на разные подчиненные серверы. Это может уменьшить время ответа подчиненных серверов на запросы и освободить больше ресурсов для обработки событий репликации.
- *Используйте современное оборудование* Чем лучше оборудование, тем выше производительность, это очевидно. Как минимум, убедитесь, что

серверы настроены должным образом и оптимально используют свои аппаратные возможности. Подчиненные серверы должны быть, по крайней мере, столь же мощными, как главный сервер.

- *Устраняйте конфликты блокировок* Блокировка таблиц MyISAM и блокировка на уровне строк в InnoDB может привести к задержке. Если какой-то запрос приводит к частым блокировкам таблиц MyISAM или InnoDB, попробуйте переписать его так, чтобы свести число блокировок к минимуму.

Заключение

Эта глава завершает обсуждение мониторинга MySQL и закладывает основы для реализации собственных схем мониторинга практически любых аспектов серверов MySQL.

Итак, вы изучили основы мониторинга операционной системы, производительности БД и серверов MySQL, и получили инструменты и знания, необходимые для настройки серверов с целью достижения их оптимальной производительности.

Джоэл улыбнулся, закончив отчет о решенной им проблеме с репликацией, но тут же замер и взглянул на дверь, догадываясь, что сейчас будет.

— Джоэл!

Джоэл вздрогнул, не веря своему предвидению.

— Сэр, я решил эту проблему с репликацией! — выпалил он.

— Замечательно! Отправь мне отчет, когда будет время.

— Я также обнаружил кое-что интересное для нашей системы обработки заказов.

Джоэл заметил, как г-н Саммерсон приподнял бровь в ожидании.

— Кажется, мы неправильно установили размеры пула и буфера. Думаю, я могу попробовать улучшить и это.

— Опять мониторинг? — спросил г-н Саммерсон.

— Да, сэр. Я собрал некоторые отчеты по механизму InnoDB, отправлю их вам по почте.

— Хорошая работа. Действительно хорошая работа!

Джоэл поймал на себе знакомый взгляд. Его руководитель о чем-то напряженно раздумывал, и это всегда означало новое задание. Поэтому Джоэл удивился, когда г-н Саммерсон просто медленно вышел.

— Хм, кажется, я, наконец-то, впечатлил его...

ГЛАВА 11

Устранение неполадок репликации

Тема сообщения была простой: «Наладить сервер в Сиэтле». Джоэл знал, что такие туманные строчки могли прийти только от одного человека. Быстрый взгляд на заголовок сообщения подтвердил, что это г-н Саммерсон. Джоэл открыл сообщение и прочел его.

«Сервер в Сиэтле опять капризничает. Я думаю, дело в репликации. Разберись с этим в первую очередь».

— Хорошо, — пробормотал Джоэл. Так как отчеты по мониторингу, созданные на прошлой неделе, не показали никаких отклонений, и он был уверен, что при последней проверке репликация была настроена правильно, Джоэл не знал, с чего начать. Зато он знал, где искать ответы.

— Похоже, нужно все-таки прочитать ту главу про решение проблем с репликацией.

В дверном проеме показалась знакомая фигура. Джоэл решил выполнить упреждающий маневр.

— Я уже работаю над проблемой!

Фигура, оказавшаяся его боссом, кивнула, приветственно махнула рукой и прошла мимо.

Репликация MySQL обычно не вызывает проблем и редко требует вмешательства в настройки, если топология работает и правильно настроена. И все же неполадки могут возникать. Например, может появиться сообщение об ошибке, и вам понадобятся четкие указания на то, с чего начинать ее устранение. Или бывает, что решение простых проблем ведет к появлению более сложных, с которыми не так легко разобраться. К счастью, все такие проблемы можно решить, если придерживаться простых правил и советов.

В этой главе представлены соображения по решению проблем репликации. Сначала мы расскажем, что может пойти не так, затем обсудим основные инструменты, помогающие решить возникшие проблемы, а в конце приведем некоторые стратегии решения и предотвращения проблем с репликацией.

Приемы, описанные в этой главе, применимы также для решения проблем репликации с участием MySQL Cluster. Информацию о решении проблем, связанных со сбоями кластера, и проблем с запуском см. в гл. 15.

Опытные пользователи понимают, что в компьютерных системах время от времени случаются сбои. Одной из задач ИТ-профессионалов является предотвращение таких сбоев и предоставление пользователям надежного доступа к данным. Тем не менее, проблемы могут возникать даже в системах, получающих надлежащее обслуживание.

Репликация MySQL не исключение. В частности, не защищено от сбоев подчиненное состояние. Это значит, что при сбое в экземпляре MySQL на подчиненном сервере этот сервер может остаться в неопределенном состоянии. В худшем случае при этом может быть поврежден журнал ретрансляции или файл *master.info*.

Вообще, чем сложнее топология и структура баз данных и выше нагрузка, и чем разнообразнее роли узлов топологии, тем выше вероятность сбоев. Впрочем, это не означает, что репликацию невозможно масштабировать. Наоборот: вы уже видели, как легко масштабируется репликация в обширных топологиях. Другими словами, неполадки репликации обычно возникают в результате неожиданного действия или изменения конфигурации.

Возможные причины проблем

Существует много факторов, способных прервать репликацию. Репликация MySQL наиболее чувствительна к проблемам с данными, будь то повреждение данных или непреднамеренное вмешательство в поток репликации. Системные сбои, приводящие к небезопасной и неконтролируемой остановке MySQL, также могут вызвать проблемы с перезапуском репликации.

Прежде чем что-то изменять в попытках решить проблему, следует подготовить резервную копию данных. В некоторых случаях резервная копия будет содержать поврежденные или отсутствующие данные, но преимущества такого подхода все равно очень весомы. Как минимум, это позволяет вернуть данные в состояние на момент ошибки, независимо от дальнейших действий — ведь даже самую плохую ситуацию на удивление легко ухудшить!

В этом разделе мы опишем наиболее распространенные сбои в репликации MySQL. Некоторые проблемы возникают гораздо чаще других. Этот список не полный (он не включает все возможные проблемы), но дает представление о том, какого рода неприятности могут случиться во время репликации. Мы приведем краткое описание наиболее вероятных причин каждой проблемы.

Проблемы на главном сервере

В большинстве случаев проблемы возникают на подчиненных серверах, и администраторы автоматически подозревают подчиненные серверы, забывая про главный. Тем не менее, при поиске источника проблемы репликации следует проверять и главный сервер.

Хранимые в памяти таблицы продолжают использоваться после сбоя главного сервера

При перезапуске главного сервера таблицы, хранимые в памяти, очищаются (как и у любой СУБД). Если же таблица, хранящаяся в оперативной памяти, реплицируется, то на подчиненный сервер могут попасть устаревшие данные, если не перезапустить главный сервер после сбоя.

К счастью, при первом после перезапуска обращении к таблице памяти специальное событие удаления сигнализирует подчиненным серверам о необходимости очистить данные, автоматически синхронизируя их. Однако задержка между обращением к таблице и отправкой события репликации может привести к тому, что подчиненный сервер будет иметь устаревшие данные. Чтобы избежать этой проблемы, используйте сценарий, который сначала очищает таблицы, а затем снова загружает их в память на главном сервере. Такой сценарий должен выполняться при запуске с помощью параметра init file:

```
#  Принудительная очистка данных на подчиненной системе
DELETE FROM db1.mem_zip;
#  Повторная загрузка данных
INSERT INTO ...
```

Первая команда — запрос удаления, который будет реплицирован на подчиненные серверы при перезапуске репликации. Далее идут операторы для повторного заполнения хранимой в памяти таблицы. Такой способ гарантирует отсутствие задержки, во время которой в памяти подчиненного сервера будет устаревшая таблица.

Произошел сбой главного сервера, двоичные журналы событий отсутствуют

Возможна ситуация, в которой главный сервер вышел из строя, не записав последние события в двоичный журнал. То есть, если сбой на сервере произойдет прежде, чем MySQL сбросит кэш двоичных событий на диск (в двоичный журнал), эти кэшированные события могут быть потеряны.

На такую проблему обычно указывает ошибка на подчиненном сервере, сообщающая о том, что событие смещения двоичного журнала отсутствует или не существует. В таком случае подчиненный сервер пытается повторно подключиться при перезапуске, используя последний известный файл двоичного журнала и номер записи журнала на главном сервере. Если файл двоичного журнала может существовать, то смещение — нет, так как события, увеличивающие смещение, не были записаны на диск.

К сожалению, способа вернуть потерянные события двоичного журнала не существует. Для решения такой проблемы необходимо проверить текущую позицию в двоичном журнале на главном сервере. По результатам проверки нужно приказать подчиненному серверу стартовать со следующего

известного события на главном сервере. После синхронизации подчиненного сервера проверьте данные на обоих серверах.

Также вероятно, что некоторые из потерянных на главном сервере событий были применены к данным до сбоя. Поэтому всегда следует сверять таблицы, чтобы выяснить, есть ли различия между копиями на главном и подчиненном серверах. Такая ситуация возникает редко, но может вызвать проблемы в дальнейшем, если на главном сервере изменится строки, ранее измененная в ходе потерянного события. Репликация этих действий на подчиненном сервере приведет к сбою: подчиненный сервер будет пытаться обновить несуществующие строки.

Например, рассмотрим ситуацию с вымышленной базой данных автодилера, в которой хранится информация о продающихся автомобилях. Для новых и подержанных машин используются отдельные таблицы, в которых установлены автоматически увеличивающиеся ключевые значения.

На главном сервере выполняется следующее:

```
INSERT INTO auto.used_cars VALUES (2004, 'Porsche', 'Cayman', 23100, 'синий');
```

Сбой происходит после выполнения следующей команды, но до того, как она записана в двоичный журнал:

```
UPDATE auto.used_cars SET color = 'белый' WHERE id = 17;
```

Таким образом, запрос на обновление потерян во время сбоя на главном сервере. При перезапуске подчиненного сервера возникает ошибка. Эту проблему можно решить, используя только что приведенный совет. Проверка числа строк на главном и подчиненном сервере покажет, что число совпадает. Обратите внимание на изменение цвета автомобиля Porsche 2004 года с синего на белый. Теперь представьте, что произойдет, когда продавец попытается подобрать покупателю синий Porsche, о котором тот так мечтал, выполнив на подчиненном сервере следующий запрос:

```
SELECT * FROM auto.used_cars
WHERE make = 'Porsche' AND model = 'Cayman' AND color = 'blue';
```

Обнаружит ли продавец, выполнивший этот запрос, синий Porsche Cayman, который он сможет продать? Хороший продавец всегда визуально проверит, есть ли у него на стоянке такая машина, но предположим, что он слишком занят для этого и поэтому сообщает, что она имеется. Каково будет его смущение (не говоря уже о несостоявшейся продаже), когда покупатель придет на тест-драйв лишь для того, чтобы обнаружить, что автомобиль-то белый!

Чтобы предотвратить потерю данных при сбое на главном сервере, включите параметр sync_binlog (со значением 1) при запуске или в файле конфигурации. Так главный сервер будет сбрасывать события в двоичный журнал незамедлительно. Хотя это может привести к значительному снижению производительности при работе с InnoDB, это очень хорошая страховка для случаев, когда потеря изменений непозволительна (и тем не менее, в зависимости от момента сбоя, последнее событие может быть потеряно).

Если приведенная ситуация не кажется особо критичной, представьте потерю обновления в медицинской базе данных или в базе данных с результатами научных исследований. Очевидно, что потеря вроде бы пустячного обновления может привести к большим проблемам. В общем-то, приведенный пример можно рассматривать как частный случай повреждения данных. Сталкиваясь с такой проблемой, всегда проверяйте содержимое таблиц. В таком случае восстановление после сбоя обеспечивает согласованность двоичного журнала и InnoDB (если установлен параметр sync binlog=1), в противном случае это никак не влияет на таблицы MyISAM.

Запрос проходит на главном сервере, но не на подчиненном

Строго говоря, это не является проблемой главного сервера, но иногда запрос (например, команда обновления или вставки) нормально работает на главном сервере и не работает на подчиненном. У ошибки такого рода может быть множество причин, но в основном это проблемы целостности ссылочных данных, конфигурации подчиненного сервера или базы данных.

Наиболее распространенная причина возникновения этой ошибки заключается в том, что запрос ссылается на таблицу, которая не существует на подчиненном сервере или имеет другую сигнатуру (другие столбцы или типы столбцов). В таком случае необходимо изменить подчиненную систему так, чтобы она соответствовала главной, и запрос можно было выполнить.

В некоторых случаях запрос может ссылаться на таблицу, которая не была реплицирована. Например, если используются какие-либо загрузочные параметры фильтрации репликации (это можно выяснить быстрой проверкой состояния главного и подчиненного серверов), то возможно, что базы данных, на которую ссылается запрос, нет на подчиненном сервере. В таком случае необходимо либо соответствующим образом настроить фильтры, либо вручную добавить недостающие таблицы и (или) базы данных на подчиненном сервере.

В других случаях причины неудачного выполнения запроса могут быть более сложными: проблемы с набором символов, поврежденные таблицы или даже поврежденные данные. Если проверка подтвердила, что подчиненный сервер настроен так же, как главный, может понадобиться ручная отладка запроса. Если не удается решить проблему на подчиненном сервере, можно попробовать выполнить обновление вручную и пропустить событие, содержащее сбойный запрос.

Чтобы пропустить событие на подчиненном сервере, используйте переменную sql_slave_skip_counter и укажите число событий главного сервера, которое требуется пропустить. Иногда это самый быстрый способ перезапуска репликации.

Повреждение таблицы после сбоя

Если на главном или подчиненном сервере происходит сбой, и после перезапуска обоих серверов оказывается, что одна или несколько таблиц повреждены или помечены MyISAM как сбойные, необходимо исправить эти проблемы перед перезапуском репликации.
Чтобы выяснить, какие таблицы повреждены, можно проверить файлы журналов сервера на наличие таких ошибок:

```
... [ERROR] /usr/bin/mysqld: Table 'db1.t1' is marked as
crashed and should be repaired ...
```

Следующую команду можно использовать для оптимизации и восстановления в один прием. Чтобы восстановить все таблицы заданной базы данных (в данном случае базы данных *db1*) выполните

```
mysqlcheck -u <user> -p --check --optimize --auto-repair db1
```

Для таблиц MyISAM можно включить автоматическое восстановление, используя параметр myisam-recover. Существует четыре режима восстановления. Подробнее см. в онлайновом руководстве MySQL Reference Manual.

После восстановления поврежденных таблиц следует выяснить, были ли повреждены таблицы на подчиненном сервере. Это необходимо, если главный и подчиненный серверы совместно используют один центр обработки данных, а причина ошибки была внешней (например, они были подключены к одному источнику питания).

Всегда выполняйте резервное копирование таблиц перед попыткой их восстановления. В некоторых случаях операция восстановления может привести к потере данных или оставить таблицу в неизвестном состоянии.

Кроме того, восстановление может привести к рассинхронизации главного и подчиненного серверов, особенно если восстановление привело к потере данных. В таком случае следует сравнить данные в пострадавших таблицах, чтобы убедиться, что серверы синхронизированы. Если данные отсутствуют на подчиненном сервере, можно повторно загрузить данные с главного сервера, а если данные пропали на главном сервере, их можно скопировать с подчиненного.

Повреждение двоичного журнала на главном сервере

Если сбой сервера или выход диска из строя приводит к повреждению двоичного журнала на главном сервере, вы не сможете перезапустить репликацию. Причин и типов повреждения двоичного журнала может быть много, но все они приводят к невозможности выполнения одного или нескольких событий на подчиненном сервере, часто приводя к ошибкам типа «невозможен анализ журнала событий ретрансляции».

В таком случае необходимо внимательно проверить двоичный журнал, найти в нем восстановимые события и поменять журналы на главном сервере командой FLUSH LOGS. В результате на подчиненном сервере может

произойти потеря данных, и он, скорее всего, выйдет из строя в этом случае. Лучший способ восстановления — повторно синхронизировать подчиненный сервер с главным, используя надежную резервную копию и утилиту восстановления. В дополнение замене журналов, можно минимизировать потерю данных и получить возможность перезапустить репликацию без ошибок.

В некоторых случаях, если легко определить, сколько событий было повреждено или отсутствует, возможно пропустить поврежденные события, используя на подчиненном сервере переменную sql_slave_skip_counter. Число событий можно определить, сравнив ссылку двоичного журнала главного сервера на подчиненном сервере с текущей позицией двоичного журнала на главном сервере.

Прерывание длительных запросов к таблицам, не использующим транзакции

Принудительно завершая запрос, изменяющий таблицу, не использующую транзакции, учтите, что этот запрос мог быть реплицирован и выполнен на подчиненном сервере. Если это произошло, изменения на главном и подчиненном серверах могут различаться.

Например, если прервать запрос, обновляющий 400 из 600 строк таблицы, когда он успел обновить только 200 строк, то возможна ситуация, в которой на подчиненном сервере были обновлены все 400 строк.

Таким образом, каждый раз, прерывая запрос, обновляющий данные на главном сервере, необходимо убедиться, что это обновление не выполнено на подчиненном сервере. Если оно выполнено (или просто на всякий случай), выполните синхронизацию данных на подчиненном сервере после исправления таблицы на главном сервере. Обычно для этого нужно создать на главном сервере резервную копию исправленных данных и восстановить ее на подчиненном сервере.

Проблемы на подчиненном сервере

Как сказано ранее, большинство сбоев возникает в результате какой-либо ошибки на подчиненном сервере. Иногда, конечно, источником проблем является главный сервер, но почти всегда виноватым оказывается все же подчиненный сервер. Ниже описаны некоторые из наиболее распространенных проблем с ним.

Использование двоичных журналов на подчиненном сервере

Одним из способов повышения надежности подчиненных серверов является включение ведения двоичных журналов при помощи параметра log-slave-updates. В результате этого подчиненный сервер будет записывать в журнал события, которые выполняет из журнала ретрансляции. Так создается двоичный журнал, который можно использовать для повторного выполнения событий в случае повреждения журнала ретрансляции (или данных).

Репликация не запустилась из-за сбоя на подчиненном сервере

После сбоя на подчиненном сервере обычно легко удается возобновить репликацию, определив последнее удачное событие на этом сервере. Выявить это событие можно, проверив выходные данные команды SHOW SLAVE STATUS.

Если же возникли ошибки из-за контроля доступа и учетной записи, возможно, что репликацию перезапустить не удастся. Причиной могут быть проблемы с аутентификацией (например, была удалена учетная запись репликации подчиненного сервера) или поврежденные таблицы на главном или подчиненных серверах. В таких случаях часто можно увидеть сообщения об ошибках подключения в консоли и журналах на подчиненном сервере MySQL.

Когда такое происходит, всегда проверяйте разрешения пользователя репликации на главном сервере. Убедитесь, что пользователю, определенному в файле конфигурации или в параметрах команды CHANGE MASTER, предоставлены необходимые привилегии:

```
GRANT REPLICATION SLAVE ON *.*
TO 'имя_пользователя'@'%' IDENTIFIED BY 'пароль';
```

Можете изменить эту команду в соответствии с собственными потребностями и использовать для решения такой проблемы.

Тайм-аут подключения подчиненного сервера и частое переподключение

Если топология включает несколько подчиненных серверов, и при этом значение server_id не установлено дублируется, вероятен конфликт идентификаторов серверов. При этом на одном из подчиненных серверов могут часто возникать тайм-ауты или частые разрывы подключения.

Такая проблема возникает просто из-за дублирования ID подчиненных серверов, и ее бывает сложно диагностировать (точнее, ее легко спутать со сбоями из-за проблем с подключением). Всегда следует проверять журнал ошибок на главном и подчиненном серверах и просматривать сообщения об ошибках. Скорее всего, сообщение об ошибке будет объяснять причину возникновения тайм-аута.

Чтобы предотвратить возникновение таких проблем, всегда устанавливайте значение server_id в файле конфигурации или в командной строке при запуске.

Результаты запроса на главном и подчиненном серверах различаются

Одна из наиболее сложных для диагностики проблем возникает, когда результаты запроса, выполненного на одном или нескольких подчиненных

серверах, отличаются от результатов на главном сервере. Эта проблема может быть легко разрешимой и безобидной, если просто различается порядок сортировки, а может оказаться и серьезной, когда в полученных данных отсутствуют некоторые строки или присутствуют лишние.

Основной причиной проблем такого рода являются различия в наборах символов на главном и подчиненных серверах. Например, на главном сервере может быть выбран один набор символов (character set) и правила сортировки (collation), а на одном или нескольких подчиненных — другие.

Если пользователи начинают жаловаться на отсутствующие или лишние строки или на различия в порядке результатов запросов, в первую очередь следует проверить параметры наборов символов на главном и подчиненных серверах.

Другой возможной причиной возникновения такой проблемы может быть использование разных механизмов СУБД на главном и подчиненном серверах (например, MyISAM на главном сервере и InnoDB на подчиненном). В таком случае порядок результатов будет различаться, если в команде ALTER TABLE указан механизм СУБД, использующий правило сортировки, отличное от того, что используется на главном сервере.

Еще более хитрой причиной проблем такого рода является различие определений таблиц на главном и подчиненном серверах. Такие различия вероятны, когда подмножество столбцов определенной таблицы совпадает, а некоторые столбцы в начале или в конце (тут важен порядок) отсутствуют на подчиненном сервере.

При этом вероятно множество проблем. Пользователь может ожидать, что данные из некоторых столбцов будут реплицированы, но на подчиненном сервере нет соответствующих столбцов. Иногда желательно иметь на подчиненном сервере меньшее количество столбцов. Невнимательный пользователь может случайно удалить столбцы, а репликация все равно пройдет. В некоторых случаях выполнение на подчиненном сервере запроса SELECT к отсутствующим столбцам вызовет ошибку, проясняющую суть проблемы. В других случаях в приложениях просто будут отсутствовать данные.

Распространенная пользовательская ошибка, приводящая к различиям в результатах запросов на главном и подчиненном серверах, — внесение изменений в таблицы и базы данных на подчиненном сервере, но не на главном. То есть, пользователь выполняет некие нереплицируемые манипуляции с данными на подчиненном сервере, которые изменяют сигнатуру таблицы, но не делает того же на главном сервере. В таких случаях запросы могут возвращать неверные результаты, неверные столбцы, неверный порядок или лишние данные, либо просто завершаться с ошибкой, ссылаясь на отсутствующие столбцы. Рекомендуем всегда проверять структуру таблиц, при работе с которыми возникают подобные проблемы, чтобы убедиться, что на главном и подчиненном серверах она совпадает. Если нет, выполните синхронизацию и повторите запрос.

Ошибки на подчиненном сервере при попытке подключения через SSL

Проблемы, относящиеся к подключениям SSL, обычно возникают из-за проблем с разрешениями, описанными выше. В этом случае предоставленные привилегии должны также включать параметр REQUIRE SSL, как показано ниже. Проверьте, существует ли пользователь репликации и имеет ли необходимые привилегии.

```
GRANT REPLICATION SLAVE ON *.*
TO 'rpl_user'@'%' IDENTIFIED BY 'password_here' REQUIRE SSL;
```

Из других проблем, относящихся к перезапуску репликации с использованием подключений SSL, можно отметить отсутствие файлов сертификатов и некорректные значения параметров SSL в файле конфигурации (например, ssl-ca, ssl-cert и ssl-key) или в команде CHANGE MASTER (например, MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT и MASTER_SSL_KEY). Проверьте параметры и пути, чтобы убедиться, что ничего не изменилось с момента последнего запуска репликации.

Хранимая в памяти таблица пуста

Если одна или несколько баз данных используют устройство хранимые в памяти таблицы, их содержимое будет потеряны при перезагрузке подчиненного сервера (но не при перезапуске потоков на этом сервере). Это ожидаемое событие, так как данные в таких таблицах не сохраняются при перезагрузке. Ее конфигурация сохранится, к таблице можно будет обращаться, но все данные из нее исчезнут.

Возможно, что при перезапуске подчиненного сервера запросы к хранимой в памяти таблице будут завершаться с ошибкой (например, запросы UPDATE) или возвращать неточные результаты (например, запросы SELECT). Таким образом, ошибка может обнаруживаться не сразу и заключаться просто в отсутствии строк в результатах.

Чтобы избежать этого, тщательно продумайте использование в БД хранимых в памяти таблиц. Не следует создавать на главном сервере хранимые в памяти таблицы, которые должны обновляться на подчиненных серверах посредством репликации, не предусмотрев процедуры для восстановления данных этих таблиц после сбоя или плановой перезагрузки сервера. Например, перед началом репликации можно выполнить сценарий, копирующий с главного сервера данные для таблицы. Если данные получены, используйте сценарий для записи данных на подчиненный сервер.

Также следует обдумать возможность фильтрации таблицы во время репликации и отказ от хранения в памяти для всех реплицируемых таблиц.

Временные таблицы отсутствуют после сбоя на подчиненном сервере

Если реплицируемые базы данных и запросы используют временные таблицы, следует учитывать некоторые важные особенности временных таблиц. Когда подчиненный сервер перезагружается, его временные таблицы теряются. Если какие-то временные таблицы были реплицированы с главного сервера, а вы после этого не можете перезагрузить подчиненный сервер, может понадобиться вручную создать эти таблицы или пропустить запросы, обращающиеся к временным запросам.

Такая ситуация часто приводит к тому, что запрос не выполняется на одном или нескольких подчиненных серверах. Решение проблемы такое же, как у проблемы с пропажей таблиц, хранимых в памяти. В частности, чтобы выполнить запрос, может потребоваться вручную воссоздать временную таблицу или синхронизировать данные на подчиненном сервере с данными на главном сервере и пропустить запрос при перезапуске подчиненного сервера.

Подчиненный сервер работает медленно и не синхронизирован с главным

Если подчиненный сервер работает слишком медленно, он не успевает обработать все события с главного сервера, что приводит к задержкам в обновлении данных. В наиболее серьезных случаях обновление на подчиненном сервере приводит к возникновению устаревших данных и получению некорректных результатов. Например, если подчиненный сервер в агентстве по продаже билетов намного отстает от главного сервера, агентство может начать продавать билеты на уже занятые места (эти места помечены на главном сервере как проданные, но подчиненный сервер вовремя не внес эту информацию в свою копию БД).

Мы обсуждали эту проблему в предыдущих главах, так что здесь только вкратце опишем решение. Чтобы выявить проблему, изучите выходные данные команды SLAVE STATUS OUTPUT на подчиненном сервере и проверьте столбец Seconds_Behind_Master, чтобы убедиться, что его значение допустимо для вашего приложения. Чтобы решить проблему, рассмотрите возможности перемещения некоторых баз данных на другие подчиненные серверы, уменьшения числа баз данных, реплицируемых на проблемный сервер, устранения задержек в работе сети (если таковые имеются) и усовершенствования хранилищ данных.

Например, можно разгрузить подчиненный сервер, используя другой сервер для массовых или сложных обновлений. Можно снизить нагрузку, связанную с репликацией, выполняя обновления на отдельном подчиненном сервере и применяя изменения к остальным системам в топологии посредством восстановления надежной резервной копии.

Потеря данных после сбоя на подчиненном сервере

Бывает, что подчиненный сервер вышел из строя и не записал последнюю известную позицию в двоичном журнале главного сервера. Эта информация сохраняется в файле *relay_log.info*. Когда такое происходит, подчиненный сервер пробует возобновить работу с неверной (старой) записи в журнале и выполнить запросы, которые уже были выполнены. Обычно это приводит к ошибкам запросов. С этим можно разобраться, пропустив дублированные события.

Однако дублирование событий может привести к повреждению данных и рассинхронизации подчиненного сервера с главным. К сожалению, проблемы такого рода сложно обнаружить. Некоторые выполненные события можно отследить при внимательной проверке файлов журналов, но для того, чтобы определить, какие из них были дублированы, может потребоваться изучение событий из двоичного журнала как на подчиненном, так и на главном сервере.

Повреждение таблицы после сбоя

При перезагрузке главного сервера после сбоя можно обнаружить, что одна или несколько таблиц повреждены или помечены MyISAM как сбойные. Прежде чем перезапускать репликацию, необходимо решить эти проблемы. После восстановления пострадавших таблиц убедитесь, что таблицы на подчиненном сервере не были повреждены в результате восстановления. Это маловероятно, но иногда случается. Если сомневаетесь, всегда вручную синхронизируйте эти таблицы с главным сервером, используя резервную копию или другую аналогичную процедуру, прежде чем запускать репликацию.

Потеря данных в результате восстановления — очень вероятная ситуация для MyISAM, когда сбой оборудования или сервера происходит во время частичной записи страницы. К сожалению, не всегда легко определить, были ли потеряны какие-то данные.

Повреждение журнала ретрансляции на подчиненном сервере

Если в результате сбоя или отказа диска повреждается журнал ретрансляции на подчиненном сервере, репликация остановится с сообщением об ошибках журнала ретрансляции. Существует много причин и типов повреждений журнала ретрансляций, но все они приводят к невозможности выполнить одно или несколько событий на подчиненном сервере.

Когда такое происходит, лучший вариант восстановления — определить, где было выполнено последнее успешное событие из двоичного журнала главного сервера, и перезапустить репликацию с использованием команды CHANGE MASTER, указав информацию из двоичного журнала главного сервера. К сожалению, это значит, что восстановление из старого журнала ретрансляции может привести к ошибкам

Ошибки во время перезапуска подчиненного сервера

Одна из более сложных для обнаружения и исправления проблем — возникновение на подчиненном сервере ошибок во время загрузки или перезапуска. Возникать могут различные ошибки, и иногда это происходит без явной причины.

Если такое произошло, проверьте размер max_allowed_packet на главном и подчиненном серверах. Если на главном сервере размер больше, чем на подчиненном, возможно, что главный сервер зарегистрировал событие, которое превысило допустимый на подчиненном сервере размер. Это может привести к случайным и на первый взгляд невоспроизводимым ошибкам.

Неудачные транзакции на подчиненном сервере

Обычно, когда не удается выполнить транзакцию, изменения откатываются, чтобы избежать проблем, связанных с частичным обновлением. Однако эта задача усложняется, когда совместно используются таблицы, защищенные и не защищенные транзакциями: во-первых, изменения откатываются, а во-вторых — нет. Это может приводить к таким проблемам, как потеря или дублирование данных, избыточные или нежелательные изменения незащищенных таблиц.

Лучший способ избежать возникновения таких проблем — не смешивать в базе данных сущности, использующие и не использующие транзакции, либо всегда использовать защиту транзакциями.

Более сложные проблемы репликации

В некоторых сложных топологиях естественным образом возникают определенные затруднения. В этом разделе мы расскажем о проблемах, часто встречающихся при использовании репликации в таких топологиях.

Изменение не реплицируется в топологии

В некоторых случаях изменения в объекте базы данных не реплицируются. Например, команда ALTER TABLE может реплицироваться, а FLUSH, REPAIR TABLE и похожие команды обслуживания — нет. Когда такое происходит, проверьте ограничения команд манипуляции данными (DML) и сервисных команд.

Часто такие проблемы возникают из-за того, что неопытный администратор или разработчик пытается выполнить администрировать базы данных на главном сервере и ожидает, что изменения будут реплицированы на подчиненные серверы.

Когда в объект базы данных вносятся изменения, затрагивающие его структуру на файловом уровне, или выполняются команды обслуживания, следует выполнить те же команды и процедуры на всех подчиненных серверах, чтобы изменения распространились по топологии.

Сообразительные администраторы часто выполняют такие задачи во время запланированного обслуживания, используя сценарии. Обычно сценарии должным образом останавливают репликацию, применяют изменения и возобновляют репликацию автоматически.

Проблемы с круговой репликацией

Если вы используете круговую репликацию и выполнили восстановление после сбоя репликации, в результате чего один или несколько серверов выпали из топологии, вероятна проблема из-за многократного повтора события на некоторых серверах. Это может привести к сбою репликации, если запрос завершается неудачно (например, из-за нарушения уникальности ключа). Возможная причина — удаление исходного сервера из топологии.

Когда такое случается, сервер, назначенный исходным, не может завершить репликацию события. Эту проблему можно решить, используя параметр IGNORE_SERVER_IDS (доступен в MySQL 5.5.2 и выше) в команде CHANGE MASTER, указав список ID серверов, для которых нужно игнорировать событие. После восстановления отсутствующих серверов необходимо изменить этот параметр, чтобы события с восстановленных серверов не игнорировались.

Проблемы с несколькими главными серверами

Как и в случае с круговой репликацией (которая является частной формой топологии с несколькими главными серверами), при восстановлении после сбоя репликации некоторые события могут быть выполнены более одного раза. Такие события обычно поступают с выпавшего сервера. Эта проблема решается так же, как и для круговой репликации — помещением ID удаленных серверов в список параметра IGNORE_SERVER_IDS команды CHANGE MASTER.

Другая проблема с репликацией при нескольких главных серверах может возникать, когда изменения в одной таблице появляются на двух главных серверах, и эта таблица имеет столбец с автоматически увеличивающимися значениями первичного ключа. В этом случае возникнет ошибка дублирования ключа. Если необходимо вставить новые строки более чем на одном главном сервере, используйте параметры auto_increment_increment и auto_increment_offset, чтобы отрегулировать приращение ключа. Например, один сервер может автоматически увеличивать только четные числа, а другой только нечетные. Это позволяет решить проблему с двумя серверами, но когда серверов больше, могут возникнуть сложности с обновлением одной таблицы с автоматическим увеличением первичного ключа. В этом случае не только гораздо сложнее отрегулировать приращение ключа, но и возникают административные проблемы, когда требуются заменить в топологии сервер, обновляющий таблицу. Например, в значениях ключа могут появиться пропуски, что в конечном итоге может привести к превышению

максимальных значений, определенных типом данных ключевого поля, в большой таблице.

Ошибка HA_ERR_KEY_NOT_FOUND

Эта знакомая ошибка встречается в топологии построчной репликации. Наиболее вероятная ее причина — конфликт, возникающий, когда строка, которую нужно обновить или удалить, отсутствует или изменена, из-за чего механизм СУБД не может ее найти. Это может являться результатом ошибки во время круговой репликации или изменений, внесенных в реплицируемые данные непосредственно на подчиненном сервере. Когда такая ошибка возникает, необходимо установить причину конфликта и восстановить данные или пропустить проблемное событие.

Средства решения проблем репликации

Если у вас есть опыт настройки и обслуживания репликации, большинство инструментов для диагностики и устранения проблем репликации будут вам знакомы.

В этом разделе мы обсудим средства диагностики проблемы репликации и дадим советы о том, как и когда их применять.

- SHOW MASTER STATUS и SHOW SLAVE STATUS При диагностировании проблем репликации эти команды SQL следует использовать в первую очередь. Как и SHOW PROCESSLIST, эти команды следует выполнять сначала на главном сервере, а затем на подчиненном, после чего проверить выходные данные. Команда для подчиненного сервера имеет расширенный набор параметров, незаменимый для диагностирования проблем репликации.
- SHOW GRANTS FOR *<имя_пользователя>* Когда возникают проблемы с доступом пользователя подчиненного сервера, в первую очередь следует проверять разрешения, которые имеет этот пользователь, чтобы убедиться, что они не изменялись.
- CHANGE MASTER Иногда файлы конфигураций изменяются, намеренно или случайно. Используйте эту команду SQL для перезаписи параметров последнего известного подключения и для диагностирования проблем с подключением подчиненного сервера.
- STOP/START SLAVE Используйте эти команды SQL для запуска и остановки репликации. Иногда требуется остановить подчиненный сервер, когда он находится в состоянии ошибки.

Проверка файлов конфигурации

Иногда проблема возникает в результате несанкционированного или забытого изменения конфигурации. Возьмите в привычку проверять файлы конфигурации при диагностировании проблем с подключением.

Проверка журналов серверов

Всегда проверяйте журналы серверов, когда диагностируете неполадки. Иногда это позволяет выявить проблемы, никак себя не проявляющие. Сообщения об ошибках и предупреждения иногда подобны китайской грамоте, но могут и оказаться очень полезными.

- SHOW SLAVE HOSTS Используйте эту команду на главном сервере для идентификации подключенных к нему подчиненных серверов, если они используют параметр report-host.
- SHOW PROCESSLIST Когда возникают проблемы, полезно выяснить, что запущено. Эта команда покажет текущее состояние каждого потока, участвующего в репликации. При поиске решения проблемы проверяйте это в первую очередь.
- SHOW BINLOG EVENTS Эта команда SQL показывает события из двоичного журнала. Если используется репликация на основе команд, эта команда будет отображать изменения в виде команд SQL.
- mysqlbinlog Эта утилита позволяет читать события из двоичного журнала и журнала ретрансляции. Нередко она показывает, какие события повреждены. Ее можно часто использовать при диагностировании проблем, связанных с событиями и двоичным журналом.
- PURGE BINARY LOGS Эта команда SQL позволяет удалять определенные события из двоичного журнала, например, события, произошедшие после указанного момента времени или после события с заданным ID. В регулярные меры по обслуживанию следует включить применение этой команды для очистки старых и более не нужных двоичных журналов.

Итак, мы рассмотрели сбои, которые могут возникнуть при репликации, и инструменты для их устранения, доступные в типичном выпуске MySQL. Теперь перейдем к описанию стратегий решения проблем репликации.

Рекомендации

Обзор потенциальных проблем и инструментов для их исправления — только часть полного решения. Существуют проверенные стратегии и рекомендации для быстрого решения проблем репликации.

В этом разделе описываются стратегии и рекомендации, которые следует применять при диагностировании и решении проблем репликации. Не следует догматически следовать порядку изложения, выбранного авторами: в зависимости от сути проблемы, полезными могут оказаться разные советы.

Изучите топологию

Тому, кто использует репликацию MySQL на небольшом количестве серверов, нетрудно запомнить конфигурацию топологии. Это может быть один главный и один или несколько подчиненных серверов или два сервера в топологии с несколькими главными серверами. Однако при усложнении топо-

логии наступает момент, когда уже невозможно запомнить структуру и все параметры конфигурации топологии.

Чем сложнее топология и конфигурация, тем сложнее определить причину проблемы и отправную точку для операций по восстановлению. Легко забыть про какой-нибудь подчиненный сервер, когда в топологии их сотни.

Хорошей идеей будет составить карту топологии и текущих параметров конфигурации. Рекомендуем вести записи о параметрах репликации в записной книжке или файле, размещенном там, где коллеги и подчиненные легко смогут его найти. Такая информация будет незаменима для того, кто разбирается в администрировании репликации в ваше отсутствие.

В эти записи следует включить текстовое описание или графическую схему топологии и указать все фильтры (на главных и подчиненных серверах) и роли всех серверов в топологии. Туда же можно добавить описание команды CHANGE MASTER со всеми параметрами и содержимое файлов конфигурации всех серверов.

Не пытайтесь создать произведение искусства, достаточно простого чертежа. На рис. 11-1 показана гибридная топология с пояснениями о фильтрах и ролях.

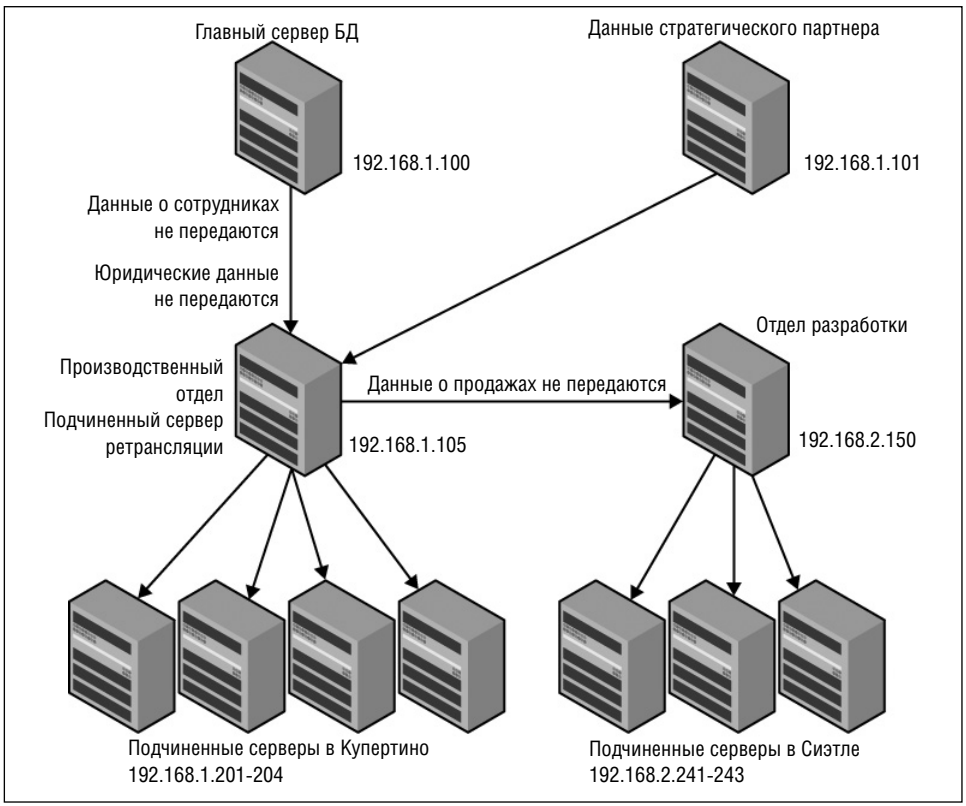


Рис. 11-1. Пример карты топологии

Обратите внимание на то, что подчиненный сервер производственного отдела (192.168.1.105) имеет два главных сервера (192.168.1.100 и 192.168.1.101). Это странно, так как подчиненный сервер не может иметь более одного главного сервера. Чтобы получить такой уровень интеграции и использовать данные, предоставляемые посторонним сервером, потребуется установить на сервере производственного отдела второй экземпляр сервера MySQL, реплицирующий данные с сервера стратегического партнера (192.168.1.101) и использующий сценарий для периодической передачи данных первому экземпляру. Это обеспечит уровень интеграции, показанный на рис. 11-1, ценой роста объема ручного труда и задержкой обновления данными от стратегического партнера.

Некоторым топологиям присущи определенные проблемы. О многих из них мы уже рассказывали в предыдущих главах. Ниже приведен список типов топологии с кратким описанием некоторых распространенных проблем и уязвимостей, о которых следует знать:

- *Звезда (топология с одним главным сервером)* Это типичная топология с одним главным и многими подчиненными серверами. Здесь нет особых ограничений и проблем, кроме тех, которые связаны с репликацией в целом. Однако при повышении подчиненного сервера до главного могут возникнуть затруднения с тем, что необходимо использовать подчиненный сервер, наиболее синхронизированный с главным. Бывает, что такой сервер сложно определить, т.к. для этого потребуется проверка состояния каждого подчиненного сервера.
- *Цепь* В этой конфигурации сервер является главным для одного подчиненного, который, в свою очередь, является главным для другого подчиненного и т.д. Последний сервер является только подчиненным. Кроме упомянутой ранее проблемы с ID серверов, может возникнуть проблема с определением позиции, в которой находился промежуточный главный или подчиненный сервер перед сбоем. Повышение нового главного или подчиненного сервера может потребовать дополнительной работы по обеспечению согласованности данных.
- *Круговая (кольцо)* То же, что и цепь, с той разницей, что нет конечного сервера. Эта топология требует внимательной настройки, чтобы репликация завершалась на исходном сервере.
- *Много главных серверов* Аналогична круговой топологии, но каждый главный сервер является подчиненным каждого другого главного сервера. Эта топология имеет все ограничения и проблемы круговой репликации, а также является наиболее сложной в управлении из-за возможности конфликтов изменений. Чтобы избежать этой проблемы, необходимо убедиться, что изменения выполняются только на одном из главных серверов. Если, например, на одном сервере выполняется запрос UPDATE, а на другом DELETE, возможно, что запрос DELETE будет выполнен перед запросом UPDATE, что приведет к неудачному завершению запроса UPDATE.

- *Гибридная* Гибридная топология использует элементы некоторых или всех других топологий. Обычно такая топология применяется в крупных организациях, разделенных на отделы с похожими, но часто изолированными элементами инфраструктуры. Изоляция обеспечивается фильтрацией, при которой данные разделяются и передаются нескольким подчиненным серверам с исходного главного сервера (иногда его называют *первичным* или *самым главным*). Этот сервер содержит все данные, отправляемые подчиненным серверам, которые становятся главными в собственных топологиях-звездах. Такая топология является самой сложной и требует тщательного документирования.

Проверяйте состояние всех серверов

Одна из наиболее действенных предупредительных мер — регулярная проверка состояния всех серверов в топологии. Это не сложно: например, можно настроить запланированную задачу, которая будет запускать клиент MySQL, выполнять команду SHOW MASTER STATUS или SHOW SLAVE STATUS и распечатывать результаты или отправлять их по электронной почте.

Конечно, эти сообщения придется читать! Можно настроить эту запланированную задачу так, чтобы она выполнялась в то время, когда у вас есть время изучить результаты. Например, эта задача может выполняться перед обедом, чтобы вы могли прочитать сообщения во время обеда или сразу после него.

Постоянное слежение за состоянием сервера — хороший способ оставаться в курсе дела, когда имеется потенциальная проблема. Кроме того, это позволяет быстрее реагировать на появление ошибок.

Следует искать ошибки, следить за задержками в работе подчиненных серверов, проверять фильтры, чтобы они соответствовали записям, и проверять, все ли подчиненные серверы работают нормально, без предупреждений и сообщений об ошибках подключения.

Проверяйте журналы

Кроме регулярной проверки состояния серверов, следует также выделить время для периодического просмотра журналов серверов. Это легко делать, подключаясь к каждому серверу по очереди при помощи графического пользовательского интерфейса MySQL Administrator и просматривая последние записи в журналах.

Если старательно выполнять эту работу, результатом будет раннее обнаружение возникших проблем. Иногда в журнал записываются ошибки и предупреждения, никак не проявляющие себя в других местах. Если сразу заметить их, работа по восстановлению будет гораздо проще.

Рекомендуем проверять журналы в то же время, когда вы проверяете состояние серверов.

Проверяйте конфигурацию

Вместе с состояниями и файлами журналов следует также периодически проверять файлы конфигурации, чтобы документация всегда содержала актуальную информацию. Это не так критично, как проверка журналов, но часто о такой проверке забывают. Рекомендуем проверять файлы конфигурации и обновлять документацию хотя бы раз в неделю, если вносится много изменений, или раз в месяц, если топология и серверы изменяются не часто. Если администраторов несколько, возможно, это лучше делать чаще.

Выполняйте безопасную остановку

Иногда требуется остановить репликацию на время диагностирования и решения проблемы. Если репликация уже остановлена, может не понадобиться ничего делать, но если работа ведется в сложной топологии, а ошибка связана с потерей данных, безопаснее будет остановить репликацию во всей топологии. Но сделать это контролируемым и безопасным способом.

Существует несколько стратегий выполнения контролируемой остановки топологии репликации. Если проблема связана с потерей данных, а задержки в работе подчиненного сервера нет, можно заблокировать таблицы на главном сервере и очистить двоичные журналы, затем подождать отклика от всех оставшихся подчиненных серверов, после чего остановить их. Это гарантирует репликацию и выполнение всех событий на работающих подчиненных серверах.

С другой стороны, если проблема серьезная, можно начать с оставшихся в топологии подчиненных серверов и идти вверх по топологии, оставляя главный сервер работающим. Однако если главный сервер продолжает работу (т.е. не все таблицы заблокированы), выполняется много обновлений, а диагностика и восстановление требует длительного времени, подчиненные серверы будут сильно отставать от главного после возобновления репликации. Если восстановление может занять много времени, лучше остановить обновления на главном сервере.

Если вы столкнулись со сложной проблемой или (что еще хуже) проблема возникла без видимых причин и предупреждений, может понадобиться полная остановка репликации. Это особенно актуально, если проблема возникла только на одном подчиненном сервере. Остановка репликации позволит изолировать сервер на время диагностики.

После сбоя выполняйте перезапуск должным образом

Кроме всего прочего, важно правильно возобновлять репликацию. Часто лучшим вариантом будет запуск топологии репликации в контролируемых условиях, например, на одном главном сервере и одном подчиненном. Даже если топология более сложная, сначала лучше запустить только необходи-

мые компоненты репликации, чтобы провести тестирование и убедиться, что проблема решена, прежде чем запускать все серверы.

Изоляция важна при решении проблем, вовлекающих одиночный сервер или набор событий. Начинать работу с одним главным и одним подчиненным сервером, полезно также потому, что в этом случае, если вы не сможете исправить проблему, вам будет проще получить помощь от специалистов по MySQL, так как проблема изолирована до минимально возможного набора параметров. Подробнее см. в разделе «Отчеты об ошибках репликации» в конце этой главы.

Выполняйте неудачные запросы вручную

Одной из стратегий, о которых часто забывают, является изучение запросов в журналах ретрансляций и двоичных журналах для поиска причин проблем. Легко застрять в поиске ошибок на подчиненном сервере и проверке всех возможных причин неполадок, но иногда (особенно, когда дело касается запроса) можно получить гораздо больше информации о ситуации, изолировав проблемный сервер и попробовав выполнить запрос вручную.

Если используется репликация на основе команд, эта задача будет легкой, так как запрос будет представлен в читаемом виде в двоичном журнале или журнале ретрансляций. Если используется репликация на основе строк, запрос все равно можно выполнить, хотя и не удастся прочитать. В этом случае неправильно сформированный запрос или запрос, использующий отсутствующие, некорректные или поврежденные ссылки, не так легко обнаружить, если не попытаться выполнить вручную.

Помните, что всегда следует создавать резервную копию данных, прежде чем пытаться выполнять операции по диагностике, способные изменить данные. Выполнение неправильного запроса — один из таких случаев. Мы встречали запросы, вызывающие ошибки репликации, но иногда успешно выполняемые вручную. Когда такое происходит, это указывает скорее на ошибку в конфигурации подчиненного сервера или проблеме с двоичным журналом или журналом ретрансляций, а не на проблему с самим запросом.

Общие процедуры

Существуют некоторые общие процедуры, которые можно использовать для применения приведенных выше рекомендаций. Сюда относится поиск проблем репликации и приостановка репликации на главном сервере.

Устранение неполадок при репликации

Ниже приведена пошаговая процедура, которую можно использовать при решении проблем репликации. Большинство шагов описано в других разделах книги и может быть вам знакомо.

Рекомендуем записывать все наблюдения, полученные при выполнении этой процедуры. Когда вся информация собрана в одном месте, можно получить более четкое представление о ситуации.

1. Проверьте состояние главного сервера. Запишите любые аномалии.
2. Проверьте состояние каждого подчиненного сервера. Запишите любые аномалии.
3. Просмотрите журналы ошибок на всех серверах.
4. Проверьте список процессов на каждом сервере. Ищите аномалии состояний.
5. Если сообщений об ошибках нет, попробуйте перезапустить подчиненный сервер. Запишите любые возникшие ошибки.
6. Проверьте конфигурацию проблемного сервера, чтобы убедиться, что ничего не изменилось.
7. Просмотрите свои заметки и продумайте план по диагностированию и исправлению ошибки.

Эта процедура слишком упрощена, но позволяет диагностировать проблему быстрее, чем бессистемной проверке сообщений об ошибках (если они вообще есть).

Приостановка репликации

Чтобы приостановить репликацию, выполните следующие действия на каждом из главных серверов, начиная с первичного (первого главного).

На главном сервере:

1. Выполните команду FLUSH TABLES WITH READ LOCK.
2. Выполните команду SHOW MASTER STATUS.
3. Запишите двоичный журнал главного сервера и позицию.

На подчиненных серверах:

4. Выполните команду SELECT MASTER_POS_WAIT (имя_двоичного_журнала позиция_в_журнале).

На каждом из оставшихся активных подчиненных серверов, подключенных к этому главному серверу, проверяйте результаты команды SHOW SLAVE STATUS, пока подчиненный сервер не будет синхронизирован с главным. Когда подчиненный сервер перешел в это состояние, его можно остановить.

При возобновлении топологии репликации все подчиненные серверы автоматически запустятся без каких-либо задержек. Эта процедура полезна в тех случаях, когда произошел разрыв и требуется восстановить работу, чтобы избежать длительной задержки подчиненных серверов.

Отчеты об ошибках репликации

Может случиться так, что вы встретите проблему репликации, с которой не сможете разобраться без посторонней помощи. Даже если у вас нет соглашения о поддержке, вы можете сообщить об ошибке в MySQL.

Лучшие отчеты об ошибках — те, в которых говорится, как воспроизвести ошибку в тестовой среде. Например, это может быть проблема репликации, которую можно продемонстрировать на главном и подчиненном тестовых серверах, используя минимальный набор необходимых данных. Часто такой отчет может состоять только из полного и точного описания конфигурации и проблемного события.

Чтобы сообщить об ошибке, зайдите на *http://bugs.mysql.com* и зарегистрируйтесь (если еще не регистрировались). Отправляя отчет об ошибке, убедитесь, что описали проблему как можно более полно. Чтобы быстрее получить результат и решить проблему, включите как можно больше деталей:

- дословно запишите все сообщения об ошибках;
- включите двоичный журнал (полностью или выдержку);
- включите журнал ретрансляции (полностью или выдержку);
- запишите конфигурацию главного и подчиненного серверов;
- скопируйте результаты команд SHOW MASTER STATUS и SHOW SLAVE STATUS;
- предоставьте копии всех журналов ошибок.

Сначала попробуйте применить все описанные выше приемы и изучить проблему при помощи всех доступных средств.

Утилита слежения за ошибками MySQL будет сообщать о продвижении в работе над вашим отчетом, отправляя сообщения электронной почты каждый раз, когда отчет об ошибке обновляется.

Заключение

В этой главе мы привели рекомендации по решению ряда распространенных проблем, возникающих при выполнении репликации. Мы рассмотрели их причины, обсудили способы решения и предотвращения в будущем. Также были рассмотрены некоторые инструменты и даны советы по поиску и устранению неполадок репликации MySQL.

Джоэл улыбнулся, услышав шаги за дверью. В этот раз он был готов и начал говорить сразу, как только увидел г-на Саммерсона.

— Я вернул сервер из Сиэтла к работе, сэр. Была проблема с парой запросов, которые не выполнялись, потому что на подчиненном сервере были изменены таблицы. Я посоветовал сотрудникам из того офиса направлять все изменения в схеме и задачи по обслуживанию таблиц на наш главный сервер.

Я послежу за ними и сообщу, если проблема опять появится.

Г-н Саммерсон улыбнулся. Джоэл почувствовал, как на его лбу проступают капли пота.

— Замечательно! Хорошая работа.

Джоэл с облегчением вздохнул.

— Спасибо, сэр.

Г-н Саммерсон собрался уходить, но задержался, как всегда делал, когда хотел дать новое задание, повернулся и произнес:

— Да, кстати...

— Сэр? — поспешил ответить Джоэл.

— Боб.

Джоэл смутился. Обычно г-н Саммерсон формулировал задачи более полными предложениями.

— Сэр? —повторил Джоэл нерешительно.

— Можешь звать меня Боб. Думаю, мы уже прошли этап «сэров».

Джоэл посидел немного, пока не обнаружил, что уставился в пустое пространство. Он моргнул и перевел взгляд на Эми из отдела разработки, стоявшую напротив него.

— Джоэл, ты в порядке?

— Что?

— По-моему, ты отключился. О чем думаешь?

Джоэл улыбнулся и откинулся в кресле.

— Кажется, я наконец-то достучался до него.

— До кого? До г-на Саммерсона?

— Да, он только что сказал, чтобы я звал его Боб.

Эми улыбнулась, скрестила руки на груди и шутливо произнесла:

— Что ж, теперь мы будем называть тебя господин Джоэл.

Джоэл засмеялся и спросил:

— Ты уже обедала?

ГЛАВА 12

Защита инвестиций

Джоэл рылся в ящике стола в поисках ручки, когда услышал знакомый стук в дверь.

— Джоэл, ты должен поговорить с аудиторами, проверяющими нашу информационную безопасность, и донести до них нашу стратегию восстановления в аварийных ситуациях. Да, и не забудь заказать кассеты для копий, чтобы убедить их в нашей готовности к неожиданностям.

— Аудиторы... и кассеты?.. — спросил Джоэл, удивленно уставившись на Саммерсона.

— Сейчас начинай делать отчет по планам восстановления, который они просили. И сообщи в отдел снабжения, какие носители тебе нужны для архивации. Там тебе покажут, какие формы надо заполнить. На всякий случай заказывай с запасом, — с этими словами г-н Саммерсон вышел.

Джоэл задвинул ящик стола и встал. Должно быть, его руководитель имел в виду носители для резервных копий. Но что он хотел сказать, когда говорил об аудиторах и планах? Джоэл подвинул стул к компьютеру и сел.

— Что ж, похоже, придется выяснить, причем тут защита информации и как делать резервные копии, — пробормотал Джоэл, открывая браузер.

В двери снова возник г-н Саммерсон.

— Да, Джоэл, я бы хотел увидеть твой план завтра утром, до совещания, оно начнется в 14:00.

Джоэл вздохнул, поняв, что от него потребуется гораздо больше, чем ему показалось на первый взгляд, и достал книгу по MySQL.

— Не зря потратил деньги, — сказал он себе, открывая главу под названием «Защита инвестиций».

Эта глава посвящена защите данных и подготовке к их восстановлению. Рассматривается практика планирования резервного копирования и восстановления данных, а также сами процедуры резервного копирования и восстановления в MySQL. Обсуждение этих тем было бы неполным без рассмотрения основ защиты информации, обеспечения целостности данных, восстановления в аварийных ситуациях и других связанных концепций. Изучив эту главу, вы поймете, что защита инвестиций — это больше, чем просто резервное копирование и восстановление данных.

Что такое защита информации?

В этом разделе рассказывается о принципах защиты информации, и даются рекомендации по обеспечению информационной безопасности. В рамках этой книги невозможно полностью раскрыть данную тему, но основа для более глубокого изучения будет заложена. По крайней мере, ознакомившись с этой главой, вы сможете внятно объяснить все необходимое руководству.

Защита информации — это управление технологиями администрирования, мониторинга, обеспечения доступности и безопасности информационных систем. Цели защиты информации — контроль и обеспечение безопасности, подлинности, конфиденциальности и доступности данных и систем. В этой книге говорится о СУБД, но базовые принципы защиты информации применимы к любым информационным технологиям. Общие сведения о защите информации можно найти на следующих сайтах:

- *http://www.nsa.gov/ia*.
- *http://en.wikipedia.org/wiki/Information_assurance*.

Три составляющих защиты информации

Некоторые исследователи разделяют защиту информации на три составляющих:

- *информационная безопасность* — защита информационных систем от повреждения посредством управления рисками, внутренними и внешними;
- *целостность информации* — обеспечение постоянной доступности и непрерывной работы систем и данных (иногда это называют *защитой данных*);
- *значимость информации* — оценка значимости системы для пользователя.

Информационная безопасность — наиболее изученная и документированная из трех составляющих. Есть множество руководств о том, как лучше всего обеспечить требуемый уровень защиты вычислительных систем, включая их физическую защиту.

Целостность информации — самая важная составляющая, хотя ей часто уделяют меньше всего внимания. В этой главе будет сказано о важности следования рекомендациям по обеспечению целостности информации.

Значимость информации — составляющая, которую, как правило, не понимают. Организации, практикующие тотальное управление качеством или близкие по духу методы, знакомы с оценкой ценности систем. Но при оценке обычно учитывают денежное выражение стоимости, упуская важность информации для работы сотрудников. Не факт, что то, что хорошо для компании, хорошо и для того, кто в ней работает: мы встречали политики, созданные с самыми лучшими намерениями, которые сильно портили жизнь сотрудникам.

Когда учитывается целостность и значимость информации, меры по обеспечению защиты включают планирование действий в аварийных ситуациях и восстановления данных, о чем часто забывают.

Хорошо спланированная стратегия защиты включает как материальные (защита компьютеров, ключ-карты, аппаратные ключи и т. д.), так и нематериальные (данные, коды доступа и т. д.) аспекты. В этой главе подробно рассматриваются аспекты защиты СУБД, которая может быть одним из важнейших ресурсов организации.

Почему важна защита информации

С ростом важности и стоимости ИТ-систем для организаций растет важность защиты прибылей и подготовки к непредвиденным ситуациям. В результате задача защиты информации встает не только перед исследователями из закрытых учреждений и правительственными организациями (это две наиболее заинтересованные в защите информации группы). Концепции информационной безопасности становятся все более популярными, и многие корпоративные аудиторы берут их на вооружение.

ИТ-специалист должен быть готов решить проблемы организации, защитив ее от непредвиденных ситуаций, включая материальные, виртуальные и финансовые угрозы. Это предполагает защиту корпоративной информации. Если ваша организация имеет собственную программу защиты информации или планирует ее внедрение, вы должны знать, как это связано с вашей работой.

Обеспечение целостности, восстановление информации и роль резервного копирования

Обеспечение целостности информации и *непрерывности бизнеса* обычно считают синонимами. Это подчеркивает важность обеспечения непрерывной работы организации и выполнения управляемого восстановления после непредвиденных аварий. Их результаты могут варьироваться от незначительных проблем до катастрофических поломок и потерь данных, оперативное восстановление после которых невозможно (например, при полном отключении электроэнергии в районе) или возможно только со значительными затратами (если, например, приходится заменять оборудование). Важно понимать, что нет единого рецепта для всех неприятностей и восстановления после них.

Помните, что оборудование ломается — это вопрос времени. Будьте к этому готовы.

Обеспечение целостности информации включает обеспечение:

- *целостности данных* — гарантирует, что данные не будут потеряны либо повреждены и будут актуальными;
- *целостности коммуникаций* — гарантирует, что каналы связи всегда доступны, могут быть восстановлены в случае сбоя и обеспечивают надежную связь;

- *целостности системы* — гарантирует, что в случае сбоя систему можно перезапустить, восстановить ее данные и состояние, обеспечивая необходимый уровень непрерывности работы.

С усилением зависимости бизнеса от информационных систем работоспособность последних становится критичной для деятельности компаний. Действительно, большинство современных компаний, использующих ИТ, стали настолько зависимыми от них, что сложно провести границу между бизнесом и работой ИТ-системы предприятия. Иными словами, бизнес не может существовать без информационных систем.

В такой ситуации необходимо понять, что эффективная работа (и даже работа вообще) невозможна при выходе из строя ИТ-системы. Очень важна возможность скорейшего восстановления системы. Возврат системы к работе после непредвиденного сбоя называется *аварийным восстановлением*.

В компаниях, попавших под наблюдение регулирующих организаций, знают (или скоро узнают), что существуют нормативы и стандарты, требующие внедрение средств информационной безопасности и аварийного восстановления на предприятии, особенно для отдельных типов данных. Примерами таких нормативов в США могут быть закон об отчетности и сохранности данных медицинского страхования (Health Insurance Portability and Accountability Act, HIPAA), Патриотический закон (Patriot Act) и закон Сарбейнса-Оксли (Sarbanes-Oxley Act, SOX).

С законом Сарбейнса-Оксли можно ознакомиться по адресу *http://www.soxlaw.com*.

Высокая доступность или аварийное восстановление?

В большинстве компаний понимают необходимость инвестиций в технологии, позволяющие быстро восстанавливать системы после сбоев любой степени серьезности. К таким технологиям относятся RAID-массивы, репликация, дополнительные источники питания и др. Все это относится к обеспечению высокой доступности, так как позволяет мгновенно или очень быстро восстанавливать работоспособность без потери данных (или свести потери к очень небольшим фрагментам, как, например, последняя измененная запись).

К сожалению, лишь немногие компании предпринимают дополнительные меры по защите от дорогостоящих потерь, используя средства аварийного восстановления. Конечно, средства аварийного восстановления и обеспечения высокой доступности в чем-то совпадают. Решения для обеспечения высокой доступности могут учитывать незначительные или даже крупные аварии. Тем не менее, эти аспекты информационной безопасности различаются. Высокая доступность обеспечивает защиту от известных или предсказуемых угроз, тогда как аварийное восстановление защищает от неожиданных ситуаций.

Аварийное восстановление

Аварийное восстановление включает процедуры, политики и планы по обеспечению целостности информации в случае катастрофы. Наиболее важным аспектом является создание и поддержание в актуальном состоянии *плана аварийного восстановления*.

Об аварийном восстановлении написано уже очень много. В этом разделе приводятся лишь общие сведения, позволяющие оценить важность аварийного восстановления.

В главе 2 уже сказано об одном из способов аварийного восстановления. Специалисты всегда используют средства восстановления после незначительных аварий, но что, если случится что-то серьезное, например полный отказ массива RAID или пожар в серверной?

Прежде чем планировать действия в худшей ситуации, следует ответить на несколько вопросов, чтоб определить предпосылки и цели плана аварийного восстановления. Кроме того, эти вопросы помогут определить критерии для оценки эффективности плана.

- К каким физическим и виртуальным угрозам следует готовиться?
- Какой уровень функционирования необходим или желателен для поддержания функционирования компании?
- Как долго компания может ожидать восстановления работы ИТ-систем?
- Какие ресурсы доступны для планирования аварийного восстановления и выполнения этого плана?

Первый этап аварийного восстановления — определение худшей из возможных ситуаций: полная потеря центра обработки данных (ЦОД) в результате катастрофы. Имеется в виду не только потеря отдельных компонентов (серверов, рабочих станций, сетевого оборудования и т. д.), но и вычислительной системы в целом. И не следует забывать о потере персонала, обслуживающего системы (данный фактор редко учитывают, хотя делать это нужно).

Это может показаться страшилками о судном дне, но события, приводившие к большим разрушениям, происходили уже не раз, и будут происходить снова. Это может быть широкомасштабное отключение электроэнергии либо стихийное бедствие, такое как ураган или землетрясение, война, наконец. Представьте, что здание, в котором расположены ваши вычислительные средства, полностью разрушено ураганом. Крыши нет, 90% оборудования повреждено водой, и все лежит в руинах. Предположим также, что бедствие произошло во время выполнения дорогостоящей и срочной транзакции. Как компания будет восстанавливаться в этом случае? Имеется ли какой-то план на этот случай? Как повлияет на доходы компании длительная потеря работоспособности? Как быстро может быть восстановлена работоспособность? Такие размышления могут мотивировать руководство компании к вложению денег в средства аварийного восстановления.

Дополнительным стимулом может быть то, что готовность к худшему позволит преодолевать рядовые неприятности гораздо меньшей кровью. Планирование — важнейший аспект аварийного восстановления, позволяющий обеспечить непрерывность работы.

Бедствия не делают исключений

Наводнения, пожары и другие бедствия могут происходить везде. В некоторых регионах стихийные бедствия случаются чаще, чем в других. Однако, независимо от того, где находится ваша организация, всегда есть некоторый риск. Но бедствия определенного рода могут случаться где угодно и в любое время. Это бедствия по вине человека.

Все организации должны быть готовы к злонамеренным действиям людей, в том числе своих сотрудников. И даже в самых защищенных системах найдутся уязвимые места.

Предположим, ваша компания находится в местности, где стихийные бедствия маловероятны. Землетрясения и наводнения не представляют реальной угрозы. Здание имеет отличную противопожарную систему, защищено физически и находится под круглосуточным наблюдением. Риск физических потерь очень невелик. Ваш предшественник подумал обо всем, и руководство не считает, что есть потребность в плане аварийного восстановления.

А теперь допустим, что какой-то сотрудник захочет навредить компании. Работая над планом аварийного восстановления, спросите себя: «Какой ущерб сможет нанести наш собственный сотрудник системам, данным или инфраструктуре?» Такие мысли могут показаться несерьезными и даже параноидальными, но печальный опыт организаций, ставших жертвами внутреннего саботажа, говорит о реальности такой угрозы.

Хороший план аварийного восстановления включает меры по снижению риска саботажа, указывает на потенциальные уязвимые места, включая физический доступ в здание и к оборудованию, и содержит описание всех административных прав для работы с важными системами и данными.

Цель аварийного восстановления — скорейшее восстановление работоспособности организации, что подразумевает восстановление и физических и электронных компонентов ИТ-систем. Хороший план включает восстановление или переезд офиса организации, а также восстановление ИТ-систем. Следует также предусмотреть варианты замены оборудования: взять резервное, использовать оставшееся или приобрести новое.

Воссоздание сети и данных с нуля требует некоторого минимального набора вычислительного и сетевого оборудования, позволяющего вернуть данные и приложения в состояния, в котором организация может функционировать на приемлемом, хотя и ограниченном уровне. Следовательно, необходимо определить минимальный набор технологий, позволяющий организации работать с минимальными потерями прибыли или без потерь вообще.

Персонал — тоже ключевой ресурс, который необходимо учитывать. Одной из процедур восстановления должно быть уведомление всех сотрудников о чрезвычайной ситуации. Это может быть простой «обзвон», когда каждый сотрудник должен позвонить нескольким коллегам и передать важную информацию. Или это может быть сложная автоматизированная система оповещения, обзванивающая всех сотрудников и передающая заранее записанное сообщение. Большинство автоматических систем подсчитывает число оповещенных, запрашивая у каждого подтверждение получения сообщения. Лучшие системы оповещения поддерживают различные способы связи. Например, в первую очередь система может звонить на домашние номера, а затем использовать электронную почту, номера мобильных телефонов и текстовые сообщения.

Политика аварийного восстановления также включает оценку рисков во время чрезвычайных ситуаций (какие ресурсы в опасности и насколько важен каждый из них?) и определение ответственных в случае недоступности (или даже гибели) руководящих сотрудников. Например, если ответственный за сетевые системы недоступен, его обязанности (и сфера ответственность) переходят к одному из его подчиненных.

Планирование аварийного восстановления

Планирование — очень обширная область, требующая учета последствий всех известных или вероятных ситуаций. Например, хороший план включает описание процедур развертывания ИТ-систем в другом месте и пошаговые инструкции по воссозданию информационных систем с нуля.

Важно отметить один фатальный недостаток некоторых планов восстановления. Время и ресурсы, потраченные на создание плана аварийного восстановления, будут потрачены впустую, если этот план будет храниться на жестких дисках систем, которые нужно будет восстанавливать. Копию плана восстановления следует хранить в другом месте и в несгораемом сейфе.

Процесс аварийного восстановления

К этому моменту у вас могли возникнуть вопросы. Как начинать аварийное восстановление? Как перейти от составления описи ИТ-систем к составлению плана восстановления? Все начинается с определения целей. Затем можно приступать к планированию и, в конечном итоге, к составлению собственного плана аварийного восстановления.

Если вам показалось, что предстоит немало работы, вы не ошиблись. В большинстве организаций для этого создается специальная группа экспертов. Наиболее успешные группы состоят из специалистов, понимающих риски для организации (организация может обанкротиться, а все ее сотрудники — потерять работу), и достаточно разбирающихся в своих областях, чтобы выявить важнейшие системы.

Это может показаться очевидным, но необходимо учитывать возможность того, что реализация плана может лечь на плечи других, а не его создателей. Это подчеркивает важность создания подробной и ясной документации, а также четкого распределения обязанностей с некоторой избыточностью.

Ниже вкратце описаны этапы типичного плана аварийного восстановления. Подробнее эти этапы рассматриваются в последующих разделах.

1. Создайте группу аварийного восстановления. Если в вашей организации много сотрудников, сообщите руководителю, что требуется сформировать группу для работы над планом аварийного восстановления. Определите ключевые роли, для исполнения которых потребуются члены группы. Включайте в группу персонал из разных отделов организации, избегайте соблазна укомплектовать группу только ИТ-специалистами. Вам потребуются разные точки зрения, и получить их можно только от специалистов в разных областях.
2. Определите цели и задачи организации. Проведите исследование текущего состояния деятельности, выясните приемлемый минимум работоспособности и определите цели процедуры аварийного восстановления.
3. Вовлеките руководство. Если инициатива исходила не от руководства, ваше задача — убедить руководителей в том, что аварийное восстановление потребует их участия. Это может быть единственным способом получения необходимых ресурсов (время, финансы, персонал).
4. Создавайте планы для самых худших ситуаций. Данный этап многие считают самым интересным. Нужно записывать сценарии различных бедствий, которые могут произойти в вашем регионе: кражи, саботаж, стихийные бедствия, пожары, эпидемии и т. д. Эти сценарии потребуются в дальнейшем, когда вы начнете создавать планы аварийного восстановления.
5. Проведите оценку доступных материальных и организационных ресурсов. Составьте полный список ИТ-систем организации. Включите в него все, что необходимо для успешной работы, включая рабочие станции сотрудников и сетевые подключения. В список нужно включить то, что имеется сейчас, а не то, что вы считаете минимальным набором. Сокращать список до минимально приемлемого набора будете позже. Следует также описать текущую структуру организации и структуру руководства, перечислив сотрудников, принимающих решения, и их заместителей. Запишите все это и сохраните для использования в дальнейшем.
6. Оцените риски. Выясните, к чему приведет потеря каждого из ресурсов. Это позволит определить минимальный уровень работоспособности. Следует определить приоритет каждого ресурса и, возможно, несколько приемлемых уровней работоспособности. Не забудьте включить подробное описание необходимых приложений и данных. Эта информация поможет решить, какие процедуры необходимо использовать в разных ситуациях. Например, для восстановления после частичной потери можно выполнить повторное развертывание уцелевших систем, а не строить

новые системы, сэкономив время и деньги. Может быть, какое-то время можно работать с ограниченной работоспособностью.

7. Разработайте план действий в непредвиденных ситуациях. Теперь, имея сценарии возможных бедствий, список ресурсов и оценку рисков, можно приступать к написанию первого чернового варианта плана аварийного восстановления. Планы действий можно составлять в любом удобном для вас виде, но большинство планировщиков используют списки с описаниями.
8. Создайте процедуры проверки. Никакой план не будет полезным, если он будет провален во время выполнения. Разработайте процедуры проверки своего плана аварийного восстановления. Начните с проверки списка угроз, внося необходимые изменения. Вернитесь к этапу 4 и повторите последующие этапы, совершенствуя план.
9. Практика — залог успеха. Закончив составление плана, попробуйте выполнить его на практике, чтобы убедиться, что он работает. Конечной целью будет демонстрация руководству возможности достижения приемлемого уровня работоспособности. Если это не удалось, вернитесь к этапу 6 и повторяйте последующие этапы, пока не получите законченный набор процедур, которые можно воспроизвести.

План аварийного восстановления необходимо полностью проверять хотя бы раз в год или после каждого значительного изменения в организации и ее ИТ-системах.

Средства аварийного восстановления

Хотя аварийное восстановление охватывает не только данные и компьютеры, основное внимание всегда уделяется именно им. Существуют различные инструменты и приемы, которые можно использовать в планах аварийного восстановления. Ниже перечислены наиболее популярные из них.

- *Резервное питание* Всегда предусматривайте источники бесперебойного питания. Длительность работы и стоимость такого оборудования во многом будет зависеть от допустимой длительности простоя компании. Если операции должны выполняться непрерывно, потребуется система, способная поддерживать работу оборудования долгое время.
- *Сетевые подключения* Если есть нужда в электронной связи с клиентами и партнерами, могут потребоваться дополнительные или альтернативные средства сетевого доступа. Это может быть просто другой носитель (например, дополнительная линия оптоволокна) или же сложный набор альтернативных точек подключения (спутниковая или сотовая связь).
- *Альтернативный офис* Если компании требуется полная работоспособность и минимальное время простоя, необходимо обеспечить возможность быстрого восстановления без потери данных и функциональных возможностей. Лучшее решение — предусмотреть альтернативный офис для ИТ-систем или компании целиком. Это можно реализовать в виде

мобильного офиса (временного передвижного офиса), содержащего копии ИТ-систем.

- *Дополнительный персонал* При планировании аварийного восстановления часто забывают об этом компоненте. К ситуациям, которые следует учитывать, относится потеря части или всего персонала, занимающего ключевые должности. Это может быть результатом инфекции, поглощения конкурирующей фирмой или даже смерти сотрудников. Независимо от причины, следует проанализировать ключевые должности в организации и предусмотреть замещение соответствующих сотрудников в случае необходимости. Одно из очевидных решений — перекрестное обучение персонала. Это обеспечит взаимозаменяемость и повысит ценность сотрудников.
- *Резервное оборудование* Наверно, самое очевидное требование для обеспечения непрерывной работы — наличие резервного оборудования. Технологии обеспечения высокой доступности могут частично удовлетворить это требование, но лучшим вариантом будет размещение резервного оборудования в альтернативном месте, что позволит быстро включить его в работу. Это оборудование не должно пылиться без дела: на него следует устанавливать все текущие обновления ПО и регулярно проверять.
- *Безопасное хранилище* Еще один аспект, о котором часто забывают, — безопасное хранилище для резервных копий важных данных. Следует подготовить такое хранилище, уровень защиты которого должен зависеть от важности данных и требуемой скорости восстановления. Так, если время простоя должно быть очень коротким, банковский сейф не позволит извлечь данные достаточно быстро. А если резервные копии хранятся в общедоступном месте, это может быть небезопасно.
- *Высокая доступность* Как мы уже говорили, средства обеспечения высокой доступности составляют первый уровень защиты в плане аварийного восстановления. Возможность использования альтернативных хранилищ данных облегчит восстановление после незначительных аварий.

Всегда храните копии данных и план аварийного восстановления в безопасном месте.

Запомните главное правило, применимое ко всем средствам и стратегиям аварийного восстановления: чем быстрее нужно выполнить восстановление и чем больше функциональности требуется восстановить, тем выше будет стоимость. Поэтому важно определить несколько уровней работоспособности, чтобы можно было адаптировать план восстановления к различным обстоятельствам. Возможно, компания находится на спаде или имеет финансовые проблемы, и работоспособностью частично придется пожертвовать. Чем больше вариантов вы предусмотрите, тем надежнее будет план восстановления.

Важность восстановления данных

Большинство компаний считает информацию своим важнейшим ресурсом (после персонала, который очень сложно заменить), поэтому первоочередная задача большинства планов аварийного восстановления — восстановление данных и обеспечение их доступности. Да, информация решает все.

Когда говорят о восстановлении данных, имеют в виду планы, политики и процедуры для восстановления информации до приемлемого уровня, обеспечивающего работоспособность компании. Возможность успешного архивирования и восстановления данных иногда называют *гибкостью* или *восстанавливаемостью*.

Ключевые аспекты восстановления данных — возможность спланировать, выполнить и проконтролировать операции резервного копирования и восстановления. О последнем аспекте часто забывают. Хорошей системой резервного копирования и восстановления будет та, на которую можно положиться. Очень часто системы резервного копирования допускают сбои без какого-либо указания на то, что данные отсутствуют, повреждены или уничтожены. Обычно это обнаруживается только при попытке восстановления данных. Понятно, что это может очень затруднить аварийное восстановление.

Терминология

Почти во всех руководствах по восстановлению данных и решениям для резервного копирования встречается два термина, которые важно понимать:

- Целевая точка восстановления (Recovery Point Objective, RPO) — состояние системы, пригодное для использования. Таким образом, RPO представляет максимальное приемлемое ухудшение работоспособности или максимальную приемлемую потерю данных. Используется для оценки успешности восстановления.
- Целевое время восстановления (Recovery Time Objective, RTO) — максимальный промежуток времени, в течение которого допустима потеря работоспособности (также называется временем простоя).

RPO сильно влияет на RTO. Если требуется восстановить все данные (или почти все, например, все реплицированные транзакции), понадобится больше работы, поэтому возможность быстрого восстановления (низкое значение RTO) будет стоить дороже. Высокое значение RPO или низкое значение RTO требует больших вложений в оборудование и ПО, а также обучение администраторов использованию сложных систем, обеспечивающих необходимые значения RTO и RPO. Например, если RTO меньше трех секунд, а RPO — полная работоспособность без потерь данных, необходимо сложное решение для обеспечения высокой доступности, позволяющее произвести восстановление после полной потери сервера, хранилища данных и т. д. без потери данных (или времени).

Резервное копирование и восстановление

В этом разделе описываются два важнейших инструмента восстановления данных. Мы подробно рассмотрим концепции резервного копирования и восстановления, а также поговорим о необходимом для этого планировании и доступных решениях.

Для СУБД, таких как MySQL, резервное копирование и восстановление — это возможность создания копий данных, которые можно хранить и при необходимости восстанавливать, возвращая данные в то состояние, в каком они находились в момент создания копии.

Последующие разделы пригодятся тем, кто не знаком с системами резервного копирования и доступными решениями для MySQL. Мы рассмотрим преимущества различных решений и расскажем о создании плана архивирования, позволяющего восстанавливать данные с минимальными потерями.

Зачем нужно резервное копирование?

Кто-то может подумать, что резервное копирование не нужно, если используется репликация или какой-то из вариантов обеспечения аппаратной избыточности. Это может быть справедливо, когда требуется быстро восстановить данные после механического или электронного сбоя, но ничем не поможет в восстановлении после серьезной аварии.

В табл. 12-1 описаны некоторые из наиболее вероятных способов потери данных и варианты их восстановления. Видно, что во многих случаях помогают резервные копии. Кроме восстановления после сбоев и ошибок, существует несколько весомых причин для включения резервного копирования в ежедневные планы по защите информации. Сюда относится создание нового подчиненного сервера в топологии репликации для использования его в качестве резервного или для передачи данных из одной сети в другую.

Табл. 12-1. Распространенные причины потери данных

Тип сбоя	Описание	Способ восстановления
Ошибка пользователя	Пользователь случайно удаляет данные или выполняет обновление с неверными значениями	Восстановите удаленные данные при помощи резервной копии
Сбой питания	Одна или несколько систем теряют питание	Используйте источники бесперебойного питания
Аппаратные сбои	Один или несколько компонентов системы выходят из строя	Используйте избыточные системы или реплицируйте данные
Программные сбои	Данные изменяются или теряются в результате трансформации	Эти сбои могут быть трудно обнаружимыми и требовать восстановления из резервной копии (если нельзя откатить изменения)

Табл. 12-1. (*окончание*)

Тип сбоя	Описание	Способ восстановления
Проблемы с помещением	Помещение, в котором находится оборудование, становится недоступным и доступ к данным теряется	Может потребоваться новое помещение для возобновления работы на требуемом уровне
Сбои сети	Серверы, содержащие данные, становятся недоступными	Может потребоваться возобновление работы сети или переход на использование нового хранилища данных
Саботаж	Данные украдены или намеренно повреждены	Устраните причину проблемы, проверьте и исправьте данные

Если вы имеете собственную плановую процедуру резервного копирования и используете стороннее хранилище, возможно, вы подготовлены к аварийному восстановлению лучше, чем думаете. Например, подумайте, что случится, если ваша СУБД, все дублирующее оборудование и даже репликант (подчиненный сервер в топологии репликации MySQL) внезапно исчезнут в результате кражи или катастрофы. Резервные копии, хранящиеся в удаленном хранилище, позволяют восстановить данные и системы до момента последнего копирования.

Сбои оборудования

В табл. 12-1 описаны некоторые распространенные причины потери данных. Однако с оборудованием могут возникать самые разные проблемы, которые могут быть гораздо серьезнее приведенных в табл. 12-1. Рассмотрим некоторые проблемы и способы их решения.

- *Потеря более одного диска из аппаратного массива* Если из строя выходит несколько дисков из одного массива RAID, восстановить массив скорее всего не удастся. Если такое произошло (а это происходит чаще, чем можно подумать), остается только восстановить данные из резервных копий.
- *Сбой дисков на главном и подчиненном серверах* Если система выходит из строя и на главном, и на подчиненном сервере (особенно в результате сбоя диска), вы можете остаться без резервного сервера, готового к работе. Это может произойти по разным причинам. Например, может повредиться определенная таблица, или диск, на котором эта таблица хранится, выйдет из строя как на главном, так и на подчиненном сервере. И снова единственным решением будет восстановление из резервных копий.
- *Сбой резервного источника питания* Наверно, одна из самых неприятных ситуаций — когда резервная система питания выходит из строя в тот момент, когда она нужна — во время отключения электроэнергии. Чтобы подготовиться к этому, можно установить несколько источников резервного питания.

Теперь допустим, что ваша СУБД работает на недорогом оборудовании (MySQL знаменита независимостью от оборудования). Вы можете заказать оборудование для замены или использовать существующее оборудование и быстро запустить сервер баз данных (конечно, скорость будет зависеть от объема восстанавливаемых данных).

Суть в том, что регулярное выполнение резервного копирования — основной принцип восстановления данных. Средства обеспечения высокой доступности, такие как репликация и массивы RAID, лучше в том смысле, что позволяют выполнить восстановление почти мгновенно, но они ничем не помогут в случае катастрофической потери. Вы уже видели, что репликация помогает предотвратить потерю данных, но что произойдет, например, если в результате катастрофы будут безвозвратно потеряны и главные, и подчиненные серверы? В этом случае поможет только недавно созданная резервная копия.

В следующих разделах мы подробнее расскажем о резервном копировании и покажем, как быстро приступить к созданию резервных копий данных из MySQL.

Требования к резервным копиям

Операция резервного копирования должна создавать копии данных, которые в дальнейшем можно восстановить. Кроме того, резервные копии должны быть последовательными. Для базы данных, использующей транзакции, это означает, что резервная копия содержит только те транзакции, которые были завершены к моменту начала копирования, и не содержит незавершенных. Также должны иметься какие-либо средства мониторинга, позволяющие следить за процессом резервного копирования и проверять состояние данных.

Существует несколько типов резервного копирования:

- *Полное копирование* Создается полная копия всех данных с сервера. Ничего не пропускается. Это самый долгий и требовательный к дисковому пространству способ копирования.
- *Разностное копирование* Копируются только те данные, которые были изменены с момента последнего полного копирования. Обычно этот тип копирования требует меньше дискового пространства и выполняется гораздо быстрее.
- *Инкрементное (добавочное) копирование* Копируются только данные, изменившиеся с момента последнего разностного или полного копирования. Обычно такое копирование выполняется в виде создания журнала изменений (т.е. двоичного журнала в MySQL). Обычно такое копирование выполняется гораздо быстрее разностного и полного копирования и, в зависимости от числа изменений с момента последнего резервного копирования, может требовать минимум места на диске.

Разрабатывая план восстановления данных, выберите схему именования архивов (их также называют резервными образами или резервными фай-

лами). Например, имя может содержать дату и другую информацию: *full_backup_2009_09_09* или *incr_backup_week_3_september*. Нет «правильных» или «неправильных» имен, так что можете давать архивам такие имена, какие считаете подходящими.

Требования к процессу восстановления

Операция восстановления должна заменять данные в системе данными из архива так, что замененные данные были бы идентичными данным из архива. Как и копирование, процесс восстановления должен поддерживать средства мониторинга, чтобы можно было следить за этим процессом и проверять состояние данных.

К сожалению, немногие системы резервного копирования удовлетворяют всем требованиям для процессов копирования и восстановления, а те, которые удовлетворяют, часто являются специализированными платформами (наборами специализированного оборудования и ПО), дорогостоящими и сложными в обслуживании. В этой главе описываются экономичные варианты систем резервного копирования и восстановления данных MySQL.

Логическое и физическое резервное копирование

Одна из неправильно понимаемых концепций резервного копирования — разница между логическим и физическим режимами. Выбранный вариант может значительно повлиять на эффективность резервного копирования и доступные варианты восстановления данных.

Логическая резервная копия — это просто набор обычных SQL-запросов SELECT. Резервное копирование выполняется посредством сканирования таблицы с прохождением каждой строки данных.

Физическая резервная копия — это копия исходных двоичных данных (файлов), часто создаваемая на уровне операционной системы. Любой метод резервного копирования, включающий копирование данных, индексов и буферной памяти (файлов) без использования построчного доступа к БД, считается методом физического резервного копирования.

Можно догадаться, что логическое резервное копирование выполняться гораздо медленнее физического, так как система должна использовать обычные механизмы SQL для последовательного чтения записей. При физическом копировании обычно используются функции операционной системы без привлечения лишних механизмов. Но для физического копирования может потребоваться блокирование таблиц, связанных с двоичными файлами, до завершения двоичного копирования, тогда как некоторые способы логического копирования не требуют блокировки доступа к таблицам во время резервного копирования.

Выбрать метод не так просто, как вы думаете. Например, если нужно сделать копию сервера баз данных MySQL, включающую все данные, можно просто отключить сервер и скопировать каталог *mysql* целиком на другой

компьютер. Также нужно скопировать каталоги с файлами InnoDB, если их стандартное расположение изменено. Таким образом будет создан второй экземпляр сервера, идентичный первому. Этот способ хорошо подходит для построения топологии репликации, но очень неудобен для резервного копирования нескольких важных баз данных, которые нельзя отключать. В таком случаем логическое копирование может быть лучшим (и единственным) вариантом.

Создание плана архивирования

Для успешного применения средств резервного копирования и восстановления требуется подготовка и немного размышлений. Часто пропускают этап разработки *плана архивирования*, на котором определяют, как часто необходимо делать копии данных.

Первый вопрос, который нужно задать себе: «Какой объем данных допустимо потерять в случае восстановления с нуля?» Другими словами, какой объем данных позволительно не восстанавливать в случае полной потери данных? Это будет вашим значением RPO или уровнем работоспособности, необходимым компании для ведения дел.

Очевидно, что любая потеря данных нежелательна. Необходимо провести оценку данных перед определением значений RPO. Может оказаться, что одни данные более важны, чем другие. Некоторые данные могут быть ключевыми для процветания и стабильности компании, тогда как другие могут быть не так важны. Понятно, что самые важные данные должны быть восстановлены без потерь. Таким образом, можно получить несколько разных RPO. Всегда рекомендуется определять несколько уровней, а затем решать, на каком уровне следует проводить восстановление в конкретном случае.

Это важно, так как план архивирования определяет, как часто необходимо выполнять резервное копирование, и какой объем данных будет скопирован. Если недопустимы любые потери данных, то, скорее всего, придется использовать варианты обеспечения высокой доступности, включающие репликацию данных на сервер, расположенный в безопасном месте. Но даже это не обеспечивает стопроцентную защиту. Например, повреждение или саботаж главного сервера может повлиять на репликацию и оставаться незамеченным какое-то время.

Скорее всего, допустимость потери данных будет определяться максимальным промежутком времени, в течение которого можно восстанавливать потерянные данные. Т. е. нужно понять, как долго восстанавливаемые данные могут быть недоступными. От этого напрямую зависят финансовые потери организации во время простоя. Для некоторых компаний каждая секунда без доступа к данным стоит очень дорого. Возможно также, что одни данные сильнее влияют на доход компании, чем другие, так что какие-то данные можно восстанавливать дольше.

Следует определить несколько сроков, приемлемых в разных обстоятельствах. Так формируются значения RTO, которые можно сопоставить с уровнями RPO, чтобы получить представление о том, сколько времени потребуется для каждого уровня восстановления. Определение уровней RTO может потребовать повторного ввода данных в систему или повторного выполнения работы по получению данных (например, повторения загрузки данных или получения обновления от бизнес-партнера).

Определив допустимые потери данных (RPO) и допустимое время простоя (RTO), следует проверить возможности системы резервного копирования и выбрать частоту и способ копирования, удовлетворяющие вашим потребностям. Но не останавливайтесь на этом. Создайте задачи для автоматического выполнения резервного копирования, запускающиеся в наиболее подходящее (или наименее неудобное) время.

Наконец, следует периодически выполнять проверки, выполняя пробное восстановление данных. Так вы будете уверенными в наличии рабочих копий без дефектов и снизите риск потери данных при восстановлении.

Утилиты резервного копирования и решения на уровне ОС

Oracle предоставляет решение InnoDB Hot Backup для резервного копирования таблиц InnoDB и MyISAM. Кроме того, есть несколько решений для выполнения резервного копирования в MySQL от сторонних производителей. Также можно выполнять некоторые несложные, но эффективные операции резервного копирования на уровне операционной системы.

В последнем разделе этой главы вкратце рассматриваются наиболее популярные альтернативы:

- InnoDB Hot Backup;
- физическое копирование файлов;
- утилита mysqldump;
- XtraBackup;
- мгновенные снимки Logical Volume Manager.

Существуют и другие инструменты, но большинство из них аналогично перечисленным.

Приложение InnoDB Hot Backup

Если вы используете InnoDB как единственный механизм БД или регулярно выполняете резервное копирование таблиц InnoDB, можете использовать приложение Oracle InnoDB Hot Backup. Его основное преимущество состоит в том, что в процессе резервного копирования базы данных остаются доступными для запросов и обновлений, и не нужно отключать сервер или явно блокировать таблицы.

С помощью Hot Backup можно копировать также таблицы MyISAM, но во время резервного копирования эти таблицы будут блокироваться.

InnoDB Hot Backup создает последовательные копии всех баз данных, не блокируя какие-либо транзакции. Можно копировать все базы данных или только некоторые, сжимать полученный файл с резервными копиями, а также выполнять выборочное копирование (например, копировать только указанные таблицы).

Это приложение доступно для MySQL 4.0 и выше и может работать на платформах Linux, Unix и Windows. Подробнее см. на веб-сайте Innobase Hot Backup (*http://www.innodb.com/products/hot-backup*).

На конференции пользователей MySQL в 2010 г. Oracle объявила о том, что со следующего выпуска InnoDB Hot Backup станет называться MySQL Enterprise Backup. В продукте с новым названием появится много новых функций.

Решение InnoDB Hot Backup состоит из двух файлов: *ibbackup* и *innobackup*. В следующих разделах мы расскажем об этом приложении и покажем, как с его помощью выполнять резервное копирование и восстановление данных.

Базовой утилитой резервного копирования является *ibbackup*. Эта утилита может выполнять три основных операции: копировать данные, применять к резервной копии команды из журнала и восстанавливать данные. Сценарий Perl с названием *innobackup* обеспечивает интерфейс высокого уровня и предоставляет некоторые дополнительные функции.

Резервное копирование с помощью ibbackup

Утилита *ibbackup* работает немного необычно: параметры копирования передаются в двух файлах, указанных в командной строке. Первый файл, которым обычно является стандартный файл конфигурации *my.cnf*, содержит информацию о копируемых БД. Второй файл содержит информацию о файлах, которые будут созданы для хранения резервной копии. Директивы в обоих файлах имеют одинаковые имена, а их значения указывают на информацию об этих файлах в файловой системе, хранящей базы данных. Типичная команда выглядит так:

```
ibbackup my.cnf backup.cnf
```

Каждый файл содержит следующие параметры:

```
datadir = каталог
innodb_data_home_dir = каталог
innodb_data_file_path = список_параметров
innodb_log_group_home_dir = каталог
innodb_log_files_in_group = номер_группы
innodb_log_file_size = размер
```

Определения этих параметров можно найти в онлайновом руководстве MySQL Reference Manual. Ниже показано содержимое файла конфигурации

с типичными параметрами первого файла, описывающего данные, которые нужно скопировать:

```
[mysqld]
datadir = /usr/local/mysql/data
innodb_data_home_dir = /usr/local/mysql/data
innodb_data_file_path = ibdata1:10M:autoextend
innodb_log_group_home_dir = /usr/local/mysql/data
set-variable innodb_log_files_in_group = 2
set-variable innodb_log_file_size = 20M
```

Теперь допустим, что нужно записать файлы с резервными копиями в каталог */home/cbell/backup*. Тогда содержимое файла конфигурации для команды ibbackup, которому мы дали имя *backup.cnf*, будет следующим:

```
datadir = /home/cbell/backup
innodb_data_home_dir = /home/cbell/backup
innodb_data_file_path = ibdata1:10M:autoextend
innodb_log_group_home_dir = /home/cbell/backup
set-variable innodb_log_files_in_group = 2
set-variable innodb_log_file_size = 20M
```

Можно сжать файл с резервной копией, используя параметр --compress:

```
ibbackup --compress my.cnf backup.cnf
```

Применение к резервной копии команд из журнала

Поскольку базы данных InnoDB продолжают работать во время выполнения резервного копирования, скопированные данные могут не быть актуальными. Чтобы получить актуальную копию, нужно синхронизировать данные, применив к файлам с резервными копиями команды из текущего журнала. Это возможно, так как утилита *ibbackup* создает журнал с именем *ibbackup_logfile*, содержащий изменения, внесенные за время резервного копирования. Этот журнала можно применить к файлам резервных копий, чтобы перевести данные в то состояние, в котором они находились в момент завершения операции резервного копирования.

Чтоб сделать это, понадобится только файл конфигурации резервного копирования и параметр --apply-log:

```
ibbackup --apply-log /home/cbell/backup.cnf
```

Если используется параметр compress, нужно указать параметр --uncompress, чтобы применить журнал к сжатой резервной копии:

```
ibbackup --apply-log --uncompress /home/cbell/backup.cnf
```

Восстановление данных при помощи ibbackup

Для восстановления данных требуется применение журнала, описанного в предыдущем разделе. По сути, эта операция указывает экземпляру MySQL

на резервные копии базы данных. Она не копирует файлы в обычное расположение *datadir* MySQL. Если нужно сделать это, необходимо скопировать файлы вручную или использовать сценарий *innobackup*. Причина этого заключается в том, что утилита *ibbackup* разработана так, чтобы никакие данные не затирались. Чтобы запустить экземпляр MySQL и использовать резервную копию данных, выполните такую команду:

```
mysqld -defaults-file=/home/cbell/backup.cnf
```

Сценарий innobackup

Файл *innobackup* — это сценарий Perl, предназначенный для автоматизирования многих операций, выполняемых утилитой *ibbackup*. С его помощью можно создавать резервные копии, восстанавливать данные, запускать экземпляр MySQL с использованием данных из резервной копии, а также копировать данные, индексы и файлы журналов из резервного каталога в исходные места.

В Windows сценарий *innobackup* в настоящее время недоступен.

В отличие от утилиты *ibbackup*, для этого сценария не требуется указывать команды в отдельном файле конфигурации. Параметры копирования или восстановления указываются в командной строке. Эти параметры описаны в табл. 12-2.

Табл. 12-2. Параметры innobackup

Параметр	Функция
--help	Показывает список всех параметров
--version	Показывает версию сценария
--apply-log	Применяет к резервной копии журнал резервного копирования для подготовки к запуску сервера MySQL с файлами резервных копий
--copy-back	Копирует данные и файлы индекса из резервной копии в исходные места
--use-memory=MB	Передается ibbackup; определяет, сколько памяти будет использоваться в процессе восстановления
--sleep=MS	Передается ibbackup; указывает, что утилита должна делать паузу после копирования каждого мегабайта данных
--compress=LEVEL	Передается ibbackup; определяет уровень сжатия
--include=REGEXP	Передается ibbackup; указывает, что копироваться должны только те файлы таблиц, которые соответствуют введенному регулярному выражению. Используется для выбороч-ного копирования
--uncompress	Передается ibbackup; разархивирует сжатую резервную копию

Табл. 12-2. (*окончание*)

Параметр	Функция
--user=ИМЯ	Имя пользователя, использующееся для подключения к серверу
--password=ПАРОЛЬ	Пароль пользователя
--port=ПОРТ	Порт сервера
--socket=СОКЕТ	Сокет сервера

Выполнение резервного копирования при помощи innobackup

Для создания резервной копии потребуется всего два параметра: файл конфигурации с сервера и расположение файлов резервных копий:

```
perl innobackup /etc/mysql/my.cnf /home/cbell/backup
```

Если хотите получить актуальную резервную копию, примените параметр --apply-log:

```
perl innobackup --apply-log /etc/mysql/my.cnf /home/cbell/backup
```

Восстановление данных при помощи innobackup

Чтобы восстановить данные, примените журнал и используйте параметр --copy-back для копирования файлов в исходные места. Ниже показаны примеры команд. Сервер необходимо остановить перед копированием и запустить после.

```
perl innobackup --apply-log /etc/mysql/my.cnf /home/cbell/backup mysqladmin
-uroot shutdown

perl innobackup --copy-file /etc/mysql/my.cnf /home/cbell/backup /etc/
init.d/mysql start
```

Дополнительные функции

InnoDB Hot Backup имеет поддержку PITR. Подробнее об этой и других функциях см. в документации приложения (*http://www.innodb.com/doc/hot_backup/manual.html*).

Доступна пробная версия InnoDB Hot Backup, действующая 30 дней, в течение которых можно использовать приложение без ограничений. Для получения бесплатной пробной версии посетите страницу *http://www.inn odb.com/products/hot-backup/order/order/*.

Физическое копирование файлов

Самый простой способ резервное копирования баз данных MySQL — обычное копирование файлов. К сожалению, для этого требуется остановка серверов. Чтобы выполнить копирование файлов, остановите сервер и скопи-

руйте каталог с данными и необходимые файлы с настройками. Часто это делают, создавая архив при помощи команды Unix tar. Полученный архив можно переместить в другую систему и восстановить каталог с данными.

Ниже приведена типичная последовательность команд tar для копирования данных с одного сервера баз данных и восстановления на другом. Выполните на сервере, данные с которого хотите скопировать, указанную команду, где *backup_2009_09_09.tar.gz* — имя файла, который будет создан, а */usr/loca/mysgl/data* — путь к каталогу с данными.

```
tar -czf backup_2009_09_09.tar.gz /usr/loca/mysql/data/*
```

Файл *backup_2009_09_09.tar.gz* должен находиться в каталоге, к которому имеют доступ оба сервера (или его необходимо физически скопировать на новый сервер). Затем, на сервере, где нужно восстановить данные, перейдите в корневой каталог нового экземпляра MySQL. Удалите существующий каталог с данными (если он существует) и выполните следующую команду:

```
tar -xvf ../backup_2009_09_09.tar.gz
```

Как говорилось ранее, файлам резервных копий следует давать понятные и содержательные имена.

Как видите из этого примера, копировать данные на уровне операционной системы очень просто. Можно не только получить единственный сжатый архивный файл, который легко перемещать, но и использовать преимущества быстрого копирования файлов. Можно даже выполнять выборочное резервное копирования, копируя отдельные файлы и подкаталоги.

К сожалению, команда tar доступна только на платформах Linux и Unix. Если В Windows эту команду можно использовать в том случае, если установлен пакет Cygwin, в который включена команда tar.

Если Cygwin или аналогичный пакет не установлен, ближайшим эквивалентом этой команды в Windows является функция архивных папок в Проводнике. Можно также использовать программы-архиваторы, такие как WinZip. Чтобы сжать каталог с данными, откройте Проводник Windows, найдите нужный каталог, но не открывайте его, а щелкните на нем правой кнопкой мыши и выберите команду для сжатия данных (Отправить | Сжатая ZIP-папка), а затем укажите имя файла .zip.

Физическое копирование файлов — самый быстрый и простой способ резервного копирования, но он требует выключения сервера. Однако это не обязательно, если внимательно следить за тем, чтобы во время копирования файла не выполнялось никаких обновлений. Для этого необходимо заблокировать все таблицы и выполнить команду flush tables, после чего отключить сервер и выполнить копирование.

Это аналогично процессу клонирования подчиненного сервера. Подробную информацию и пример клонирования с использованием копирования файлов см. в гл. 2.

Кроме того, в зависимости от объема данных, сервер должен быть отключен не только в время копирования, но и во время загрузки любых дополнительных данных, таких как записи из кэша, использования таблиц памяти для быстрого поиска и т. д. Поэтому физическое копирование может не подходить в некоторых случаях.

К счастью, есть сценарий Perl для автоматизации этого процесса, созданный Тимом Бунсом (Tim Bunce). Этот сценарий называется *mysqlhotcopy.sh* и находится в папке *./scripts* в каталоге MySQL. Он позволяет делать копии баз данных. Однако он работает только с механизмами хранилищ MyISAM и Archive в ОС Unix и Netware.

Утилита *mysqlhotcopy.sh* также предоставляет возможности настройки. Подробнее об этом см. по адресу *http://dev.mysql.com/doc/refman/5.4/en/mysqlhotcopy.html*.

Утилита mysqldump

Наиболее популярной альтернативой физическому копированию файлов является клиентское приложение mysqldump. С некоторых пор оно входит в состав MySQL, а первоначально было разработано Игорем Романенко. Это приложение создает набор команд SQL, воссоздающий базы данных при выполнении. Например, если запустить резервное копирование, выходные данные будут содержать все команды CREATE, необходимые для создания баз данных и содержащихся в них таблиц, а также все команды INSERT для вставки данных в эти таблицы.

Это очень удобно, когда нужно выполнить поиск и замену в тексте данных. Просто выполните резервное копирование базы данных, измените полученный файл в текстовом редакторе, а затем восстановите базу данных, чтобы применить изменения. Многие пользователи MySQL используют этот прием для исправления всевозможных ошибок, возникших в результате пакетного изменения данных. Это гораздо проще, чем, допустим, выполнять тысячу команд UPDATE со сложными условиями WHERE.

Недостатком приложения mysqldump является то, что оно требует гораздо больше времени, чем средства физического копирования, такие как InnoDB Hot Backup, LVM или просто копирование файлов, и гораздо больше дискового пространства. Вопрос времени может стать проблемой, если резервное копирование выполняется часто, нужно быстро восстановить базу данных после сбоя или требуется передать файл резервной копии по сети.

С помощью mysqldump можно копировать все базы данных, только некоторые из них или даже отдельные таблицы из указанной базы данных. Ниже приведены примеры каждого из этих вариантов:

```
mysqldump -uroot -all-databases
mysqldump -uroot db1, db2
mysqldump -uroot my_db t1
```

С помощью mysqldump можно также выполнять быстрое копирование таблиц InnoDB. Параметр --single-transaction выполняет команду BEGIN в начале резервного копирования, что заставляет механизм InnoDB читать таблицы в единой операции чтения. Таким образом, любые внесенные изменения применяются к таблицам, но данные остаются такими, каким были в момент начала операции копирования. Но при этом никакие другие подключения не должны использовать команды языка определения данных (DDL), такие как ALTER TABLE, DROP TABLE, RENAME TABLE и TRUNCATE TABLE, так как операция последовательного чтения не изолирована от изменений DDL.

Параметры --single-transaction и --lock-tables являются взаимоисключающими, так как команда LOCK TABLES выполняет явную фиксацию транзакции.

Эта утилита имеет несколько параметров, управляющих резервным копированием. В табл. 12-3 описаны наиболее важные из них. Полный набор параметров можно найти в документации по адресу *http://dev.mysql.com/doc/refman/5.4/en/mysqldumb.html*.

Табл. 12-3. Параметры mysqlbackup

Параметр	Функция
--add-drop-database	Добавляет команду DROP DATABASE перед каждой базой данных
--add-drop-table	Добавляет команду DROP TABLE перед каждой таблицей
--add-locks	Добавляет перед и после каждой добавленной таблицей команды LOCK TABLES и UNLOCK TABLES, соответственно
--all-databases	Включает все базы данных
--create-options	Включает в команды CREATE TABLE все параметры таблиц, специфичные для MySQL
--databases	Включает только список баз данных
--delete-master-logs	На главном сервере удаляет двоичные журналы после выполнения резервного копирования
--events	Выполняет резервное копирование событий из добавленных баз данных
--extended-insert	Использует альтернативный синтаксис команды INSERT, добавляющий каждую строку как выражение VALUES
--flush-logs	Очищает файлы журналов перед началом резервного копирования
--flush-privileges	Добавляет команду FLUSH PRIVILEGES после резервного копирования базы данных mysql

Табл. 12-3. (*окончание*)

Параметр	Функция
-ignore-table=db.tbl	Не выполняет резервное копирование указанной таблицы
--lock-all-tables	Блокирует все таблицы во всех базах данных во время выгрузки данных
--lock-tables	Блокирует все таблицы перед добавлением
-log-error=filename	Записывает предупреждения и ошибки в указанный файл
-master-data[=значение]	Добавляет в выходные данные имя файла и двоичного журнала и позицию
--no-data	Не записывает никакие данные из строк таблиц (только команды CREATE)
-password[=пароль]	Пароль для подключения к серверу
--port=номер_порта	Номер порта TCP/IP для подключения
-result-file=имя_файла	Записывает выходные данные в указанный файл
--routines	Добавляет хранимые процедуры и функции
--single-transaction	Выполняет команду BEGIN SQL перед выгрузкой данных с сервера. Это позволяет создавать последовательные мгновенные снимки таблиц InnoDB
--tables	Перекрывает параметр –databases
--triggers	Добавляет триггеры
-where='условие'	Добавляет только строки, удовлетворяющие условию
--xml	Создает выходные данные в формате XML

Эти параметры можно включить в файл конфигурации MySQL под заголовком [mysqldump]. В большинстве случаев можно указать параметры, просто удалив дефисы в начале. Например, чтобы всегда создавать выходные данные в формате XML, включите в файл конфигурации текст xml.

Одна из очень удобных функций mysqldump — возможность выгружать схему базы данных. Обычно для этого используется набор команд CREATE, воссоздающий все объекты без команд INSERT, добавляющих данные. Это может быть полезным для ведения записей об изменениях в схеме. Если использовать параметр --no-data совместно с параметрами для включения всех объектов (например, --routines и --triggers), то можно использовать mysqldump для создания схемы базы данных.

Обратите внимание на параметр --master-data. Этот параметр может быть очень полезным для выполнения PITR, так как сохраняет информацию о двоичном журнале, как это делает InnoDB Hot Backup.

Существует гораздо больше параметров для управления работой утилиты. Если вам подходит создание резервных копий в виде команд SQL,

изучите эти параметры, чтобы наиболее эффективно использовать mysqldump.

XtraBackup

Независимая компания Percona, занятая Open Source-разработкой и консалтингом по MySQL (LAMP), создала на основе InnoDB механизм БД XtraDB с открытым исходным кодом. В XtraDB внесен ряд усовершенствований для лучшего масштабирования на современном оборудовании и обратной совместимости с InnoDB.

В попытках создать решение для резервного копирования данных XtraDB, компания создала утилиту XtraBackup. Эта утилита оптимизирована для использования с InnoDB и XtraDB, но может работать также с таблицами MyISAM. Она предоставляет многие функции, требующиеся от решения для резервного копирования, включая сжатие и инкрементное копирование.

Исходный код XtraBackup можно получить по адресу *https://launchpad.net/percona-xtrabackup*. Скомпилировать и запустить XtraBackup можно на большинстве платформ. Эта утилита совместима с MySQL 5.0 и 5.1. Онлайн-руководство по XtraBackup расположено по адресу *http://www.percona.com/docs/wiki/per cona-xtrabackup:xtrabackup_manual*.

Мгновенные снимки LVM

Большинство систем Linux и некоторые Unix-системы предоставляют еще один способ резервного копирования баз данных MySQL, обладающий большими возможностями. В нем используется технология под названием *диспетчер логических томов* (logical volume manager, LVM).

В Windows имеется похожая технология под названием *теневое копирование тома* (Volume Shadow Copy). К сожалению, в отличие от LVM, она не имеет утилит для создания мгновенных снимков произвольных разделов и папок. Однако она позволяет делать мгновенные снимки тома целиком, что может быть полезным, если на этом томе хранится только каталог базы данных. Подробнее см. в документации Microsoft.

LVM —подсистема управления дисками, предоставляющая много административных возможностей для быстрого создания, удаления и изменения размера томов, позволяющая обойтись без использования старых, часто сложных и склонных к ошибкам дисковых утилит.

В плане резервного копирования LVM предоставляет возможность создания *мгновенного снимка*, т.е. копии активного тома, без прерывания работы приложений, обращающихся к данным этого тома. Идея состоит в том, что сначала создается мгновенный снимок (это относительно быстрая операция), а затем выполняется резервное копирование этого снимка, а не исходного тома. Внутри LVM мгновенный снимок обрабатывается при помощи механизма, следящего за изменениями, произошедшими с момента создания

снимка, так что сохраняются только измененные сегменты диска. Таким образом, мгновенный снимок занимает меньше места, чем полная копия тома, и когда выполняется резервное копирование, LVM копирует файлы такими, какими они были в момент создания мгновенного снимка. Можно сказать, что мгновенные снимки «замораживают» данные тома.

Еще одним преимуществом LVM и мгновенных снимков для резервного копирования баз данных является способ использования томов. Рекомендуется использовать отдельные тома для каждой установки MySQL, чтобы все данные находились на одном томе — это позволит быстро выполнять резервное копирование при помощи мгновенных снимков. Конечно, в некоторых ситуациях можно использовать несколько логических томов. Например, использовать по одному логическому тому для каждого табличного пространства или даже разные логически тома для таблиц MyISAM и InnoDB.

Начало работы с LVM

Если в вашей системе Linux не установлена подсистема LVM, можно установить ее, используя диспетчер пакетов. Например, чтобы установить LVM в Ubuntu, выполните следующую команду:

```
sudo apt-get install lvm2
```

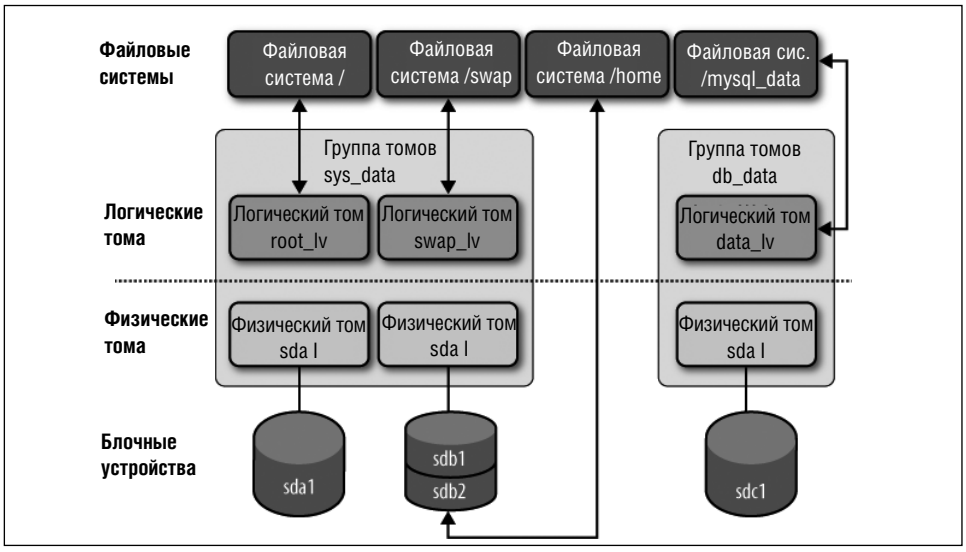


Рис. 12-1. Структура LVM

Не все системы LVM одинаковы, но описанная ниже процедура, основанная на типичном дистрибутиве Debian, нормально работает на таких системах, как Ubuntu. Мы не будем всесторонне описывать работу с LVM, а просто дадим представление о том, как можно использовать LVM для резервного копирования баз данных. Узнать о специфике типов LVM, поддерживаемых вашей системой, можно узнать из документации ОС или в Интернете.

Прежде чем углубиться в детали, рассмотрим базовые принципы LVM. В реализации LVM существует иерархия уровней. На самом низком уровне находится сам диск. Выше находятся *разделы*, позволяющие нам взаимодействовать с диском. Над разделом — *физический том*, который и является управляющим механизмом LVM. Физические тома можно включать в *группу томов* (в которой может быть несколько физических томов), а группа томов может содержать один или несколько *логических томов*. На рис. 12-1 показана связь между файловыми системами, группами томов, физическими томами и блочными устройствами.

Логический том может действовать как обычная смонтированная файловая система или как мгновенный снимок. Создание логического тома — мгновенного снимка является основой для использования мгновенных снимков для резервного копирования. В последующих разделах мы расскажем, как начать экспериментировать с LVM и создавать резервные копии данных.

Имеется несколько полезных команд, с которыми следует познакомиться. Ниже приведен список наиболее часто используемых команд и их возможностей. Более подробную информацию см. в документации.

- *pvcreate* — создает физический том;
- *pvscan* — показывает сведения о физических томах;
- *vgcreate* — создает группы томов;
- *vgscan* — показывает сведения о группах томов;
- *lvcreate* — создает логический том;
- *lvscan* — показывает сведения о логических томах;
- *lvremove* — удаляет логический том;
- *mount* — монтирует логический том;
- *unmount* — демонтирует логический том.

Чтобы использовать LVM, нужно иметь либо новый диск, либо диск, который можно логически демонтировать. Эта процедура выполняется следующим образом (выходные данные получены на лэптопе с Ubuntu 9.04):

1. Создайте резервную копию существующего каталога данных MySQL.

```
tar -czf ~/my_backups/backup.tar.gz /dev/mysql/datadir
```

2. Разбейте диск на разделы.

```
sudo parted
select /dev/sdb
mklabel msdos
mkpart test
quit
```

3. Создайте физический том для диска.

```
sudo pvcreate /dev/sdb
```

4. Создайте группу томов.

```
sudo vgcreate /dev/sdb mysql
```

5. Создайте логический том для данных. В этом примере мы создали том на 20 Гб.

```
sudo lvcreate -L20G -ndatadir mysql
```

6. Создайте на логическом томе файловую систему.

```
mke2fs /dev/mysql/datadir
```

7. Смонтируйте логический том.

```
sudo mkdir /mnt
sudo mount /dev/mysql/datadir /mnt
```

8. Скопируйте архив и восстановите данные на логический том.

```
sudo cp ~/my_backups/backup.tar.gz
sudo tar -xvf backup.tar.gz
```

9. Создайте экземпляр сервера MySQL и используйте параметр --datadir, чтобы указать папку на логическом томе.

```
./mysqld --console -uroot --datadir=/mnt
```


Если хотите поэкспериментировать с LVM, рекомендуем использовать диск, данные на котором не боитесь потерять. Удобное и дешевое решение — небольшой USB-диск.

Это все, что необходимо для начала работы с логическим томом. Поэкспериментируйте с утилитами LVM, чтобы приобрести навыки эффективной работы с ними, прежде чем применять их в рабочей среде.

LVM для резервного копирования и восстановления

Чтобы выполнить резервное копирование, потребуется очистить и временно заблокировать все таблицы, сделать мгновенный снимок, а затем разблокировать таблицы. Блокировка необходима, чтобы обеспечить завершение всех происходящих транзакций. Ниже описана процедура выполнения при помощи команд уровня оболочки:

1. На клиенте MySQL выполните команду FLUSH TABLES WITH READ LOCK.
2. Создайте мгновенный снимок логического тома (для этого используется параметр `-s`).

```
sudo lvcreate -L20M -s -n backup /dev/mysql/datadir
```

3. На клиенте MySQL выполните команду UNLOCK TABLES (теперь сервер может возобновить операции).
4. Смонтируйте мгновенный снимок.

```
sudo mkdir /mnts
sudo mount /dev/mysql/backup /mnts
```

5. Выполните резервное копирование мгновенного снимка.

```
tar -[FIXTHIS]f snapshot.tar.gz /mnts
```

Конечно, мгновенные снимки лучше периодически обновлять, чтобы можно было сделать новую резервную копию. Для автоматизации этого процесса существуют сценарии, написанные энтузиастами из Интернета, но лучше всего использовать проверенный способ удаления и повторного создания мгновенного снимка:

1. Демонтируйте мгновенный снимок.

```
sudo umount /mnts
```

2. Удалите мгновенный снимок (логический том).

```
sudo lvremove /dev/mysql/backup
```

После этого можно снова создать мгновенный снимок и выполнить резервное копирование. Если вы создадите собственный сценарий, рекомендуем удалять мгновенный снимок после того, как проверите, успешно ли была создана резервная копия, чтобы сценарий выполнял надлежащую очистку.

Если нужно восстановить мгновенный снимок, просто восстановите данные. Реальное преимущество LVM состоит в том, что все операции по созданию мгновенного снимка и резервной копии при помощи утилиты tar можно включить в сценарий, который затем периодически запускать (например, как задачу cron). Это поможет автоматизировать выполнение резервного копирования.

LVM в ZFS

Процедура выполнения резервного копирования в файловой системе ZFS компании Sun Microsystems (доступна в Solaris 10) очень похожа на процедуру LVM в Linux. Здесь мы приведем различия для пользователей Solaris.

В ZFS логические тома (которые Sun называет файловыми системами, если можно выполнять чтение и запись, и мгновенными снимками, если доступно только чтение) хранятся в пуле, аналогичном группе томов. Чтобы сделать копию или архив, просто создайте мгновенный снимок файловой системы.

Чтобы создать файловую систему ZFS, выполните следующую команду:

```
zpool create -f mypool c0d0s5
zfs create mypool/mydata
```

Для создания резервной копии (мгновенного снимка новой файловой системы) используйте следующую команду:

```
zfs snapshot mypool/mydata@backup_12_Dec_2009
```

Для восстановления файловой системы в указанном месте используйте следующие команды:

```
cd /mypool/mydata
zfs rollback mypool/mydata@backup_12_Dec_2009
```

ZFS позволяет выполнять не только копирование тома (файловой системы) целиком, но и поддерживает выборочное восстановление файлов. Подробнее о ZFS и выполнении в ней резервного копирования и восстановления см. по адресу *http: //dlc.sun.com/osol/docs/content/ZFSADMIN/gavvx.html*.

Сравнение способов резервного копирования

InnoDB Hot Backup, mysqldump и другие средства резервного копирования различаются в очень важных аспектах и имеют множество нюансов. Проведем сравнение способов резервного копирования по следующим параметрам: поддержка горячего резервного копирования, стоимость, скорость копирования, скорость восстановления, тип копирования (логическое или физическое), ограничения платформы (ОС) и поддерживаемые механизмы хранилищ. В табл. 12-4 приведены сведения для сравнения способов резервного копирования, описанных в этой главе.

Табл. 12-4. Сравнение способов резервного копирования

	InnoDB Hot Backup	mysqldump	Физическое копирование	Xtra-Backup	Мгновенные снимки LVM/ZFS
Поддержка горячего резервного копирования	Да (только InnoDB)	Да (только InnoDB, требуется параметр --single-transaction	Нет	Да (InnoDB и XtraDB)	Да (требует очистки и блокировки таблиц)
Стоимость	Платная лицензия	Бесплатно	Бесплатно	Бесплатно	Бесплатно
Скорость копирования	Средняя	Медленная	Быстрая	Средняя	Быстрая
Скорость восстановления	Быстрая	Медленная	Быстрая	Быстрая	Быстрая
Тип копирования	Физическое	Логическое	Физическое	Физическое	Физическое
ОС	Все	Все	Все	Все	Только ОС с поддержкой LVM
Механизмы	InnoDB, MyISAM	Все	Все	InnoDB, XtraDB, MyISAM	Все

В Windows InnoDB Hot Backup поддерживается не полностью. Сценарии Perl не запускаются в некоторых конфигурациях Windows. Подробнее см. в документации.

Во время планирования процедур восстановления данных можете использовать табл. 12-4 для поиска наиболее подходящих вам утилит. Например, если требуется выполнять горячее резервное копирование базы данных InnoDB, а стоимость не важна, лучшим выбором будет Innobase InnoDB Hot Backup. С другой стороны, если важна скорость, если требуется копировать базы данных, использующие различные механизмы хранилищ, и вы работаете в Linux, хорошим выбором будет LVM.

Резервное копирование и репликация MySQL

Существует два способа использования резервного копирования при помощи репликации MySQL. Из предыдущих глав вы узнали о репликации MySQL и способах ее использования для масштабирования и обеспечения высокой доступности. В этой главе мы рассмотрим еще два способа применения репликации MySQL, относящиеся к резервному копированию: использование репликации для создания резервной копии данных и использование резервных копий, полученных ранее для PITR.

- *Резервное копирование и восстановление* Часто в топологию добавляют дополнительный сервер для создания резервных копий. Это позволяет выполнять резервное копирование, не прерывая работу основного сервера, так как все операции, требующие отключения, можно выполнить на дополнительном сервере.
- *PITR* Даже если вы регулярно выполняете резервное копирование, может потребоваться восстановить сервер до определенного момента времени. Администрируя резервные копии должным образом, можно восстановить сервер до нужного момента с точностью до секунды. Это может быть очень полезным при восстановлении после человеческих ошибок, таких как неверный ввод команд и ввод некорректных данных, или для отмены изменений, которые больше не нужны. Возможности бесконечны, но они требуют существования соответствующих резервных копий.

Резервное копирование и восстановление с использованием репликации

Одним общим недостатком резервных копий является то, что они создаются в определенный момент времени (обычно ночью, чтобы не мешать выполнению других операций). А если возникает проблема, требующая восстановления главного сервера в том состоянии, в котором он был в другой момент, после создания резервной копии, то вам не повезло? Вовсе нет! Это не только возможно, но и довольно легко, если объединить резервные копии с двоичными журналами.

В двоичный журнал записываются все изменения, внесенные в базу данных во время ее работы, поэтому, восстановив резервную копию и воспроизведя двоичный журнал до нужной секунды, можно восстановить сервер до требуемого состояния.

Наиболее важный этап процедуры восстановления — это, конечно, восстановление. Поэтому мы подробно рассмотрим выполнение восстановления, прежде чем говорить о процедуре резервного копирования.

PITR

Чаще всего резервные копии в репликации используют для восстановления к состоянию в определенный момент времени (point-in-time recovery, PITR). Это дает возможность восстановить систему после ошибки (потери данных или аппаратного сбоя), вернув ее в состояние, ближайшее к последнему корректному состоянию, минимизировав потерю данных. Чтобы это сработало, необходимо иметь хотя бы одну резервную копию.

После восстановления сервера можно восстановить самую последнюю резервную копию и применить двоичный журнал, используя имя этого журнала и позицию в нем, как отправную точку.

Ниже описана процедура выполнения PITR с использованием системы резервного копирования.

1. Верните сервер в работоспособное состояние.
2. Найдите последнюю резервную копию баз данных, которые требуется восстановить.
3. Восстановите последнюю резервную копию.
4. Примените двоичный журнал с помощью утилиты mysqlbinlog, используя начальную позицию (или начальную дату и время), соответствующую последней резервной копии.

Сейчас вы можете подумать: «Какой двоичный журнал надо использовать для PITR-восстановления после резервного копирования?» Ответ зависит от того, как в последний раз выполнялось резервное копирование. Если вы очистили двоичный журнал перед запуском резервного копирования, нужно использовать имя и позицию текущего журнала (новый открытый файл журнала). Если очистка не выполнялась, используйте имя и позицию предыдущего журнала.


Для облегчения PITR всегда очищайте журналы перед резервным копированием. После этого отправной точкой будет начало файла.

Восстановление после репликации ошибки

Теперь посмотрим, как резервное копирование помогает в восстановлении случайных изменений в топологии репликации. Предположим, один из пользователей совершил катастрофическое (но допустимое) изменение, которое реплицировалось на все подчиненные серверы. Репликация тут не поможет, но резервные копии могут спасти положение.

Чтобы выполнить восстановление после изменений данных в репликации топологии, выполните следующее:

1. Удалите базы данных на главном сервере.
2. Остановите репликацию.
3. Восстановите последнюю резервную копию, полученную до происшествия.
4. Запишите текущую позицию в двоичном журнале главного сервера.
5. Восстановите последнюю резервную копию, полученную до происшествия на подчиненных серверах.
6. Выполните восстановление PITR на главном сервере, как описано в предыдущем разделе.
7. Снова запустите репликацию с записанной позиции и позвольте подчиненным серверам выполнить синхронизацию.

Короче говоря, хорошая стратегия резервного копирования — это не только необходимая защита от потери данных, но и важный инструмент репликации.

Пример восстановления

Теперь рассмотрим конкретный пример. Предположим, вы запускаете резервное копирование каждый день в 2 часа ночи и сохраняете полученные копии для дальнейшего использования. Допустим, что все двоичные журналы доступны. В действительности вы будете регулярно удалять файлы двоичных журналов, чтобы очистить место на диске, но в этом примере будем считать, что ни один из них не удален.

Перед вами поставлена задача восстановить базу данных на момент 2009-12-19 12:54:23, так как в это время любимые картинки вашего руководителя были случайно удалены чрезмерно усердным сотрудником, который посчитал, что просьба «почистить стол» относилась также и к рабочему столу на экране компьютера.

1. Найдите резервную копию, полученную до 2009-12-19 12:54:23.
2. В общем-то, нет разницы, какую именно копию вы возьмете, но чтобы сэкономить время, лучше выбрать ближайшую, которой будет копия, полученная в 2 часа ночи того дня.
3. Восстановите резервную копию, чтобы создать точную копию базы данных на момент 2009-12-19 02:00:00.
4. Найдите все файлы двоичного журнала, включающие весь диапазон изменений с 2009-12-19 02:00:00 по 2009-12-19 12:54:23. Неважно, есть ли там события, произошедшие до или после этого отрезка времени, главное, чтобы файлы двоичного журнала покрывали нужный диапазон.
5. Воспроизведите события из файлов журнала, используя утилиту mysqlbinlog, указав начальное время 2009-12-19 02:00:00 и конечное 2009-12-19 12:54:23.

После этого можно сказать руководителю, что его картинки спасены.

Чтобы автоматизировать эту процедуру, нужно немного позаниматься учетом. Это покажет вам, что нужно сохранить при выполнении резервного копирования, так что давайте просмотрим информацию, которая понадобится для восстановления.

- Чтобы правильно использовать резервные копии, важно помечать каждую из них, указывая начальное и конечное время, к которому относится эта копия. Это поможет определить нужную копию.
- Также потребуется позиция в двоичном журнале резервной копии. Это необходимо, так как время — недостаточно точный указатель на то, с какого места надо начинать воспроизведение событий из журнала.
- Также потребуется сохранить информацию о том, какой диапазон представляет каждый файл двоичного журнала. Строго говоря, это не обязательно, но может быть весьма полезным, так как помогает избежать обработки всех файлов журнала, соответствующих выбранной резервной копии. Такую операцию сервер MySQL не выполняет автоматически, поэтому ее придется проделать вручную.
- Эти файлы не будут храниться вечно, поэтому важно сортировать всю информацию, резервные копии и файлы двоичного журнала таким образом, чтобы их можно было легко архивировать, когда потребуется освободить дисковое пространство.

Образы восстановления

В администрировании резервных копий помогают *образы восстановления*. Образ восстановления — это виртуальный, а не физический контейнер. Он содержит только информацию о том, где находятся все необходимые элементы, позволяющие выполнить восстановление.

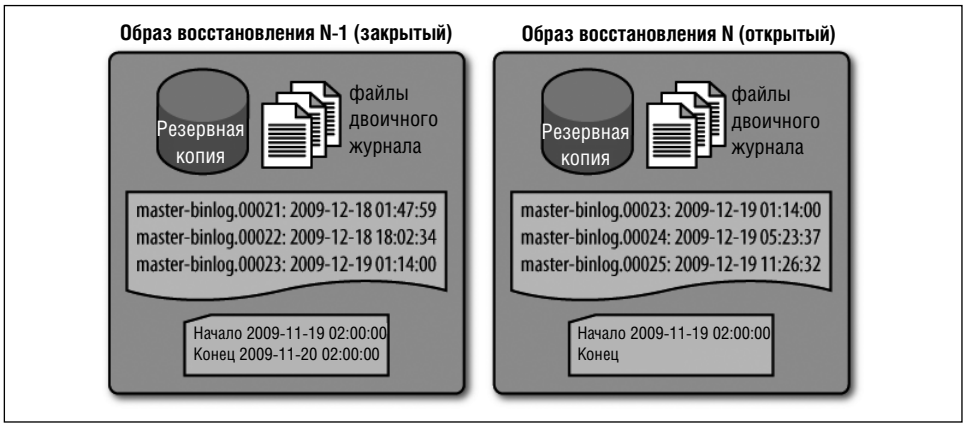


Рис. 12-2. Последовательность образов восстановления и их содержимого

На рис. 12-2 показана последовательность образов восстановления и содержимое каждого из них. Последний образ в последовательности особен-

ный, называется он *открытым образом восстановления*. Это образ восстановления, в который до сих пор добавляются изменения, и он еще не имеет конечного времени. Остальные образы являются *закрытыми образами восстановления* и имеют конечное время.

Каждый образ восстановления имеет некоторые фрагменты информации, необходимые для восстановления:

- *Резервная копия* — резервная копия, необходимая для восстановления базы данных.
- *Набор файлов двоичного журнала* — файлы двоичного журнала должны покрывать весь диапазон, соответствующий образу восстановления.
- *Время начала и, если есть, время конца* — начальное и конечное время промежутка, соответствующего образу восстановления. Если это открытый образ, он не имеет конечного времени. Такой образ нет смысла архивировать, поэтому для наших целей его конечным временем будет текущее время.

Файлы двоичного журнала обычно содержат события, выходящие за пределы диапазона, обозначенного начальным и конечным временем, но все события из этого диапазона должны присутствовать в образе восстановления.

- *Имя и начальное время для каждого файла двоичного журнала*. Чтобы извлечь нужные файлы двоичного журнала из архива, нужно знать их имена и их время начала и конца. Время начала можно извлечь из файла двоичного журнала при помощи mysqlbinlog.

Процедура резервного копирования

Процедура резервного копирования собирает всю необходимую информацию и структурирует ее так, чтобы мы могли использовать ее как для архивирования, так и для восстановления. Предположим, что в этой процедуре мы хотим создать образ восстановления *n* и имеем последовательность образов восстановления от образа 1 до образа n-1.

1. Создайте резервную копию любым способом и запишите ее имя и соответствующую ей позицию в двоичном журнале. Позиция в двоичном журнале также содержит имя файла двоичного журнала.

   Если для резервного копирования выполнялось отключение сервера, временем копирования следует считать время блокировки таблиц, а позицией журнала — позицию, полученную командой SHOW MASTER STATUS после блокировки базы данных.
2. Создайте новый открытый образ восстановления n, имеющий следующие параметры:
   - резервная копия образа n теперь является резервной копией с шага 1;
   - позиция образа n теперь является позицией с шага 1;
   - файлы двоичного журнала образа n неизвестны, но начинаются с имени файла из позиции с шага 1;

- начальное время образа n является начальным временем образа, определенным по событию из позиции двоичного журнала с шага 1.

3. Закройте образ n-1, заметив следующее:
   - файлы двоичного журнала образа n-1 теперь являются файлами двоичного журнала от позиции образа n-1 до позиции образа n;
   - конечное время образа n-1 теперь то же, что и начальное время образа n.

PITR в Python

Для управления различными типами способов резервного копирования мы создали класс PhysicalBackup, приведенный на лист. 12-1. Этот класс содержит два метода:

- *class PhysicalBackup.PhysicalBackup(image_name)* Конструктор класса принимает имя образа, который будет использоваться во время резервного копирования или восстановления сервера.
- *PhysicalBackup.backup_from(server)* Этот метод создает резервную копию сервера и сохраняет ее под именем образа, переданным конструктору.
- *PhysicalBackup.restore_on(server)* Этот метод использует образ резервной копии и восстанавливает его на сервере.

Используя класс для представления способа резервного копирования таким образом, легко заменить этот способ любым другим способом, если он имеет указанные методы.

Лист. 12-1. Класс, представляющий способ физического копирования

```
class BackupImage(object):
   «Class for representing a backup image»

   def __init__(self, backup_url):
      self.url = urlparse.urlparse(backup_url)

   def backup_server(self, server, db):
      «Backup databases from a server and add them to the backup image.»
   pass

   def restore_server(self, server):
      «Restore the databases in an image on the server»
      pass

class PhysicalBackup(BackupImage):
   «A physical backup of a database»

   def backup_server(self, server, db=»*»):
      datadir = server.fetch_config().get('datadir')
      if db == «*»:
         db = [d for d in os.listdir(datadir)
            if os.path.isdir(os.path.join(datadir, d))]
      server.sql(«FLUSH TABLES WITH READ LOCK»)
```

```
      position = replicant.fetch_master_pos(server)
      if server.host != «localhost»:
         path = basename(self.url.path)
   else:
      path = self.url.path
   server.ssh([«tar», «zpscf», path, «-C», datadir] + db)
   if server.host != «localhost»:
      subprocess.call([«scp», server.host + «:» + path, self.url.path])
   server.sql(«UNLOCK TABLES»)
   return position

def restore_server(self, server):
   if server.host == «localhost»:
      path = self.url.path
   else:
      path = basename(self.url.path)

   datadir = server.fetch_config().get('datadir')

   try:
      server.stop()
      if server.host != «localhost»:
         call([«scp», self.url.path, server.host + «:» + path])
      server.ssh([«tar», «zxf», path, «-C», datadir])
   finally:
      server.start()
```

Следующий этап — ввести представление образа восстановления, как показано на лист. 12-2. Образ восстановления хранит фрагменты данных в пяти полях:

- *RecoveryImage.backup_image* — резервная копия для использования.
- *RecoveryImage.start_time* — начальное время образа восстановления.
- *RecoveryImage.start_position* — позиция в файле двоичного журнала, соответствующая резервной копии. Используйте эту позицию вместо начального времени в качестве стартовой точки для воспроизведения событий из файлов двоичных журналов, так как это будет более точным. За одну секунду может выполниться множество транзакций, поэтому использовать начальное время нельзя, так как в качестве отправной точки будет выбрано первое событие, произошедшее в указанную секунду, тогда как в действительности начало должно быть другим.
- *RecoveryImage.binlog_files* — список файлов двоичного журнала, являющихся частью текущего образа восстановления.
- *RecoveryImage.binlog_datetime* — словарь сопоставлений имен файлов двоичного журнала с датой и временем (дата и время берется из первого события файла двоичного журнала).

Кроме этого, образ восстановления должен иметь следующие вспомогательные методы:

- *RecoveryImage.contains(datetime)* — определяет, содержится ли значение datetime в образе восстановления. Так как файл двоичного журнала может быть заменен в середине секунды, включено значение конечного времени (end_time).
- *RecoveryImage.backup_from(server)* — создает новый открытый образ восстановления, создавая резервную копию сервера (server), и собирает информацию о резервной копии.
- *RecoveryImage.restore_to(server, datetime)* — восстанавливает образ восстановления на сервере, так что применяются все изменения до момента времени datetime включительно. Это предполагает, что значение datetime находится в диапазоне образа восстановления. Если это значение меньше начального времени, ничего не будет применено, а если больше конечного времени, то применено будет все.

Лист. 12-2. Представление образа восстановления

```
class RecoveryImage(object):
   def __init__(self, backup_method):
      self.backup_method = backup_method
      self.backup_position = None
      self.start_time = None
      self.end_time = None
      self.binlog_files = [] self.binlog_datetime = {}

   def backup_from(self, server, datetime):
      self.backup_position = backup_method.backup_from(server)

   def restore_to(self, server):
      backup_method.restore_on(server)

   def contains(self, datetime):
      if self.end_time:
         return self.start_time <= datetime < self.end_time
      else:
         return self.start_time <= datetime
```

Так как для управления образами восстановления требуется манипулирование несколькими образами, мы ввели класс RecoveryImageManager, представленный на лист. 12-3. Этот класс содержит два метода в дополнение к конструктору:

- *RecoveryImageManager.point_in_time_recovery(server, datetime)* — метод, выполняющий восстановление PITR сервера server до момента времени datetime.
- *RecoveryImageManager.point_in_time_backup(server)* — метод, выполняющий резервное копирование сервера для восстановления PITR.

Диспетчер образов восстановления следит за всеми образами восстановления и используемыми способами резервного копирования. Здесь мы предполагаем, что для всех образов восстановления используется одинаковый способ резервного копирования, но это условие не обязательно.

Лист. 12-3. Класс RecoveryImageManager

```
class RecoveryImageManager(object):
   def __init__(self, backup_method):
      self.__images = []
      self.__backup_method = backup_method
   def point_in_time_recovery(server, datetime):
      from itertools import takewhile
      from subprocess import Popen, PIPE

      for im in images:
         if im.contains(datetime):
            image = im
            break
      image.restore_on(server)

      def before(file):
         return image.binlog_datetime(file) < datetime

      files = takewhile(before, image.binlog_files)
      command = ["mysqlbinlog",
               "--start-position=%s" % (image.backup_position.pos),
               "--stop-datetime=%s" % (datetime)]
      mysqlbinlog_proc = Popen(mysqlbinlog_command + files, stdout=PIPE)

      mysql_command = ["mysql",
            "--host=%s" % (server.host),
            "--user=%s" % (server.sql_user.name),
            "--password=%s" % (server.sql_user.password)]
      mysql_proc = Popen(mysql_command, stdin=mysqlbinlog_proc.stdout)
      output = mysql_proc.communicate()[0]
   def point_in_time_backup(self, server):
      new_image = RecoveryImage(self.__backup_method)
      new_image.backup_position = image.backup_from(server)
      new_image.start_time = event_datetime(new_image.backup_position)

      prev_image = self.__images[-1].binlog_files
      prev_image.binlog_files = binlog_range(prev_image.backup_position.
file,
                  new_image.backup_position.file)
      prev_image.end_time = new_image.start_time

      self.__images.append(new_image)
```

Автоматизация резервного копирования

Автоматизировать резервное копирование довольно легко. В предыдущем разделе мы показали, как выполнять резервное копирование и восстановление при помощи репликации. В этом разделе мы обобщим эту процедуру, чтобы ее можно было применить к копированию и восстановлению без использования репликации.

Единственная трудность, которая может возникнуть — обеспечение механизма для автоматического присваивания имен файлам резервных копий. Это можно делать по-разному. На лист. 12-4 показан метод, присваивающий файлам имена на основе времени копирования. Этот метод можно добавить в библиотеку Python в дополнение к методам репликации. Это та же библиотека, которую мы использовали в предыдущих главах.

Лист. 12-4. Сценарий резервного копирования

```
#!/usr/bin/python

import MySQLdb, optparse

#  --
#  Анализ аргументов и чтение конфигурации
#  --
parser = optparse.OptionParser()
parser.add_option(«-u», «--user», dest=»user»,
                  help=»User to connect to server with»)
parser.add_option(«-p», «--password», dest=»password»,
                  help=»Password to use when connecting to server»)
parser.add_option(«-d», «--database», dest=»database»,
                  help=»Database to connect to»)
(opts, args) = parser.parse_args()

if not opts.password or not opts.user or not opts.database:
   parser.error(«You have to supply user, password, and database»)

try:
   print «Connecting to server...» #
   #  Подключение к серверу
   #
   dbh = MySQLdb.connect(host=»localhost», port=3306,
                         unix_socket=»/tmp/mysql.sock»,
                         user=opts.user, passwd=opts.password,
                         db=opts.database)

   #
   #  Выполнение восстановления
   #
   from datetime import datetime
```

```
   filename = datetime.time().strftime(«backup_%Y-%m-%d_%H-%M-%S.bak»)
   dbh.cursor().execute(«[РЕЗЕРВНОЕ КОПИРОВАНИЕ]%s'» % filename)
   print «\nBACKUP complete.»

except MySQLdb.Error, (n, e):
   print 'CRITICAL: Connect failed with reason:', e
```

Вместо [РЕЗЕРВНОЕ КОПИРОВАНИЕ] подставьте исполняемую команду для выполнения резервного копирования.

Автоматизировать восстановление немного проще, так как не требуется создавать имя для копии. Однако в зависимости от типа установки, способа использования и конфигурации, могут потребоваться дополнительные команды, обеспечивающие отсутствие деструктивного взаимодействия с операциями других приложений и пользователей. На лист. 12-5 показан типичный метод восстановления, который можно добавить в библиотеку Python в дополнение к методам репликации.

Лист. 12-5. Сценарий восстановления

```
#!/usr/bin/python

import MySQLdb, optparse


#  --
#  Анализ аргументов и чтение конфигурации
#  --

parser = optparse.OptionParser()
parser.add_option(«-u», «--user», dest=»user»,
                  help=»User to connect to server with»)
parser.add_option(«-p», «--password», dest=»password»,
                  help=»Password to use when connecting to server»)
parser.add_option(«-d», «--database», dest=»database»,
                  help=»Database to connect to»)
(opts, args) = parser.parse_args()

if not opts.password or not opts.user or not opts.database:
   parser.error(«You have to supply user, password, and database»)

try:
   print «Connecting to server...»
   #
   #  Подключение к серверу
   #
   dbh = MySQLdb.connect(host=»localhost», port=3306,
                         unix_socket=»/tmp/mysql.sock»,
                         user=opts.user, passwd=opts.password,
                         db=opts.database)
```

```
#
#  Выполнение восстановления
#
from datetime import datetime

filename = datetime.time().strftime(«backup_%Y-%m-%d_%H-%M-%S.bak»)
dbh.cursor().execute(«[КОМАНДА ВОССТАНОВЛЕНИЯ]%S`»,
                     (database, filename))
   print «\nRestore complete.»
except MySQLdb.Error, (n, e):
   print `CRITICAL: Connect failed with reason:`, e
```

Вместо [КОМАНДА ВОССТАНОВЛЕНИЯ] подставьте свою исполняемую команду, выполняющую восстановление.

Как видно из лист. 12-5, восстановление можно автоматизировать. Однако многие предпочитают выполнять восстановление вручную. Автоматизированное восстановление может быть полезным в тестовом окружении, в котором требуется восстановить базу данных до известного состояния. Еще одно применение — в системах разработки, где для каждого проекта требуется подготовить определенное окружение.

Заключение

В этой главе мы поговорили о защите информации, подробно рассмотрев ее аспекты, наиболее важные для ИТ-специалиста. Мы показали важность планирования аварийного восстановления, рассказали, как создать собственный план восстановления, и выяснили, что СУБД являются важными составляющими восстановления. В завершение рассмотрены некоторые способы защиты данных MySQL при помощи резервного копирования.

В следующих главах мы рассмотрим более сложные темы, включая MySQL Enterprise, облачные вычисления и MySQL Cluster.

Джоэл глянул на окно терминала и ввел еще одну команду для проверки резервных копий баз данных. Он настроил сценарий для резервного копирования всех баз данных и был уверен, что эта простая операция сработает. Сожалел он лишь о том, что это только начало работы по созданию плана аварийного восстановления. Ему не терпелось начать эксперименты с другими сценариями и назначить первое совещание своей команды по планированию восстановления. Он уже наметил несколько кандидатур для нее, но быстрый стук в дверь отвлек его.

— Джоэл, ты уже заказал носители? Что там насчет плана? — спросил г-н Саммерсон.

Джоэл улыбнулся и ответил:

— Да, можно скопировать всю базу данных с помощью...

— Замечательно, Джоэл. Не надо подробностей, просто сделай так, чтобы аудиторы от нас отстали, хорошо?

Джоэл лишь улыбнулся и кивнул, когда его руководитель скрылся за дверью, чтобы озадачить очередного сотрудника. Интересно, подумал Джоэл, понимает ли г-н Саммерсон, какую работу надо проделать, чтобы решить поставленные им задачи... Он открыл почтовую программу и начал писать заявку на выделение дополнительного персонала и ресурсов.

ГЛАВА 13

MySQL Enterprise

Джоэл открыл еще одно окно терминала и начал читать вывод. Он тер глаза, но на экране все расплывалось, а числа сливались, когда он пытался следить за всеми удаленными серверами сразу. Он по-прежнему не успевал составить отчет...

Пока серверов было немного, он просматривал данные о них в электронных таблицах, но теперь, когда их стало больше 30, такая процедура оказалась слишком утомительной. Друзья советовали купить корпоративный пакет программ для мониторинга, но его босс не одобрял траты, не дающие немедленной отдачи.

— Эй, Джоэл!

Джоэл поднял глаза и увидел в дверях своего приятеля Дуга из отдела технической поддержки, стоящего с кружкой кофе.

— Привет, — устало ответил Джоэл.

— Ты выглядишь так, будто тебе пора отдохнуть.

— Нет времени. Мне надо составить отчет о состоянии всех серверов, а я не могу это сделать, пока полностью не проверю их.

— Ты совсем забыл об автоматизации?

— Вовсе нет, читал я об этих корпоративных пакетах, но не знаю, какой выбрать, да и одобрит ли Саммерсон...

— Ну, если бы ты мог доказать ему, какая куча времени у тебя уходит зря на всю эту рутину... — произнес Дуг, опасно жестикулируя своей кружкой.

Джоэл на мгновенье задумался.

— Если бы я только мог продемонстрировать ему экономию времени от использования хороших программ...

— Это хороший план. Так как насчет кружечки кофе? Я угощаю.

По пути Джоэл болтал с Дугом о концерте, на который они недавно ходили вместе. Вернувшись в офис, он начал читать о MySQL Enterprise от Oracle.

Мониторинг множества серверов требует серьезной работы. Инструменты, необходимые для надлежащего выполнения этой задачи, многочисленны. Большинство из них легко освоить, но все равно для этого требуются некоторые усилия. Облегчить работу с различными инструментами и сбор данных могут сценарии, но с ростом числа серверов сбор и анализ данных начинает занимать слишком много времени.

Приемы мониторинга, описанные в предыдущих главах, могут и не подойти для развертываний с большим числом серверов. В организации, где серверов несколько десятков, мониторингом и созданием отчетов, если это делается вручную, можно под завязку загрузить нескольких квалифицированных ИТ-специалистов.

В этой главе мы говорим о мониторинге, но то же самое справедливо и для администрирования. В частности, администраторы могут сэкономить время, используя инструменты, позволяющие автоматизировать обычные задачи, такие как заполнение и обслуживание таблиц и т. д.

К счастью, эта проблема не нова и вполне разрешима. Существует несколько пакетов для мониторинга предприятия, которые могут сделать жизнь в море серверов гораздо проще.

Один из самых недооцененных инструментов мониторинга MySQL — MySQL Enterprise Monitor (MEM), входящий в пакет MySQL Enterprise. Средства MEM значительно повышают возможности мониторинга и профилактического обслуживания и могут намного ускорить диагностику и уменьшить время простоя. Хотя это платная утилита, экономия на обслуживании центра обработки данных с избытком покрывает расходы.

В этой главе рассматривается пакет инструментов MySQL Enterprise и рассказывается, как эти инструменты помогают поддерживать производительность и доступность серверов MySQL на высочайшем уровне. Кроме того, мы приведем пример использования средств мониторинга в сложной топологии репликации.

Начинаем работу с MySQL Enterprise

Пакет MySQL Enterprise был выпущен в 2006 г. Он включает набор средств мониторинга и служб поддержки продукта. Этот новый пакет предназначен для специалистов, использующих MySQL для управления данными. Создатели MySQL осознали потребность организаций в стабильности и надежности, и пакет MySQL Enterprise стал ответом на эти запросы.

Если вы не готовы приобрести подписку на MySQL Enterprise или хотите попробовать поработать с этим пакетом, прежде чем примете решение, можете получить 30-дневную пробную версию. Пробную подписку можно оформить по адресу *http://mysql.com/trials/*.

В MySQL Enterprise добавлено веб-приложение MEM, а также отдельный экземпляр сервера MySQL, служащий хранилищем данных, собранных другими приложениями, установленными на серверах MySQL, которые называют *агентами*. На основе собранных данных MEM создает отчеты, которые можно улучшить эвристическими методами (их называют *советниками*), помогающими следовать в работе рекомендациям, основанным на исследованиях и опыте специалистов.

В следующих разделах описываются доступные возможности службы подписки MySQL Enterprise и приводится обзор процесса установки. Затем мы подробнее рассмотрим предоставляемые возможности и преимущества.

Варианты подписки

Пакет MySQL Enterprise можно приобрести в одном или нескольких вариантах: базовом, серебряном, золотом и платиновом. Базовый пакет самый дешевый и содержит меньше всего инструментов. Каждый следующий вариант предоставляет все больше возможностей. Таким образом, можно выбрать вариант, наиболее подходящий для вашего бюджета и потребностей в повышении доступности серверов.

- **Базовый** Этот вариант включает сервер MySQL версии Pro. Он предоставляет базовое обслуживание сервера, включающее обновления ПО и два обращения за поддержкой. Срок между обращениями и время ответа — два дня. Также предоставляется доступ к обширной базе знаний MySQL для поиска информации о распространенных проблемах и их решениях. Средства мониторинга не включены. Базовый вариант лучше всего подходит для организаций, имеющих потребность в ПО для работы, но не нуждающихся в сложных средствах мониторинга и немедленном ответе на обращение в техподдержку.
- **Серебряный** Включает то же, что и базовый, плюс средства мониторинга с ограниченным набором советников по администрированию и обновлению. Включена возможность телефонной поддержки с неограниченным числом обращений и интервалом повторных обращений 4 часа. Этот вариант хорошо подходит для большинства организаций, использующих решения на основе MySQL и желающих обеспечить корректную настройку своих серверов.
- **Золотой** Содержит сервер MySQL версии Advanced (поддерживающий разделы) и включает все возможности серебряного пакета плюс дополнительные советники для мониторинга репликации и памяти. Кроме того, включен анализатор запросов MySQL для монитора. Добавлена консультационная поддержка, охватывающая репликацию и разделы. Время ответа на обращения — два часа, с возможностью срочного ответа. Это отличный вариант для организаций, использующих репликацию MySQL.
- **Платиновый** Включает все возможности золотого пакета, всех советников, расширенную консультационную поддержку по всем областям и круглосуточную телефонную поддержку; время ответа — два часа, с возможностью срочного ответа в течение 30 минут. Этот вариант даже позволяет иметь собственную сборку сервера MySQL. Платиновый пакет предназначен для организаций, требующих наивысшего уровня поддержки критичных решений по управлению данными.

Предлагаемые варианты весьма разнообразны, и с ростом организации можно переходить на более подходящие варианты. Более подробную ин-

формацию об этих пакетах и их стоимости см. по адресу *http://mysql.com/products/enterprise/features.html*.

Обзор установки

Приобретая подписку на MySQL Enterprise, вы получаете ключ продукта и учетные данные для входа на портал MySQL Enterprise. Подключение к этому порталу необходимо для активации инструментов MySQL Enterprise. Портал содержит все необходимое: обновления, новости, сведения о подписке и доступ к базе знаний.

Этот портал называется центром клиентов MySQL Enterprise (MySQL Enterprise Customer Center) и находится по адресу *https://enterprise.mysql.com*.

Для установки MySQL на серверы, не подключенные к сети или не имеющие доступа к Интернету, можно скачать ключ продукта с портала в виде файла и использовать его во время установки. Также потребуется сопроводительный файл *.jar*. Оба эти файла можно скачать с портала.

Сначала посетите портал и скачайте файлы установки средств Enterprise для ваших платформ, соответствующий выпуск сервера MySQL, руководство по началу работы и документацию для пакета Enterprise. Руководство по началу работы содержит инструкции по установке и настройке средств Enterprise на платформах Mac OS X, Linux и Windows.

Мы не будем приводить полные подробные инструкции, а вкратце опишем этапы установки и настройки пакета MySQL Enterprise. Для установки необходимо пройти как минимум следующие этапы:

1. Установите компонент для управления службами, включая хранилище показателей (отдельная установка MySQL). Эта процедура различается для разных ОС. Например, в Mac OS X устанавливается файл *mysqlmonitor-2.1.0.1096-osx-installer*.
2. Активируйте подписку и включите инструментальную панель Enterprise Dashboard. На этом этапе требуется ввести ключ продукта, либо предоставив файл, либо зайдя на портал MySQL Enterprise.
3. Установите агент мониторинга на серверы MySQL. Эта задача также различается для разных ОС. Например, в Linux устанавливается агент *mysqlmonitor-agent-2.1.0.1093.linux-glibc2.3-x86-32bit-installer.bin*.
4. Настройте инструментальную панель в соответствии с вашим окружением.

Хотя можно установить агент мониторинга на любой сервер, с которого потом выполнять мониторинг других серверов, лучше всего установить агент на тот сервер, мониторинг которого будет выполняться. Это позволяет агенту передавать инструментальной панели сведения об ОС, данные производительности и параметры настройки. Если установить агент на другой сервер, секция «Система» в отчете инструментальной панели будет пустой.

Можно настроить агента, чтобы он сообщал сведения о нескольких серверах или передавал сведения нескольким мониторам. Первая возможность позволяет использовать один агент для мониторинга двух и более экземпляров сервера MySQL с одной системы. Вторая позволяет нескольким инструментальным панелям получать данные от MEM. Обе эти возможности описаны в документации по MEM.

Установка пакета MEM включает независимый веб-сервер и экземпляр MySQL, которые устанавливаются на той системе, где будет размещена инструментальная панель и хранилище показателей. Эта система будет принимать данные от каждого агента мониторинга. Эта процедура очень просто выполняется и не требует какого-либо опыта в веб-администрировании.

Во время установки пакета мониторинга необходимо выбрать имена для нескольких учетных записей и записать расположение (например, IP-адрес) сервера. Эта информация потребуется при установке агентов мониторинга. В руководстве по началу работы говорится обо всех этих элементах. Особо обратите внимание на примеры экранов из этого руководства. Можете распечатать это руководство и использовать его для записи сведений, которые необходимо указать во время установки.

Установка агентов мониторинга тоже очень проста. После установки и запуска сервера мониторинга и подтверждения ключа продукта можно установить по одному агенту на каждом сервере MySQL в сети. В некоторых системах может потребоваться запуск агента вручную, о чем подробно рассказывается в руководстве.

После того как установлен и запущен минимум один агент, можно вернуться к инструментальной панели и начать ее настройку в соответствии с собственными потребностями.

Компоненты MySQL Enterprise

Пакет MySQL Enterprise состоит из приложений сервера MySQL, набора инструментов MEM и средств поддержки продукта.

Сервер MySQL Enterprise

Подписка MySQL Enterprise включает две версии сервера MySQL. Версия Pro является рабочим выпуском с наиболее стабильным набором функций. Версия Advanced содержит некоторые экспериментальные функции, такие как горизонтальные таблицы и разбиение индексов на разделы, позволяющие повысить производительность работы очень крупных баз данных.

MEM

MEM — это механизм непрерывного мониторинга сервера и отправки оповещений. На веб-сайте MySQL об этом очень хорошо сказано: «Как будто вам

помогает виртуальный администратор баз данных, дающий рекомендации по устранению уязвимостей системы безопасности, улучшению репликации, оптимизации производительности и многому другому». Что бы там ни говорили маркетологи, MEM предоставляет профессиональные инструменты, позволяющие удовлетворять растущие потребности ЦОД организации.

Перечислим ключевые особенности MEM:

- единый экран для мониторинга работоспособности всех серверов;
- более 600 показателей, включая сервер MySQL и ОС, в которой он установлен;
- возможность мониторинга производительности, репликации, схемы и безопасности;
- немедленное отображение сведений о работоспособности системы на графике;
- уведомление о нарушениях пороговых значений показателей;
- реализация рекомендаций и наборов правил от создателей MySQL.

Инструменты MEM представлены распределенным веб-приложением, работающим во внутренней сети. На каждый сервер MySQL устанавливается агент мониторинга, который отправляет показатели компоненту веб-сервера, называемому инструментальной панелью Enterprise (Enterprise Dashboard). Этот компонент отображает все сведения и графики, отражающие состояние серверов. Также имеются советники, реализующие рекомендации по обеспечению корректной настройки и максимально эффективной работы серверов.

Enterprise Dashboard

Лицом MySQL Enterprise является инструментальная панель Enterprise Dashboard, веб-приложение, работающее на сервере мониторинга. Это приложение предоставляет единое расположение для мониторинга всех серверов по отдельности или группами. Можно просматривать данные о доступности, безопасности и производительности всех серверов в одном месте. Можно проверять относительную работоспособность каждого сервера, просматривать графики производительности и использования памяти и наблюдать за ключевыми показателями операционной системы.

Enterprise Dashboard представляет данные мониторинга и оповещений в легко читаемом виде. На рис. 13-1 показан пример для простой установки.

Как видите, вся ключевая информация показана на одном экране. Имеются вкладки для советников, событий, дополнительных графиков производительности, анализатора запросов, репликации и параметров настройки. Вкладка What's New? содержит ссылки на новости и события, связанные с инструментами и вашим вариантом подписки Enterprise.

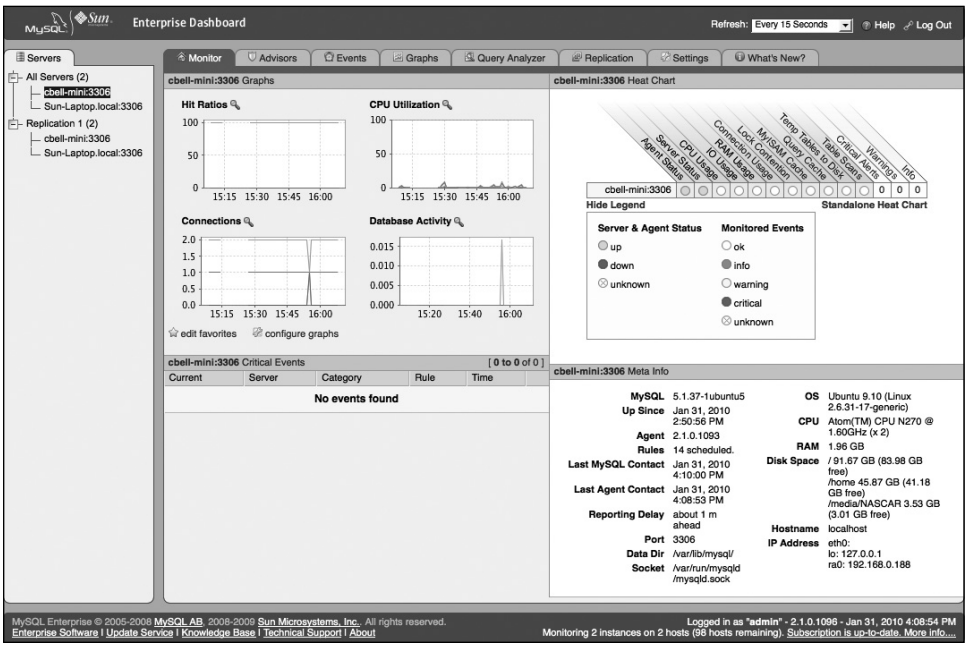


Рис. 13-1. MySQL Enterprise Dashboard

Агент мониторинга

Агент мониторинга — это специальное небольшое приложение, собирающее информацию о сервере MySQL, включая сведения об операционной системе, и являющееся ключевым компонентом для выполнения мониторинга. Это приложение, предназначенное для установки на каждый сервер, за которым нужно наблюдать, работает почти прозрачно, без какого-либо заметного влияния на производительность системы.

Советники

Среди инструментов MySQL Enterprise есть функция, отличающаяся от обычных решений мониторинга. Это механизм, выполняющий мониторинг определенных аспектов производительности и конфигурации системы и отправляющий оповещения, когда сервер отклоняется от стандартов, определенных проектировщиками MySQL. Это значит, что вы можете незамедлительно получать обратную связь по любым действиям, делающим конфигурацию, безопасность и производительность неоптимальными. Такой механизм называется советником. Существует много советников, выполняющих мониторинг различных областей:

- *администрирование* — мониторинг администрирования и производительности в целом;
- *обновление* — мониторинг условий обновления и отправка оповещений о потенциальных проблемах с ошибками, присущими определенным вер-

сиям. Также предлагаются стратегии обновления для исправления ошибок, связанных с проблемами обновления;

- *производительность* — определение различий в производительности на основе правил и рекомендаций разработчиков MySQL;
- *схема* — определение изменений в базе данных и объектах схемы. Можно настроить мониторинг изменений и отправку оповещений при возникновении нежелательных или неожиданных изменений;
- *использование памяти* — определение изменений в использовании памяти и отправка оповещений при возникновении неоптимальных условий;
- *безопасность* — определение и потенциальных уязвимостей в системе безопасности и отправка оповещений;
- *репликация* — определение условий репликации, связанных с вопросами конфигурации, работоспособности, синхронизации (задержек) и производительности;
- *пользовательские* — можно создавать своих советников для поддержки собственных стандартов.

Каждый советник полностью охватывает определенный аспект работы сервера, используя набор правил, основанных на рекомендациях производителей. Советники помогают определить, в каких областях серверы требуют вашего внимания, и дают советы по улучшению или исправлению ситуации. Если возможностей советников вам мало, можете создать собственных советников, удовлетворяющих ваши потребности.

Анализатор запросов (золотой и платиновый пакеты)

Сложные базы данных и приложения часто требуют выполнения сложных запросов. Язык SQL предоставляет широкие возможности, но часто запросы пишутся так, что выполняются не так эффективно, как могли бы. Более того, плохо написанные запросы могут быть причиной плохой производительности. Опытные администраторы БД знают об этом и во время решения проблем с производительностью базы данных часто в первую очередь проверяют запросы.

Обычно найти некорректный запрос можно в журнале медленных запросов или списке процессов (т.е., выполнив команду SHOW PROCESSLIST). Найдя запрос, требующий внимания, можно использовать команду EXPLAIN для выяснения того, как выполняется этот запрос. Эта процедура хорошо известна и часто применяется, но она требует времени и ее нелегко автоматизировать с помощью сценариев. Чем сложнее база данных, тем больше усилий требуется для выявления некорректных запросов.

Таким способом можно проверять пользовательские запросы, но что делать с приложениями, у которых команды SQL включены в код? Такие случаи — одни из самых трудных для диагностики и исправления, и требуют изменения приложения. Если допустить, что это возможно, что вы посоветуете разработчикам для улучшения запросов?

К сожалению, команды SQL, включенные в код приложений, редко принимают во внимание при поиске причин проблем с производительностью. Администраторы БД и разработчики склонны слишком быстро обвинять во всем сервер или ОС, а не код приложения или встроенные запросы SQL. Что еще хуже, MySQL не поддерживает надежные наборы показателей производительности и предоставляет мало помощи в поиске проблемных запросов.

Не было бы лучше видеть список всех долго выполняющихся на сервере запросов и иметь возможность проверять самые медленные из них? Это можно делать при помощи анализатора запросов (Query Analyzer), входящего в набор инструментов Enterprise Monitor.

Включить анализатор запросов можно через панель Enterprise Dashboard. Установка и настройка требует немного работы, но если возникнут вопросы, обратитесь к руководству Getting Started.

Анализатор запросов показывает совокупные сведения о производительности запросов в реальном времени. Все запросы со всех серверов показываются в одном месте, так что не требуется ходить от сервера к серверу в поисках плохо работающих запросов. Этот же список содержит историю запросов, так что не нужно беспокоиться о дополнительном пространстве для журналов.

Можно получить два разных представления каждого запроса: каноническое представление (без числовых данных), позволяющее просмотреть графическую версию запроса, и представление с информацией о времени и данными, к которым обращается запрос. Советник может даже сообщать о том, когда и на каком сервере выполнен определенный запрос.

Анализатор запросов позволяет увидеть, какие запросы имеют низкую производительность, и найти решение, проверив план выполнения запроса. Очевидно, что одна только эта функция может сэкономить много времени, особенно когда требуется разработать или настроить приложение для развертывания или оптимизировать запросы для получения более высокой производительности.

Поддержка MySQL

Подписка MySQL Enterprise включает доступ к службе поддержки, которая может помочь в разработке, развертывании и управлении серверами MySQL. Поддержка включает решение проблем, консультирование, доступ к онлайновой базе знаний, содержащей распространенные решения, а для платинового пакета еще и персонального технического менеджера, который будет вашим контактным лицом, помогающим в получении поддержки.

Использование MySQL Enterprise

Мы рассмотрели функции и компоненты MySQL Enterprise, а теперь приведем пример того, как эти инструменты могут принести пользу организации.

В этом примере мы будем использовать гипотетическую (хотя и часто встречающуюся) инфраструктуру компании, основанную на веб-технологиях. Здесь хорошо представлены модели репликации, используемые в индустрии. Предлагаемая инфраструктура включает репликацию с несколькими главными серверами и репликацию наборов, при которой СУБД реплицируют только отдельные базы данных.

Инфраструктура (рис. 13-2) включает дата-центр с двумя главными серверами, обеспечивающими высокую доступность и балансировку нагрузки. На этих серверах размещены базы данных компании, каждая из которых предназначена для отдельной линейки продуктов. К главному дата-центру подключен подчиненный сервер для выполнения внутренних задач и повседневных операций. Этот сервер также имеет подчиненные серверы для различных отделов, включая сервер для сторонних организаций, на котором работают службы независимой верификации и проверки (IVV). К тому же главному серверу подключен еще один подчиненный сервер для отделов разработки, отвечающий за построение и улучшение линеек продуктов.

Каждый из подчиненных серверов может иметь (и обычно имеет) дополнительные нереплицируемые базы данных. Например, на производственном сервере обычно размещаются базы данных персонала, которая не реплицируется на большинство подчиненных серверов (например, она не реплицируется в центр разработки). Точно так же, сервер для сторонних компаний имеет собственную базу данных результатов, а сервер разработки имеет различные версии баз данных для линеек продуктов, находящихся на разных стадиях разработки.

Установка

Установка MySQL Enterprise включает настройку серверов баз данных для запуска последнего выпуска Pro или Advanced. Можно использовать существующие установки серверов MySQL, но для максимального возврата инвестиций следует использовать версии, поставляемые по подписке MySQL Enterprise. Если серверы БД настроены и нормально работают, можно приступать к установке Enterprise Monitor и агентов мониторинга.

Начать следует с установки MEM на компьютер в сети, с которого можно подключиться ко всем серверам, мониторинг которых будет выполняться (мы рекомендуем всегда использовать MEM для мониторинга серверов MySQL). В процессе установки запишите имя хоста или IP-адрес этого сервера, а также имя пользователя и пароль, указанные для доступа агента. Процедура установки очень проста и подробно описана на портале подписки Enterprise.

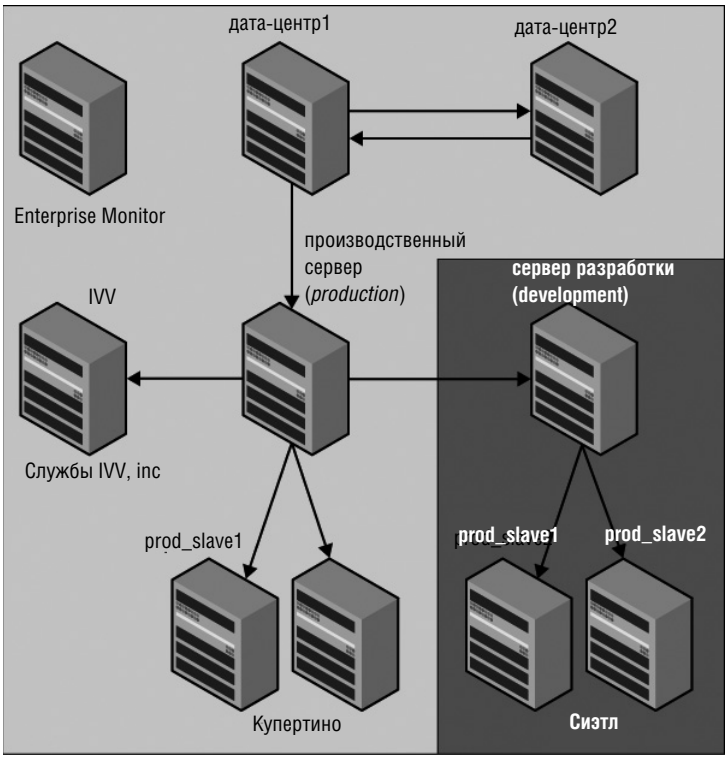


Рис. 13-2. Пример информационной инфраструктуры

Для установки MySQL Enterprise требуется несколько учетных записей пользователей. Кроме учетной записи, предоставленной по подписке Enterprise, понадобится учетная запись администратора MEM, учетная запись агента для доступа к серверу MEM и учетная запись для агента мониторинга на каждом сервере MySQL. Если перепутать эти учетные записи, установка может не удастся.

Установив и запустив MEM на сервере мониторинга, можно приступать к установке агента мониторинга на каждом из серверов MySQL. После установки агента мониторинга потребуется указать учетную запись и пароль для подключения этого агента к серверу MySQL. Лучше использовать одинаковые имена и пароли на всех серверах, но при этом не забывать, что это разные учетные записи, которые нужно создать на всех серверах по отдельности. Права этой учетной записи предоставьте следующим образом:

```
GRANT SELECT, REPLICATION CLIENT, SHOW DATABASES, SUPER, PROCESS ON *.*
TO 'имя_пользователя'@'localhost' IDENTIFIED BY 'пароль_агента';
```

Создав учетную запись и предоставив ей привилегии для доступа к серверу, запустите агента мониторинга и понаблюдайте за MEM. Сервер должен показаться в MEM через несколько секунд, в зависимости от параметров обновления.

Процедура установки MEM и агента мониторинга немного различается на разных платформах. За подробностями обратитесь к документации.

Повторите установку на каждом из серверов и понаблюдайте за результатами в Enterprise Dashboard. На рис. 13-3 показано окно Enterprise Dashboard для указанной инфраструктуры с отчетами от всех агентов мониторинга.

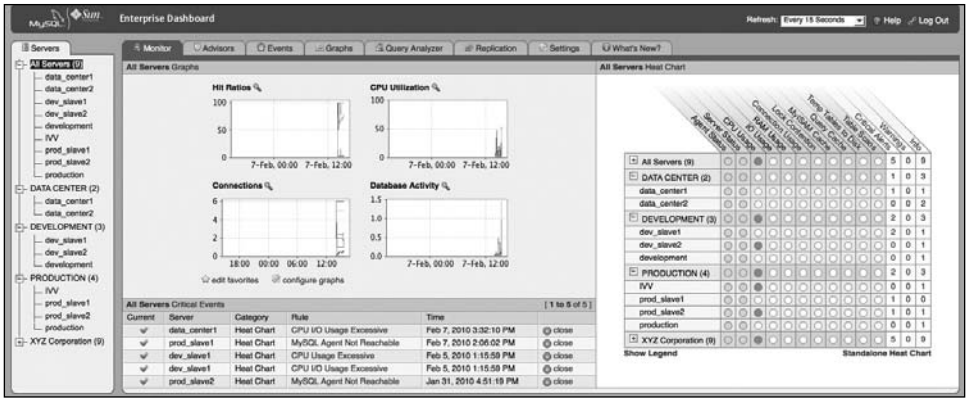


Рис. 13-3. Enterprise Dashboard

Каждый из серверов сообщает о состоянии сервера и агента. На диаграмме справа эти данные представлены зелеными точками (зеленый цвет — это хорошо). В центре находятся четыре составных графика, показывающие частоту обращения к кэшу запросов, сведения об использовании ЦП, подключении и активности баз данных. Ниже приведен список стандартных критических событий, о которых сообщают советники. На приведенном рисунке видны предупреждения о ЦП, вводе-выводе и использовании.

В этом окне показана вся необходимая информация о серверах, на основе которой можно быстро оценить общую работоспособность инфраструктуры. Дальше облегчить мониторинг уже просто некуда!

Решение проблем с агентом мониторинга

Хотя процедура установки агента мониторинга очень проста, иногда возникают проблемы. Если вводить корректную информацию, все должно работать нормально.

Если агент мониторинга не передает информацию MEM, попробуйте выполнить следующее:

- если агент мониторинга запустился, но сервер не отображается в Enterprise Dashboard, или если для агента или службы на диаграмме указано состояние ошибки (красная точка), проверьте файл *mysql-monitor-agent.log*. Этот файл содержит много информации, помогающей решить большинство проблем. Файлы журналов находятся в следующих расположениях:

  - Mas OS X: */Applications/mysql/enterprise/agent*.
  - Linux/Unix: */opt/mysql/enterprise/agent on Linux*.
  - Windows: *C:\Program Files\MySQL\Enterprise\Agent*.
- проверьте учетную запись и разрешения, указанные для агента мониторинга на сервере MySQL;
- проверьте учетную запись и пароль в файле *agent-instance.ini*. Убедитесь, что указаны те же учетные данные, что и на сервере MySQL, мониторинг которого выполнятся;
- проверьте порт и имя хоста локального сервера MySQL. Убедитесь, что эта информация соответствует тому, что записано в файле *agent-instance.ini*;
- проверьте, какой порт указан для сервера в файле *agent-instance.ini*. Убедитесь, что можете зайти на локальный сервер MySQL, используя порт, имя пользователя и пароль, указанные в этом файле;
- проверьте имя хоста, имя пользователя и пароль для сервера MEM в файле *mysql-monitor-agent.ini*. Убедитесь, что сервер MEM отвечает на команду ping.

Если проблемы возникли в работе анализатора запросов, проверьте также порт прокси-сервера в файле *mysql-monitor-agent.ini* и убедитесь, что можете подключить клиента MySQL к прокси-серверу, используя информацию, указанную в этом файле.

Мониторинг

Есть несколько областей, мониторинг которых в сложной инфраструктуре гораздо проще выполнять при помощи MySQL Enterprise:

- диаграмма работоспособности;
- сведения об оповещениях;
- составные графики серверов;
- сведения о сервере;
- сведения о репликации;
- советники.

В следующих разделах мы подробнее рассмотрим каждую из этих областей.

Серверы можно переименовывать при помощи команды Manage Servers на странице Settings. Это позволяет использовать более понятные имена в окне Enterprise Dashboard, оставляя настоящие имена серверов без изменения.

Кроме того, можно создавать группы, объединяя связанные серверы. Это очень удобно, так как группа отображается в различных элементах управления, что позволяет сворачивать ее и изменять способ отображения.

Диаграмма работоспособности

Как было показано ранее, диаграмма, расположенная в правой части страницы мониторинга в окне Enterprise Dashboard, позволяет быстро получить представление о работоспособности серверов. Легенда (которую можно включать и отключать) показывает наборы цветов, обозначающих состояния операций от полностью работоспособного (зеленый) до отключенного и не отвечающего (красный). Это позволяет быстро увидеть, какие области требуют дальнейшей проверки. На рис. 13-4 показан пример диаграммы для описанной выше инфраструктуры.

Рис. 13-4. Диаграмма работоспособности

Обратите внимание на то, что в заголовке диаграммы перечислены категории, представляющие критические области мониторинга. Эти области охватывают аспекты, описанные в главах 8-10 (ЦП, память и т.д.). В отличие от сведений, получаемых при выполнении мониторинга вручную, на этой диаграмме показаны относительные показатели работоспособности, позволяющие быстро оценить состояние.

Кроме общих аспектов, представлены аспекты, специфичные для MySQL, такие как конфликты при блокировках, использование кэша MyISAM, использование кэша запросов и число сканирований таблиц. При помощи MySQL Enterprise можно получать гораздо больше информации, как вы увидите далее, но за этими аспектами наблюдают чаще всего.

Справа от списка категорий находятся столбцы, содержащие число недавно произошедших критических событий, предупреждений и сообщений.

Значения на диаграмме периодически обновляются и могут увеличиваться или уменьшаться, в зависимости от времени возникновения и решения проблем.

Сведения об оповещениях

Одна из лучших функций диаграммы, которая может не быть очевидной, — возможность щелкнуть на любой из точек или чисел, чтобы получить дополнительную информацию. Например, если щелкнуть на точке в столбце I/O usage (использование ввода-вывода) для сервера, на котором возникли проблемы ввода-вывода, появится полный список оповещений для этой системы (рис. 13-5). После этого можно щелкнуть на самом недавнем оповещении и получить подробный отчет о нем (рис. 13-6).

All Servers Critical Events [**1 to 5** of 5]

Current	Server	Category	Rule	Time	
✓	data_center1	Heat Chart	CPU I/O Usage Excessive	Feb 7, 2010 3:32:10 PM	close
✓	prod_slave1	Heat Chart	MySQL Agent Not Reachable	Feb 7, 2010 2:06:02 PM	close
✓	dev_slave1	Heat Chart	CPU Usage Excessive	Feb 5, 2010 1:15:59 PM	close
✓	dev_slave1	Heat Chart	CPU I/O Usage Excessive	Feb 5, 2010 1:15:59 PM	close
✓	prod_slave2	Heat Chart	MySQL Agent Not Reachable	Jan 31, 2010 4:51:19 PM	close

Рис. 13-5. Пример списка оповещений

В этом отчете указан сервер, от которого пришло оповещение, время события и рекомендации по дальнейшим действиям. Вкладки в верхней части позволяют закрыть оповещение, убрав его с экрана (что можно сделать, приняв какое-либо решение относительно ситуации), просмотреть дополнительные сведения (расширенное описание проблемы) и увидеть, как было запущено оповещение.

Results | Close Event | Details | Advanced

INFO Alert - CPU Usage Excessive (v 1.7 *)

Server
dev_slave2

Time
Feb 7, 2010 3:31:13 PM (17 minutes ago)

Status
Open

Advice
Use whatever system tools are available to you on **dev_slave2** (e.g. vmstat, perfmon, top, Task Manager, etc.) to investigate how and why the CPU is overloaded, so you can determine the appropriate action to take to improve the situation. The cpu_idle time on **dev_slave2** is low relative to cpu_sys (kernel), cpu_user (non-kernel), and cpu_wait (waiting on I/O), indicating excessive CPU usage.

Recommended Action
None specified.

Notifications
No notifications set.

hide

expand »

Рис. 13-6. Пример отчета об оповещении

Отчеты об оповещениях выделяют MySQL Enterprise среди других инструментов для мониторинга. Именно это имеют в виду, когда говорят о «виртуальном администраторе баз данных». Оповещения значительно об-

легчают работу, позволяя быстро узнавать о проблемах на всех серверах организации и получать информацию о них в одном расположении. Это экономит время и избавляет от рутинной работы по диагностике и активному мониторингу. И, в добавок ко всему, вы получаете советы по быстрому решению проблем.

Составные графики серверов

В центре экрана Enterprise Dashboard находится составной набор графиков, показывающих цветные линии для каждого из наблюдаемых серверов (рис. 13-7). По умолчанию эти графики имеют очень небольшой размер, но можно изменить и размер, и частоту сбора информации.

Эти графики являются еще одним способом получения графического представления о работоспособности систем. Даже на небольшом графике легко заметить аномалии. Как и в случае с диаграммой работоспособности, можно щелкать на графиках, чтобы просматривать подробную информацию о каждом событии.

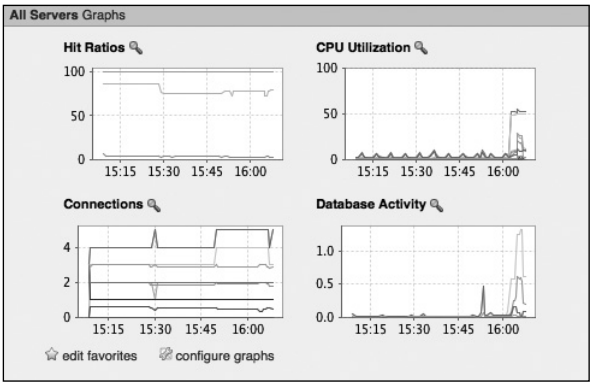


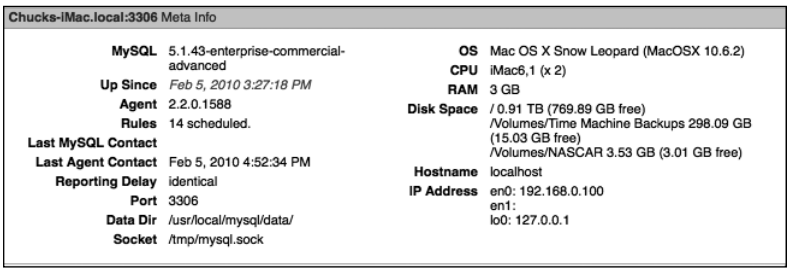
Рис. 13-7. Пример составных графиков сервера

Сведения о сервере

Еще одна полезная функция Enterprise Dashboard — возможность щелкнуть на имени сервера в списке для получения подробной информации о нем. В полученном отчете указана версия сервера MySQL, время его последнего запуска (время работы), местоположение данных, операционная система, а также сведения о ЦП, объеме памяти, дисковом пространстве и сетевых подключениях.

Эту информацию можно использовать для инвентаризации (определения оборудования, используемого в сети), а также для быстрого выяснения версии ОС, чтобы иметь представление о том, как решать возникшую проблему. Например, можно удаленно зайти на сервер, предварительно выяснив его имя, IP-адрес, версию MySQL и, что самое важное, версию операционной системы. Обычно администраторы запоминают или записывают эту

информацию. На рис. 13-8 показан пример отчета Enterprise Dashboard со сведениями о сервере.



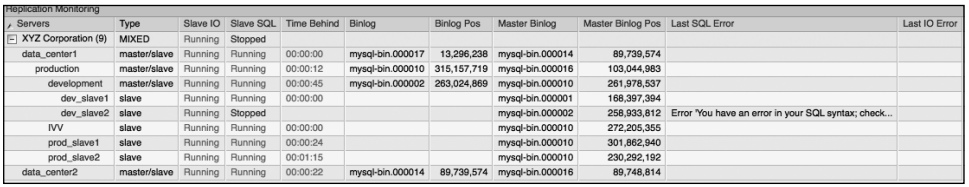
Chucks-iMac.local:3306 Meta Info

MySQL	5.1.43-enterprise-commercial-advanced	**OS**	Mac OS X Snow Leopard (MacOSX 10.6.2)
Up Since	*Feb 5, 2010 3:27:18 PM*	**CPU**	iMac6,1 (x 2)
Agent	2.2.0.1588	**RAM**	3 GB
Rules	14 scheduled.	**Disk Space**	/ 0.91 TB (769.89 GB free) /Volumes/Time Machine Backups 298.09 GB (15.03 GB free) /Volumes/NASCAR 3.53 GB (3.01 GB free)
Last MySQL Contact			
Last Agent Contact	Feb 5, 2010 4:52:34 PM	**Hostname**	localhost
Reporting Delay	identical	**IP Address**	en0: 192.168.0.100 en1: lo0: 127.0.0.1
Port	3306		
Data Dir	/usr/local/mysql/data/		
Socket	/tmp/mysql.sock		

Рис. 13-8. Сведения о сервере

Сведения о репликации

Вкладка Replication (Репликация) окна Enterprise Dashboard содержит список всех серверов, участвующих в репликации. Информация представлена в виде списка, в котором, как и в любом списке MEM, можно щелкать на элементах для получения дополнительной информации. На рис. 13-9 приведен пример отчета со сведениями о репликации.



Replication Monitoring

Servers	Type	Slave IO	Slave SQL	Time Behind	Binlog	Binlog Pos	Master Binlog	Master Binlog Pos	Last SQL Error	Last IO Error
XYZ Corporation (9)	MIXED	Running	Stopped							
data_center1	master/slave	Running	Running	00:00:00	mysql-bin.000017	13,296,238	mysql-bin.000014	89,739,574		
production	master/slave	Running	Running	00:00:12	mysql-bin.000010	315,157,719	mysql-bin.000016	103,044,983		
development	master/slave	Running	Running	00:00:45	mysql-bin.000002	263,024,869	mysql-bin.000010	261,978,537		
dev_slave1	slave	Running	Running	00:00:00			mysql-bin.000001	168,397,394		
dev_slave2	slave	Running	Stopped				mysql-bin.000002	258,933,812	Error 'You have an error in your SQL syntax; check...	
IVV	slave	Running	Running	00:00:00			mysql-bin.000010	272,205,355		
prod_slave1	slave	Running	Running	00:00:24			mysql-bin.000010	301,862,940		
prod_slave2	slave	Running	Running	00:01:15			mysql-bin.000010	230,292,192		
data_center2	master/slave	Running	Running	00:00:22	mysql-bin.000014	89,739,574	mysql-bin.000016	89,748,814		

Рис. 13-9. Сведения о репликации

Обратите внимание на то, что элементы списка сгруппированы по топологии (например, «XYZ Corporation», которую можно переименовать). Указаны тип топологии, роль (роли) сервера и ключевые сведения о репликации, включая состояние потоков, время отставания от главного сервера, текущий двоичный журнал, позиция в журнале, сведения о главном журнале и последние ошибки.

В этом примере мы видим проблему на подчиненном сервере *dev_slave2*, возникшую во время выполнения запроса. Это отличный пример того, как можно быстро получить представление о топологии репликации. В списке указаны главные и подчиненные серверы, сгруппированные по иерархии. Т.е., главный сервер указан сразу под группой, а под ним его подчиненные серверы. На рис. 13-9 легко увидеть, что сервер *development* имеет два подчиненных сервера: *dev_slave1* и *dev_slave2*, в то время как сам он является подчиненным для сервера *production*. Наличие всех сведений о репликации в одном месте избавляет от необходимости проведения мониторинга репликации на каждом сервере в отдельности.

Советники

Оповещения и графики не были бы такими информативными без рекомендаций, предоставляемых советниками. Все активные советники отображаются на вкладке Advisors (Советники) окна Enterprise Dashboard. Здесь же можно создавать собственных советников. На рис. 13-10 показан список советников, доступных по умолчанию в платиновой подписке.

На рис. 13-10 показаны советники, активные для определенного сервера в сети. Можно включать, отключать и изменять расписание любого советника (при удалении расписания теряются все собранные данные).

Наверно, самая полезная функция на этой странице — добавление собственных советников. Это позволяет расширять MEM в соответствии с собственными потребностями. Также это предоставляет возможность, так необходимую при переходе от ручного мониторинга к автоматическому: возможность сохранить проделанную работу.

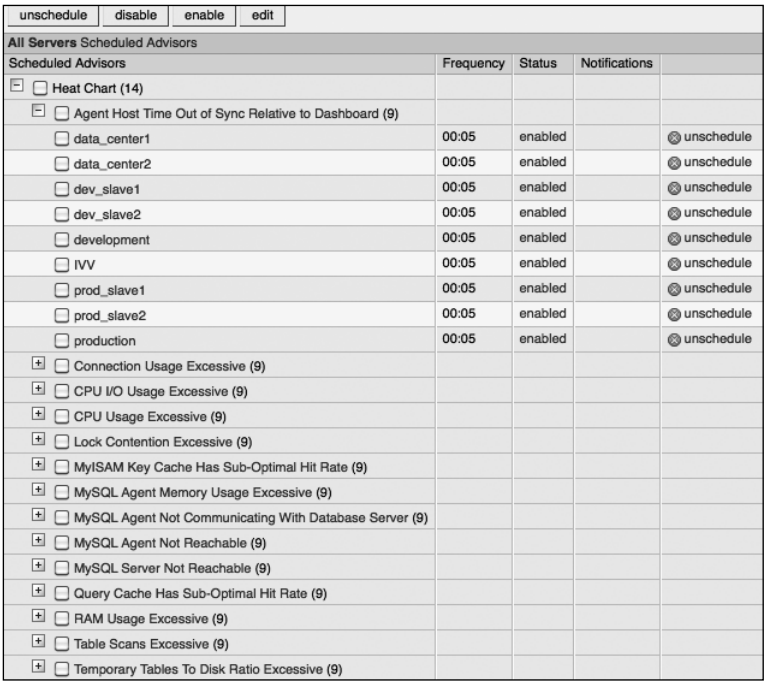


Рис. 13-10. Советники

Например, если вы создали механизм отчетов, выполняющий мониторинг пользовательского приложения, вы можете создать для него советника и добавить оповещения в Enterprise Dashboard. О том, как добавлять советников и оповещения, см. в руководстве по MySQL Enterprise Monitor на портале подписки Enterprise. Эта функция — одна из наиболее мощных и мало используемых функций MySQL Enterprise.

Анализатор запросов

Анализатор запросов — новейшая возможность MySQL Enterprise. Опытные администраторы давно знакомы с анализатором запросов для MySQL, представленным отдельным приложением, но с недавних пор он входит в состав инструментов Enterprise.

Анализатор запросов работает, перехватывая команды SQL при помощи MySQL Proxy и обрабатывая их, передавая затем на локальный сервер для выполнения. Это позволяет записывать статистические данные, которые можно просматривать в любое время. Анализатор запросов также поддерживает советников, отправляющих оповещения о медленных запросах. На рис. 13-11 приведена концептуальная схема, показывающая, как MySQL Proxy перехватывает запросы и передает данные MEM.

Анализатор запросов работает через определенный пользователем порт 6446 (по умолчанию) и может вызывать некоторое снижение производительности. Поэтому его следует включать только во время диагностики проблем.

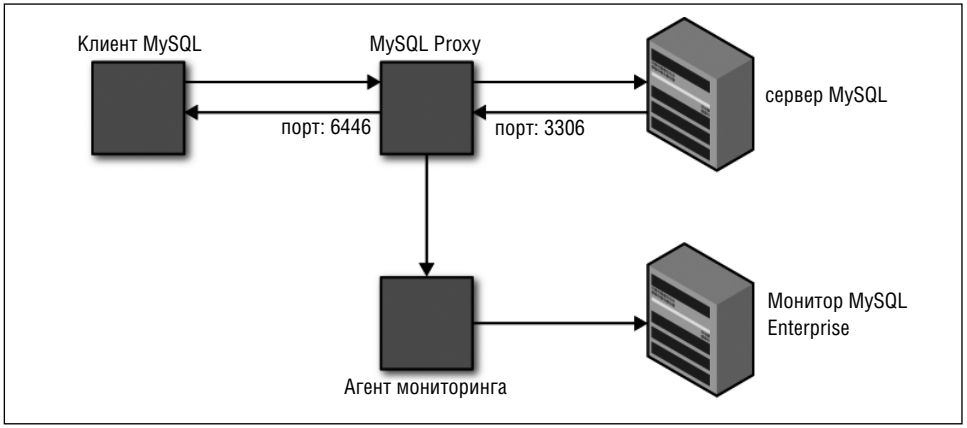
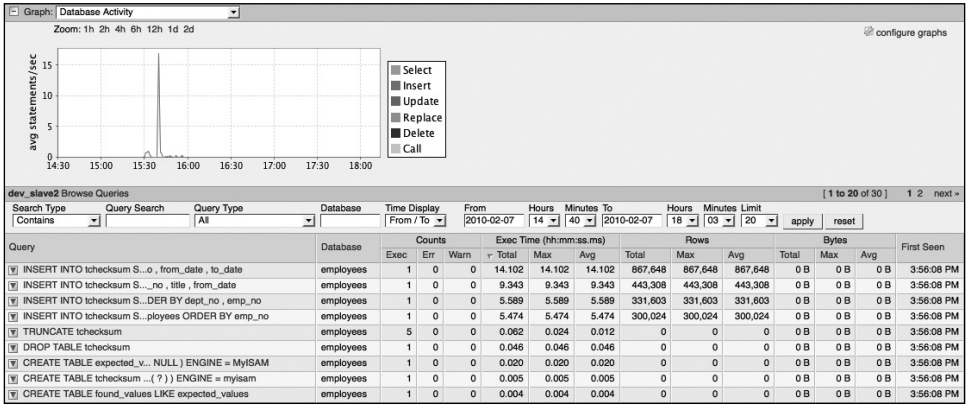


Рис. 13-11. Сбор данных для анализатора запросов при помощи MySQL Proxy

Чтобы собрать данные для анализатора запросов, необходимо указать клиенту, чтобы он подключался к порту MySQL Proxy, и настроить агента на использование порта 6446. Если не видите сообщений о запросах в Enterprise Dashboard, проверьте, удается ли подключиться к этому порту.

Хотя MySQL Proxy может приводить к незначительным задержкам в выполнении запросов, преимущества, предоставляемые возможностью анализа некорректных запросов, с избытком окупают эти задержки. Иногда анализатор запросов используют только на отдельных серверах разработки или экспериментальных системах, а не на рабочих серверах. Одна из положительных сторон анализатора запросов и Enterprise Dashboard — тесная интеграция этих инструментов. Если появилось оповещение о медленном запросе или требуется изучить отчет об использовании ЦП или другой отчет MySQL, вы увидите страницу анализатора запросов (которую можно открыть и напрямую,

щелкнув на ярлыке вкладки Query Analyzer). На рис. 13-12 показан пример страницы анализатора запросов в окне Enterprise Dashboard.



Query	Database	Counts			Exec Time (hh:mm:ss.ms)			Rows			Bytes			First Seen
		Exec	Err	Warn	Total	Max	Avg	Total	Max	Avg	Total	Max	Avg	
INSERT INTO tchecksum S...o , from_date , to_date	employees	1	0	0	14.102	14.102	14.102	867,648	867,648	867,648	0 B	0 B	0 B	3:56:08 PM
INSERT INTO tchecksum S..._no , title , from_date	employees	1	0	0	9.343	9.343	9.343	443,308	443,308	443,308	0 B	0 B	0 B	3:56:08 PM
INSERT INTO tchecksum S...DER BY dept_no , emp_no	employees	1	0	0	5.589	5.589	5.589	331,603	331,603	331,603	0 B	0 B	0 B	3:56:08 PM
INSERT INTO tchecksum S...ployees ORDER BY emp_no	employees	1	0	0	5.474	5.474	5.474	300,024	300,024	300,024	0 B	0 B	0 B	3:56:08 PM
TRUNCATE tchecksum	employees	5	0	0	0.062	0.024	0.012	0	0	0	0 B	0 B	0 B	3:56:08 PM
DROP TABLE tchecksum	employees	1	0	0	0.046	0.046	0.046	0	0	0	0 B	0 B	0 B	3:56:08 PM
CREATE TABLE expected_v... NULL) ENGINE = MyISAM	employees	1	0	0	0.020	0.020	0.020	0	0	0	0 B	0 B	0 B	3:56:08 PM
CREATE TABLE tchecksum ...(?)) ENGINE = myisam	employees	1	0	0	0.005	0.005	0.005	0	0	0	0 B	0 B	0 B	3:56:08 PM
CREATE TABLE found_values LIKE expected_values	employees	1	0	0	0.004	0.004	0.004	0	0	0	0 B	0 B	0 B	3:56:08 PM

Рис. 13-12. Страница анализатора запросов

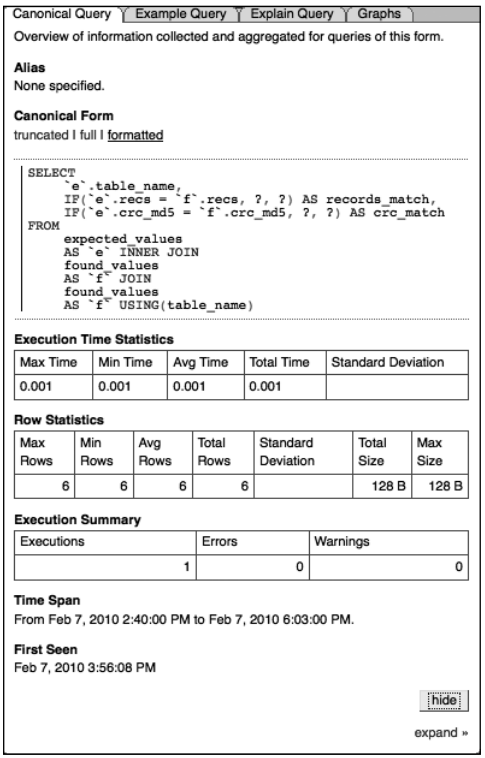
В левой части страницы анализатора запросов показан список серверов. Можно щелкнуть на имени сервера, чтобы просмотреть список выполненных на нем запросов, отсортированный по времени выполнения. Также можно использовать диаграмму, расположенную вверху, чтобы уменьшить отрезок времени и просмотреть запросы, выполненные на нем.

Список можно отсортировать по любому столбцу, щелкнув на заголовке этого столбца. Можно немного облегчить просмотр повторяющихся запросов и диагностических операций, если отсортировать их по времени выполнения, расположив самые долгие запросы вверху и просматривать команды запросов, объем обрабатываемых данных и т. д.

Можно щелкнуть на любой строке, чтобы получить более подробный отчет о запросе. На рис. 13-13 показан пример отчета в каноническом виде. Как и в случае с другими отчетами, можно просматривать дополнительную информацию, переходя по вкладкам: сам запрос на вкладке Example Query, выходные данные команды EXPLAIN на вкладке Explain Query (если эта возможность включена) и графики для времени выполнения, числа выполнений и числа возвращенных строк на вкладке Graphs.

Этот отчет содержит сведения о перехваченном запросе, включая каноническую форму запроса (графическое представление запроса в том виде, в каком он был написан), информацию о выполнении, в т.ч. затраченное время, и возвращенные или обработанные строки.

И снова мы видим инструмент мониторинга, позволяющий значительно сэкономить время при диагностике проблем с MySQL. Анализатор запросов из набора инструментов MySQL — важный инструмент мониторинга, помогающий поддерживать работоспособность серверов и обеспечивать эффективное выполнение запросов.



Canonical Query | Example Query | Explain Query | Graphs

Overview of information collected and aggregated for queries of this form.

Alias
None specified.

Canonical Form
truncated | full | formatted

```
SELECT
     `e`.table_name,
     IF(`e`.recs = `f`.recs, ?, ?) AS records_match,
     IF(`e`.crc_md5 = `f`.crc_md5, ?, ?) AS crc_match
FROM
     expected_values
     AS `e` INNER JOIN
     found_values
     AS `f` JOIN
     found_values
     AS `f` USING(table_name)
```

Execution Time Statistics

Max Time	Min Time	Avg Time	Total Time	Standard Deviation
0.001	0.001	0.001	0.001	

Row Statistics

Max Rows	Min Rows	Avg Rows	Total Rows	Standard Deviation	Total Size	Max Size
6	6	6	6		128 B	128 B

Execution Summary

Executions	Errors	Warnings
1	0	0

Time Span
From Feb 7, 2010 2:40:00 PM to Feb 7, 2010 6:03:00 PM.

First Seen
Feb 7, 2010 3:56:08 PM

hide

expand »

Рис. 13-13. Канонический отчет о запросе

MySQL Enterprise и облачные вычисления

Инструменты MySQL Enterprise могут хорошо функционировать в среде облачных вычислений. Однако при этом действуют те же правила относительно долговечности данных и экземпляров серверов. Убедитесь, что запросили постоянный IP-адрес для сервера MEM. Частые запуски и остановки экземпляров не нанесут ущерба, но изменение имен хостов и некоторые изменения конфигурации могут привести к прекращению получения отчетов от агентов мониторинга. Обычно эта проблема решается усечением таблицы *mysql.inventory*, но лучше использовать одинаковые имена и IP-адреса для всех серверов.

Одним из главных преимуществ запуска инструментов MySQL Enterprise в платном облаке является то, что вы платите только за вычисления и место для хранения данных. Передача данных внутри облака обычно выполняется бесплатно или гораздо дешевле, чем передача данных в облако и из облака.

Дальнейшая информация

Полное и подробное описание MEM и множества его функций выходит за рамки нашей книги. В Интернете можно найти массу информации на эту тему. Подробнее о пользовательских советниках и прочем см. в руковод-

стве по MySQL Enterprise Monitor, расположенном на портале подписки Enterprise. Ответы на распространенные вопросы можно найти в базе знаний на этом же портале.

Заключение

MySQL — наиболее популярная СУБД с открытым исходным кодом. И этому есть веские причины. Гибкость этой СУБД позволяет использовать ее организациям, имеющим инфраструктуру любой сложности: от одиночного веб-сервера до систем оперативной обработки транзакций (OLTP) и крупных дата-центров с высокой доступностью. MySQL может справиться со всем этим, и не только. MEM позволяет расширять инфраструктуру на основе MySQL, сохраняя максимальную производительность и надежность.

Если требуется обеспечить высокую доступность и построить максимально надежный дата-центр, рассмотрите возможность приобретения платиновой подписки MySQL Enterprise. Существуют и другие решения, но ни одно из них не даст такой глубокой экспертной оценки, как MySQL Enterprise, и не предоставит возможностей, аналогичных советникам и анализатору запросов, по такой выгодной цене.

Джоэл был во всеоружии. Он отправил шефу коммерческое предложение на так необходимую ему подписку MySQL Enterprise для всех серверов компании. Господин Саммерсон имел репутацию прижимистого руководителя, так что Джоэл был готов отстаивать свое приобретение и не сдаваться, пока Саммерсон хотя бы не выслушает его.

Каждый раз, слыша шаги за дверью, Джоэл инстинктивно настораживался, зная, что его начальник может войти в любой момент. В очередной раз услышав шаги, Джоэл напрягся. Однако Саммерсон промелькнул, даже не заглянув к нему. «Подождем», — прошептал Джоэл.

— Джоэл, мне понравилась эта штука MEM. Напиши, сколько это будет стоить и...

— Я уже подготовил для вас резюме, Боб, — прервал его Джоэл.

Саммерсон поднял брови и сказал:

— Отправь-ка его мне, и я посмотрю, сможем ли мы втиснуть это в бюджет на следующий год.

Сказав это, г-н Саммерсон ушел к своей следующей жертве.

Джоэл с облегчением опустился на стул и с удовлетворением сложил руки на груди:

— Будем считать, что я молодец.

ЧАСТЬ III

Среды с высокой доступностью

Облачные вычисления и кластер MySQL открывают новые возможности по обеспечению высокой доступности. Третья часть книги посвящена этим темам.

ГЛАВА 14

Облачные вычисления

Джоэл прикрыл дверь своего кабинета, чтобы повесить пиджак, и тут же в дверь постучали, от чего он чуть не подпрыгнул.

— Входите, — сказал Джоэл, открывая дверь и отходя к столу. — Г-н Саммерсон, доброе утро, сэр!

— Доброе утро, Джоэл. Ты хорошо справился с отчетом о высокой доступности и масштабируемости наших серверов. Особенно мне понравились рекомендации по повышению производительности.

— Спасибо... — Джоэл затаил дыхание, лихорадочно пытаясь угадать, какую задачу поставят перед ним на этот раз.

— Вчера вечером мы подписали договор о доработке одного из наших продуктов под требования нового заказчика. На договоре еще чернила не высохли, и я не буду углубляться в детали, скажу лишь, что нам понадобится много новых серверов в высокодоступной среде. Конечно, для баз данных мы будем использовать MySQL.

Джоэл попытался вспомнить все, что читал об обеспечении высокой доступности в MySQL, прикидывая, сколько денег потребуется для развертывания группы серверов. Г-н Саммерсон тем временем продолжал:

— ...И еще надо учесть балансировку нагрузки.

— Да, сэр, — произнес Джоэл после неловкой паузы.

— Проблема в том, что у нас нет средств на покупку новых серверов, и по договору мы должны обеспечивать хостинг баз данных клиента всего шесть месяцев. Естественно, руководство никогда не согласится оплатить новое оборудование, которое станет ненужным через полгода. Не говоря уже о том, что это существенно снизит нашу прибыль.

Джоэл не знал, что сказать, и просто ждал.

— Поэтому ты должен придумать решение на основе облачных вычислений.

Сказав это, г-н Саммерсон похлопал Джоэла по плечу и ушел. Джоэл постоял еще немного, потом подошел к столу и сел. Он старался следить за появлением новых технологий, но про облачные вычисления слышал впервые. Джоэл достал свою уже порядком потрепанную книгу по MySQL и открыл главу, до которой он еще не добрался.

— Что ж, посмотрим, что это такое...

Современная экономика предъявляет новые требования и позволяет внедрять новые решения при планировании ИТ-инфраструктуры. Компании больше не могут позволить себе просто покупать новое оборудование каждый раз, когда требуется повысить вычислительную мощность. Хотя стоимость оборудования за последнее десятилетие заметно снизилась, то же самое произошло и с прибылями компаний, особенно в последнее время.

Поэтому принимаемые решения должны требовать минимума финансовых вложений и использовать самые недорогие услуги и инструменты, позволяющие расширить клиентурную базу и линейки продуктов, снижая затраты и повышая прибыль. В конце концов, все дело здесь в деньгах.

Потребность в недорогих решениях вычислительных задач привела к появлению нового способа использования компьютеров, основанного на принципе «оплачивается только использованное», позволяющего компаниям наращивать вычислительные мощности в соответствии с текущими потребностями. Такова суть того, что называют *облачными вычислениями* (cloud computing).

Что такое облачные вычисления?

С термином «облачные вычисления» сплошь и рядом возникает путаница: к сожалению, у этого термина несколько определений, иногда противоречивых. Одни утверждают, что это модный термин для существующих технологий, другие спорят об академических, научных и иногда социальных аспектах, а третьи настаивают на том, что облачные вычисления — это будущее информационных технологий.

Кое-кто категорически утверждает, что это не более чем просто сетевые вычисления, есть и такое мнение, что облачные вычисления — это не меньше, чем Интернет целиком. Обе эти точки зрения неверны. Наконец есть те, кто считает такие вычисления особым видом услуг, и эта точка зрения ближе всего к истине.

В общем, облачные вычисления — это группа обновленных технологий, включающих сетевые вычисления и виртуализацию, вкупе с программными интерфейсами приложений (API) и утилитами для обеспечения доступа к виртуализированным средам. Джордж Риз (George Reese) в книге *Cloud Computing Architectures* (O'Reilly, *http://oreilly.com/catalog/9780596156374*) говорит: «В технологиях, составляющих облачные вычисления, нет ничего принципиально нового». Это трезвое наблюдение, которое не могут осознать некоторые «эксперты» и маркетологи. Как бы то ни было, прорыв достигнут за счет объединения существующих технологий, благодаря которому такие гиганты как Amazon открыли совершенно новый способ их использования.

Джеймс Говернор (James Governor) в своем труде «15 Ways to Tell It's Not Cloud Computing» («15 отличий НЕ облачных вычислений») весьма категорично судит, что можно считать облачными вычислениями, а что нет. Если

перефразировать его слова, то получится, что облачными вычислениями не является ИТ-решения, сложные для объяснения и освоения, изолированные, требующие выделенного подключения или приобретения дополнительного оборудования. Прав он или нет — вопрос, но он заставил задуматься некоторых руководителей компаний, пытающихся добавить в название своих продуктов слово «облачные». Словом, многие имеют неправильное представление о том, что такое облачные вычисления.

Термин «облачные вычисления» происходит от концептуальной схемы, обозначающей ресурсы, расположенные в крупной сети (т. н. облаке). Символ облака используется потому, что характеристики ресурса (оборудование, ОС и т. д.) скрыты и не имеют отношения к тому, какие сервисы этот ресурс предоставляет, — это просто ресурс, который можно использовать. Таким образом, вместо шлюзов, маршрутизаторов и серверов вы видите ресурсы, предоставляемые как сервис. Пользователей ресурса не интересует, как реализован этот сервис. Главное, чтобы этот ресурс удовлетворял их потребности и был доступен, когда необходимо.

Чтобы не путаться в определениях, возьмем за основу определение Национального института стандартов и технологий США (National Institute of Standards and Technology, NIST). Итак, облачные вычисления — это модель обеспечения удобного сетевого доступа по запросу к общему набору настраиваемых вычислительных ресурсов (таким как сети, серверы, хранилища, приложения и службы), оперативно предоставляемых и освобождаемых с минимальным участием поставщика услуг. Такая модель облака обеспечивает высокую доступность, обладает пятью основными характеристиками, включает три модели предоставления услуг и четыре модели развертывания.

Большинство исследователей выделяет следующие основные характеристики облачных вычислений:

- *Самообслуживание по запросу* Пользователи могут выбирать, что и когда им нужно, без непосредственного взаимодействия с поставщиком или посредниками.
- *Сетевой доступ* Ресурсы доступны посредством существующих сетевых технологий.
- *Пулы ресурсов* Пользователи совместно используют оборудование поставщика (например, с использованием модели множественной аренды).
- *Оперативная расширяемость* Пользователи могут быстро задействовать дополнительные ресурсы, вручную или автоматически.
- *Управление услугами* Пользователям предоставляются средства для мониторинга и управления ресурсами, активные или пассивные (см. гл. 10).

Три модели предоставления услуг таковы:

- *Инфраструктура как услуга (Infrastructure as a Service, IaaS)* Ресурсы предоставляются как виртуальные экземпляры аппаратной или программной платформы. Клиент может добавлять виртуальные вычисли-

тельные ресурсы по запросу (например, серверы или балансировщики нагрузки). Таким образом, компоненты ИТ-инфраструктуры предоставляются как компоненты или промежуточное ПО. Пользователь имеет доступ к ресурсам и может управлять ими (например, управлять предоставленным сервером).

- *Платформа как услуга (Platform as a Service, PaaS)* Интерфейс API позволяет клиентам создавать приложения, предназначенные специально для выполнения на оборудовании (платформах) провайдера. Поставщик предоставляет среду для размещения информации и средства программирования, что позволяет пользователям создавать решения для определенного окружения.
- *ПО как услуга (Software as a Service, SaaS)* ПО предоставляется как ресурс в виде приложений, запускающихся на оборудовании провайдера. Пользователь видит только интерфейс, позволяющий работать с этим ПО как с прикладной программой. Оборудование, операционная система и все остальное скрыто и управляется исключительно поставщиком. Это самая старая модель, в настоящее время включенная в определения облака, долгое время известная под именем *поставщик услуг доступа к ПО* (Application Service Provider, ASP).

Модели развертывания описывают доступность полученных решений. Существует четыре модели:

- *закрытая* — доступ к ресурсам получает только клиент;
- *доступ для сообщества* — совместный доступ одного или нескольких клиентов;
- *открытая* — доступ к ресурсам имеют все;
- *гибридная* — инфраструктура, включающая разные модели. Обычно происходит разделение на закрытые и открытые ресурсы, которые могут взаимодействовать.

Документ с полным описанием представления NIST об облачных вычислениях можно найти по адресу *http://csrc.nist.gov/groups/SNS/cloud-computing*.

Архитектуры облачных вычислений

В этом разделе мы вкратце расскажем о базовых технологиях, часто применяемых в облачных вычислениях. В большинстве решений для облачных вычислений используются технологии из следующего списка:

- виртуализация;
- сетевые вычисления;
- транзакции;
- эластичность;
- библиотеки ПО.

Виртуализация

Существует много форм виртуализации. Если вы использовали VirtualBox компании Sun или Virtual PC компании Microsoft, вы уже знакомы с виртуализацией. По сути, эта технология создает псевдоплатформу, основанную на концептуальной модели вычислительной техники. Например, можно запустить экземпляр ОС Windows на компьютере под управлением Linux, используя VirtualBox. VirtualBox программно имитирует все компоненты компьютера, формируя среду, в которой Windows может работать, как на реальном оборудовании.

Это только одна из форм виртуализации. Существуют различные механизмы эмуляции оборудования, оптимизации загрузки, выполнения и управления экземплярами. Виртуализация, используемая в большинстве решений IaaS, требует использования заранее подготовленных файлов с описанием машин (называемых *образами*), где каждая виртуальная машина называется *экземпляром* образа. Например, облако Amazon использует распространенную open source-технологию виртуализации Xen. Эта технология позволяет масштабировать виртуализированнное оборудование (например, наращивать число процессоров), обеспечивает отказоустойчивость и дает другие преимущества.

Более того, некоторые поставщики позволяют изменять существующие образы, настраивая машины под текущие потребности, используя специальные утилиты или описание машины в определенном формате. Если вы решите перейти от одного поставщика к другому, это может вызвать проблемы: не факт, что образы окажутся переносимыми. Прежде чем потратить время и силы на настройку образа, проверьте, что сказано в документации поставщика об образах и их настройке.

Сетевые вычисления

В прошлом, когда вычислительные мощности были ограниченными, для решения сложных аналитических или научных задач была изобретена технология *сетевых вычислений*, позволяющая задействовать дополнительные вычислительные мощности путем объединения компьютеров в группы. Задача разбивается на небольшие подзадачи, которые можно передавать отдельным системам для выполнения, а результаты затем собираются воедино.

Ключевая технология, обеспечивающая взаимодействие разных компьютеров, — сложный механизм организации очереди. Этот механизм похож на технологический процесс: система, управляющая данными (*диспетчер данных*), передает задания и необходимые данные подчиненным системам и получает от них результаты. Для этого требуется установить одну или несколько систем, управляющих очередями, которые будут раздавать задания подключенным компьютерам, а затем возвращать результаты диспетчеру данных. Если пользователь желает поучаствовать в сетевых вычислениях, он подключает свой компьютер к системе управления очередями и обраща-

ется к пулу пакетов обработки, содержащих задание и данные. Его система-донор выполняет задание и отправляет результаты системе управления очередями. Упрощенный пример этого процесса показан на рис. 14-1.

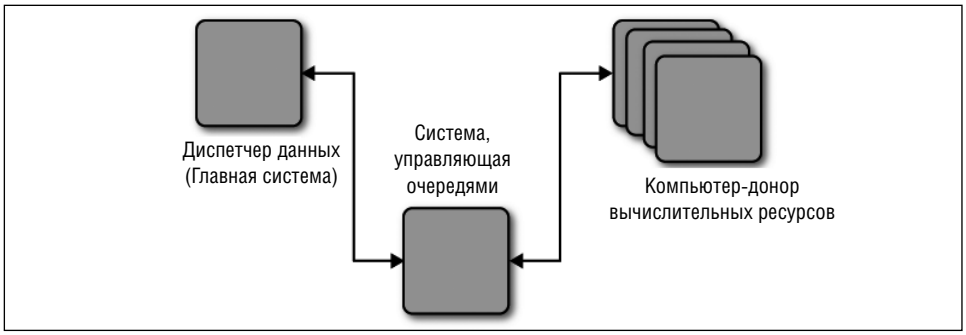


Рис. 14-1. Процесс сетевых вычислений

Системы управления очередями часто используются в облачных вычислениях. Это позволяет переносить существующие решения для сетевых вычислений и создавать новые решения в системах облачных вычислений. Поэтому некоторые и настаивают на том, что облачные вычисления — это просто сетевые вычисления с поддержкой виртуализации. Но, как будет показано ниже, облачные вычисления включают гораздо больше, чем эти технологии.

Транзакционные вычисления

Транзакционные вычисления знакомы пользователям БД: несколько порций данных обрабатываются вместе как одна транзакция и объединяются с остальными данными. Идея заключается в определении задания, по обработке некоторых данных в единой операции (транзакцией). Лучшие решения с применением сетевых вычислений используют эту концепцию для обеспечения получения надлежащих результатов. Однако облачные вычисления немного сложнее. В частности, крупные приложения, использующие транзакции, обрабатывают их достаточно длительное время, тогда как решениям на основе сетевых вычислений на обработку отводится очень небольшое время.

Хорошие новости заключаются в том, что в облаке все-таки можно создать вычислительную систему, использующую транзакции. Чтобы сделать это, необходимо обеспечить долговечность доступных вычислительных ресурсов и предоставить механизмы сегментации и параллельной обработки данных. Если вы подумаете, что это похоже на фермы серверов, то будете правы. Большинство поставщиков облачных вычислений предоставляют виртуализированные ресурсы для решений на основе транзакций, включая балансировщики нагрузки, постоянные экземпляры и постоянно выделенные сетевые ресурсы.

Эластичность

Термин «эластичность» (elasticity) используется для обозначения абстрактного характера сетевого или системного ресурса, предоставляемого в пользование. Например, Amazon позволяет назначать определенный IP-адрес любому экземпляру сервера в облаке. Это очень важно в транзакционных системах, где несколько серверов должны отвечать на обращения по одному адресу. Виртуализировать серверы, чтобы они могли работать в любом месте облака, полезно, но при этом нужно гарантировать неизменность их IP-адреса.

В этом случае IP-адрес становится эластичным ресурсом, который можно выдавать любому экземпляру. Он не привязан к конкретной системе. Аналогично, эластичными могут быть дисковые ресурсы, что позволяет сделать данные, хранимые на этом ресурсе, доступными для любого экземпляра в облаке.

Эластичность решает проблему работы виртуализированных систем в пулах. Такие системы полностью соответствуют принципу plug-and-play, и их легко создавать и уничтожать. Например, в процессе разработки можно менять компьютеры с разными операционными системами и (возможно, с небольшими изменениями) продолжать обращаться к тем же данным, не создавая БД с нуля.

Библиотеки ПО

Вас, возможно, заинтересует, как все эти технологии совместно работают и как можно работать с ресурсами в динамическом окружении. Ответ заключается в том, что большинство поставщиков облачных вычислений имеют специальные наборы утилит для создания и управления ресурсами облака.

Например, Amazon имеет утилиты со специальными API для управления ресурсами, создания экземпляров и томов (дисковых объектов) и много другого. Это утилиты Amazon EC2 API Tools для работы с ресурсами облака и Amazon EC2 AMI (Amazon Machine Image) Tools для создания и изменения образов систем.

Аналогично, Microsoft Azure имеет расширения среды разработки .NET, позволяющие создавать приложения для работы с облаком и запускать их в облаке Azure. Однако на этом сходство заканчивается, так как среда Microsoft Azure требует создавать приложения на основе этих библиотек, а Amazon — нет.

Таким образом, библиотеки ПО являются тем скрепляющим материалом, который позволяет всем перечисленным технологиям работать вместе, превращаясь в нечто большее, чем они являются по отдельности — среду облачных вычислений.

Экономичны ли облачные вычисления?

В этом споре ломают копья и аналитики и ученые мужи. А ответ зависит от того, какое облако используется, сколько серверов требуется, какой объем дискового пространства необходим и как долго всем этим придется пользоваться.

В одном исследовании (*http://www.uptimesoftware.com/uptimeblog/cloud-virtualiza tion/cost-of-cloud-computing-expensive*) показано, что стоимость решения с использованием облака для типичной компании, занимающейся электронной коммерцией, за пятилетний период не намного выше по сравнению с традиционным решением (покупка собственного оборудования).

Это исследование показывает, что в традиционном решении первоначальные вложения очень высоки. По истечении пяти лет оборудование остается в собственности компании (или списывается как устаревшее), тогда как оборудование облака для фирмы как бы не существует и не влияет на стоимость проекта.

Это значит и то, что при использовании облака нет регулярных затрат на обновление оборудования. В указанном исследовании эти затраты не учитываются, но если бы учитывались, то стоимость традиционного решения за пять лет оказалась бы значительно выше.

Чтобы ответить на вопрос об экономичности облачных вычислений, вам придется провести собственное исследование. Стоимость будет разной для разных компаний и проектов. Лучший способ оценить эту стоимость — определить количество нужных вам серверов, объем хранимых данных, объем данных, перемещаемых по облаку, и необходимые дополнительные функции (VPN, балансировка нагрузки и т. д.). В частности, нужно выяснить, какие платные компоненты предоставляет поставщик, и оценить стоимость решения на основе этих параметров. Сделав это, можно определить стоимость традиционного решения, добавить стоимость обслуживания и модернизации, а затем провести объективное сравнение.

Многие клиенты используют облачные вычисления не для экономии денег, а из-за их гибкости. Об этом мы поговорим в следующем разделе. Однако некоторые организации не пользуются облачными вычислениями из-за разных опасений. Например, некоторые компании не допускают хранения своих данных в системах, которыми они не владеют, и к которым (предположительно) может получить доступ посторонний администратор (т. е. сотрудник поставщика облака). Если у вас возникли подобные опасения, свяжитесь с поставщиком облака и обсудите все беспокоящие вас моменты и трезво оцените риски. Одно из возможных решений — выделить конфиденциальные данные и хранить в облаке только общедоступную информацию. При использовании веб-служб Amazon (Amazon Web Services, AWS) доступны функции, позволяющие изолировать экземпляры облака или подключать их через VPN к собственной ИТ-инфраструктуре (подробнее см. по адресу *http://aws.amazon.com/vpc*).

Применение облачных вычислений

Выше мы объяснили, что такое облачные вычисления, теперь разберемся, что с их помощью можно делать. Выгоду от их применения могут получить практически все: стартапы, ищущие недорогой способ выхода на рынок, исследователи, которым требуются значительные вычислительные мощности на короткое время, ИТ-менеджеры, пытающиеся удовлетворить требования пользователей и при этом уложиться в бюджет. В этом разделе мы обсудим некоторые распространенные способы использования облачных вычислений:

- *традиционные веб-сервисы* — ресурсы облака предоставляют содержимое или приложения пользователям через Интернет;
- *совместно используемые сервисы* — в облаке работает одно или несколько приложений, совместно используемых разными пользователями. Пример — приложение, позволяющее партнерам взаимодействовать и совместно работать над данными (например, в цепочке поставщиков).
- *горизонтальное масштабирование* — позволяет быстро расширять приложение, используя решения, работающие в облаке и подключающиеся к сети предприятия;
- *расширение облака* — пользователи могут быстро развертывать временные ресурсы для решения текущих краткосрочных вычислительных задач;
- *исследования и разработка* — разработчики могут изучать различные конфигурации систем и приложений, не приобретая специального оборудования.

Как видите, облачные вычисления имеют много областей применения, и каждый день обнаруживаются новые. То, что мы показали — лишь небольшая часть возможностей.

Преимущества облачных вычислений

К потенциальным преимуществам облачных вычислений относится следующее:

- *Меньше время выполнения и больше скорость отклика* Применяя распределенные вычисления или горизонтальное масштабирование, можно уменьшить время выполнения задач и значительно ускорить доступ к данным. Такого же эффекта можно добиться при помощи аппаратных решений, но стоимость будет очень высокой. Решения с использованием облака позволяют создавать необходимое количество экземпляров систем и платить только за то, что реально используется.
- *Минимизация задач по обслуживанию и рисков, связанных с инфраструктурой* Можно не беспокоиться об аппаратных сбоях. Хозяин машин, работающих в облаке — поставщик услуг, он сам обслуживает их, так что вам не требуются дополнительные сотрудники и затраты.
- *Меньше стоимость входа* Возможность платить только за то, что используется, освобождает от вложений в ИТ-инфраструктуру, которая, к

тому же, может и не понадобиться. Хорошо, если инфраструктуру можно расширять «на лету», но еще лучше, что ее можно быстро сокращать, не задумываясь о том, куда девать лишнее оборудование.

- *Выше темпы разработки* Низкая планка первоначальных вложений и возможность платить только за необходимое — два фактора, позволяющие начать работу над новым приложением с гораздо меньшими инвестициями, чем в прошлом (на собственном или арендованном оборудовании). Благодаря этому, начинающим разработчикам стало легче создавать конкурентоспособные продукты.

Конечно, на каждое преимущество найдется и недостаток. Перечислим риски, появляющиеся при использовании облачных вычислений:

- *Сбои служб* Соглашения об уровне услуг (Service Level Agreement, SLA) для облачных вычислений расплывчаты или вовсе не существуют. Так что остается только надеяться, что поставщик не подведет.
- *Пиковые нагрузки обходятся дорого* Если нагрузка неожиданно возросла, с ней можно справится путем подключения дополнительных ресурсов облака, но при этом возрастет и плата за пользование ресурсами.
- *Недостаток функций* В какой-то момент вашей архитектуре или приложении может понадобиться функция, не поддерживаемая поставщиком.
- *Риски для безопасности* Компьютеры используются совместно с другими пользователями, и ошибки в ПО могут привести к утечке или краже данных.

Поставщики облачных вычислений

Когда начинает развиваться какая-то новая технология, обязательно появляется множество поставщиков, продуктов и служб, утверждающих, что они предоставляют эту технологию. Облачные вычисления — не исключение. Существуют сотни поставщиков, предоставляющих все — от специализированного оборудования, служб и платформ до проектов — «под ключ». Если воспользоваться определением NIST, приведенным выше, сразу станет ясно, что многие поставщики удовлетворяют не всем пунктам этого определения.

Однако есть несколько поставщиков, в полной мере соответствующих определению облачных вычислений. Ниже приведен список из 10 лучших поставщиков и краткое описание предоставляемых ими решений:

- *3Tera (http://www.3tera.com)* Поставщик IaaS, специализирующийся на быстром горизонтальном масштабировании.
- *Akamai (http://www.akamai.com)* Поставщик IaaS, специализирующийся на управлении данными через Интернет.
- *Amazon (http://aws.amazon.com)* Поставщик облачных вычислений, предлагающий виртуализированные решения SaaS, PaaS и IaaS, а также хранилища данных.

- *Enki Consulting (http://www.enkiconsulting.net)* Поставщик IaaS, специализирующийся на закрытых виртуальных центрах обработки данных (ЦОД).
- *IBM Blue Cloud (http://www.ibm.com/ibm/cloud)* Поставщик облачных вычислений, предлагающий виртуализированные решения SaaS, PaaS и IaaS.
- *Joyent (http://www.joyent.com)* Поставщик IaaS, специализирующийся на обеспечении нужд крупных предприятий.
- *Layered Technologies (http://www.layeredtech.com)* Поставщик PaaS и IaaS.
- *Rackspace (http://www.rackspace.com)* Поставщик PaaS, специализирующийся на хостинге веб-приложений.
- *Salesforce.com (http://www.salesforce.com)* Поставщик SaaS, специализирующийся на системах управления взаимодействием с клиентами (CRM), поддерживающих совместную работу.
- *Terremark (http://www.terremark.com)* Поставщик IaaS.

Даже Apple объявила о планах по созданию сервисов и продуктов, доступных через Интернет с оплатой по факту использования. Хотя ничего еще не сказано конкретно, уже ясно, что эти решения не представляются как облачные сервисы, они будут соперничать с проектами Google Docs и Microsoft Office Online. В общем, концепции, лежащие в основе облачных вычислений, оказывают заметное влияние на многих крупных игроков на поле информационных технологий.

Учитывая популярность, широкий ассортимент, зрелость и сложность AWS, в этой главе мы сосредоточимся на продуктах Amazon. При выборе поставщика облачных вычислений мы рекомендуем рассматривать все подходящие варианты, однако решения Amazon получают все большее распространение.

«Amazon» как «ксерокс»

Многие называют любой копировальный аппарат «ксероксом» и используют глагол «ксерокопировать», многие специалисты из области высоких технологий называют облачные вычисления (или просто «облака») названиями продуктов AWS. Время покажет, станут ли решения Amazon стандартом, по которым будут оцениваться все остальные решения. Однако мы часто слышим, что в определение облачных вычислений включают эластичность — термин, ставший популярным благодаря названию IaaS-продукта от Amazon: Elastic Cloud Computing (EC2).

AWS

Amazon предлагает широкий набор решений и средств разработки, имеющий общее название AWS, хотя часто его называют просто «облаком Amazon».

В AWS входят вычислительные службы (собственно, облако), средства доставки содержимого, средства поддержки СУБД, решения для электронной коммерции, средства обмена сообщениями, средства мониторинга, сетевые утилиты, решения для платежей и создания счетов, хранилища данных, службы поддержки продуктов AWS, средства управления веб-трафиком и средства поиска рабочей силы. Amazon регулярно добавляет новые продукты, так что со времени написания книги этот список мог стать длиннее.

Продукты Amazon платные, но разработаны так, что пользователи могут использовать и оплачивать только необходимые услуги. В следующем разделе мы обсудим технологии, доступные в AWS, и вкратце расскажем, как работать с облачными вычислениями в варианте Amazon.

Обзор технологий

Так как технологии многочисленны и разнообразны, мы остановимся только на том, что касается создания надежных ЦОД. Если захотите узнать о продуктах AWS подробнее, зайдите на сайт *http://aws.amazon.com* и найдите раздел Products.

Все перечисленные ниже технологии основаны на простых веб-сервисах, что позволяет легко строить приложения, способные взаимодействовать со всеми другими утилитами посредством веб-интерфейса RESTful.

- *Amazon Elastic Compute Cloud (EC2)* Вместе с Amazon Simple Storage Service (S3) является основой облака Amazon. Это главная технология, которая делает возможной работу облака. Она управляет виртуальными вычислительными ресурсами.
- *Amazon Elastic MapReduce* Предоставляет среду для обработки больших массивов данных, используя инфраструктуру Hadoop.
- *Автомасштабирование (Auto Scaling)* Обеспечивает автоматическое масштабирование решений по заданным пользователем параметрам. Это ключевая функция для построения решений с высокой доступностью.
- *Amazon CloudFront* Служба управления содержимым, позволяющая предоставлять статическое и потоковое (иногда его называют активным) содержимое пользователям в различных географических точках.
- *Amazon SimpleDB* Предоставляет очень простую нереляционную СУБД.
- *Amazon Relational Database Service (RDS)* Вариант СУБД MySQL от Amazon. Можете использовать эту службу, чтобы не создавать и не обслуживать собственный экземпляр MySQL.
- *Amazon Fulfillment Web Service (FWS)* Предоставляет набор инструментов для электронной коммерции. Те же инструменты используются в широко известном интернет-магазине Amazon.
- *Amazon Simple Queue Service (SQS)* Служба управления очередью сообщений, используемая в решениях на основе распределенных вычислений.

- *Amazon CloudWatch* Предоставляет возможности мониторинга всех ресурсов из облака Amazon, потребляемых пользователем.
- *Amazon Virtual Private Cloud (VPC)* Недавно появившаяся интересная возможность, которая позволяет предприятиям расширять свою инфраструктуру, используя VPN-подключения к облаку Amazon. Ресурсы, размещенные в VPC, взаимодействуют с внешней инфраструктурой так, как будто они находятся в интрасети. Такое облако позволяет решить проблему быстрого наращивания вычислительных ресурсов.
- *Эластичная балансировка нагрузки (Elastic Load Balancing)* Еще один ключевой компонент служб облака. Обеспечивает возможность балансировки нагрузки на сеть в используемом решении.
- *Amazon Flexible Payments Service (FPS)* Библиотека разработки, предоставляющая средства обработки платежей для создания благотворительных фондов и сайтов электронной коммерции.
- *Amazon DevPay* Онлайновая служба билинга и учета для онлайн-магазинов.
- *Amazon S3* Еще один ключевой компонент облака Amazon. Предоставляет быстрое постоянное хранилище для файлов размером до 5 Гб.
- *Amazon Elastic Block Storage (EBS)* Ключевой модуль, хранящий данные. Это блочное устройство для хранения и получения данных, которое можно присоединить к любому экземпляру.
- *AWS Import/Export* Службы импорта–экспорта больших объемов данных из облака.
- *AWS Premium Support* Службы поддержки для всех продуктов AWS, предоставляющие персональную помощь в построении и запуске приложений с использованием продуктов AWS.
- *Alexa Web Information Service* Служба, собирающая метаданные о трафике и структуре веб-сайтов.
- *Alexa Top Sites* Служба, ранжирующая веб-сайты на основе данных о трафике и посещаемости.
- *Amazon Mechanical Turk* Решение для организации совместной работы, поддерживающее найм кадров по запросу. Создано специально для интеграции в вычислительные системы задач, решаемых оператором, например создания изображений, записи звука и т. д.

Мы привели список продуктов, имеющих отношение к облачным вычислениям, а теперь рассмотрим базовые технологии, о которых необходимо знать, чтобы приступить к работе со своим первым решением на основе облака. Хотя этот список может показаться слишком коротким по сравнению с только что приведенным, в него включены наиболее востребованные технологии, на основе которых строятся решения для облачных вычислений. Изучив эти технологии, можно переходить к изучению более сложных служб.

Amazon EC2

Служба Amazon EC2 стала доступной в 2006 г бета-пользователям, а на рынок вышла в 2008. EC2 — движущая сила, стоящая за динамическими вычислительными возможностями облака Amazon. Эта служба предоставляет виртуальные серверы, которые можно создавать «на лету» и на которые можно устанавливать доступные операционные системы и среды. В сущности, EC2 — это то, благодаря чему облако работает.

Виртуализация в EC2 использует открытую технологию Xen и позволяет виртуализировать и настраивать множество аспектов оборудования. Платформа виртуализации Xen, созданная компанией XenSource (позже купленной Citrix), позволяет гостевым операционным системам, таким как Linux, Windows и Solaris, работать параллельно в виртуальных машинах на одном и том же оборудовании.

Виртуальная машина в EC2 называется экземпляром, и к ней можно подключаться и администрировать ее, как обычную ОС на «реальном» ПК.

Одна из интересных возможностей виртуализации Xen и EC2 — поддержка виртуальных экземпляров как 32- так и 64-разрядных процессоров. Ядро процессора называется вычислительной единицей (computational unit, CU). Это не только «квант» вычислительных ресурсов, но оплачиваемая единица ресурсов. Чем больше CU вы используете, тем дороже обойдется экземпляр. Следовательно, ваша задача — использовать минимум CU, позволяющий выполнять поставленные задачи. Типы экземпляров перечислены в табл. 14-1.

Табл. 14-1. Типы экземпляров

Тип	Процессоры	Память	Локальное хранилище	Платформа	Интенсивность ввода-вывода	Название
Small	1 CU EC2	1,7 Гб	160 Гб на экземпляр (150 Гб + корневой раздел 10 Гб)	32-разрядная	Умеренная	m1.small
Large	4 CU EC2 (2 виртуальных ядра по 2 CU в каждом)	7,5 Гб	850 Гб на экземпляр (2 x 420 Гб + корневой раздел 10 Гб)	64-разрядная	Высокая	m1.large
Extra Large	8 CU EC2 (4 виртуальных ядра по 2 CU в каждом)	15 Гб	1690 Гб на экземпляр (4 x 420 Гб + корневой раздел 10 Гб)	64-разрядная	Высокая	m1.xlarge

Табл. 14-1. (*окончание*)

Тип	Процессоры	Память	Локальное хранилище	Платформа	Интенсивность ввода-вывода	Название
High-CPU Medium	5 CU EC2 (2 виртуальных ядра по 2,5 EC2 в каждом)	1,7 Гб	350 Гб на экземпляр (340 Гб + корневой раздел 10 Гб)	32-разрядная	Умеренная	c1.medium
High-CPU extra large	20 CU EC2 (8 виртуальных ядер по 2,5 CU в каждом)	7 Гб	1690 Гб на экземпляр (4 x 420 Гб + корневой раздел 10 Гб)	64-разрядная	Высокая	c1.xlarge
High-memory extra large	6,5 CU EC2 (2 виртуальных ядра по 2,5 CU в каждом)	17,1 Гб	420 Гб на экземпляр (1 x 420 Гб)	64-разрядная	Умеренная	m2.xlarge
High-memory double extra large	13 CU EC2 (4 виртуальных ядра по 3,25 CU в каждом)	34,2 Гб	850 Гб на экземпляр (1 x 840 Гб + корневой раздел 10 Гб)	64-разрядная	Высокая	m2.2xlarge
High-memory quadruple extra large	26 CU EC2 (8 виртуальных ядер по 3,25 CU в каждом)	68,4 Гб	1690 Гб на экземпляр (2 x 840 Гб + корневой раздел 10 Гб)	64-разрядная	Высокая	m2.4xlarge

При работе с ресурсами Amazon оплачивается время, в течение которого пользователь занимал виртуальную машину, наценочным коэффициентом является тип экземпляра. Физически экземпляры могут быть разбросаны по всему миру. Подробнее о стоимости и конфигурации экземпляров см. по адресу *http://aws.amazon.com/ec2/#pricing*.

Экземпляры EC2 используют AMI-образы. AMI состоит из операционной системы и дополнительного ПО. Amazon предлагает множество готовых AMI (см. каталог), чтобы облегчить начало работы с EC2, пример — AMI LAMP, включающий Linux, Apache, MySQL и PHP/Perl/Python.

Amazon S3

В 2006 году Amazon создала S3 — свою основную веб-службу для онлайнового хранения данных. S3 предоставляет разработчикам простое и безопас-

ное постоянное онлайновое хранилище практически бесконечной емкости. На высоком уровне технология S3 аналогична (как минимум по концепции) сети хранения данных (SAN) в том, что ресурсы доступны для любого подключенного устройства. Плата зависит от объема хранимых данных и ширины канала, занимаемого при сохранении и получении данных.

В отличие от традиционных файловых систем, использующих структуру каталогов, в S3 используется механизм хранения объектов, называемый *сегментами* (buckets), которым назначаются общедоступные имена. Т.е. любой пользователь может записать что-то в ваш сегмент, которому дано имя, например, по названию вашей компании. Так что остерегайтесь использования распространенных имен типа «база_данных» или «документы». Большинство пользователей добавляют в имена сегментов свои доменные имена, но даже это не гарантирует защиту от перезаписи данных в том же сегменте.

Элементы, хранящиеся в сегментах, называются *объектами*. Каждый объект может быть размером от 1 байта до 5 Гб. Сегменты и содержащиеся в них объекты физически расположены в одном или двух регионах, называемых *зонами доступности* (одна зона охватывает ЦОД Северной Америки, другая — ЦОД в Европе), но обращаться к ним можно из любой точки мира. Более того, Amazon предоставляет веб-сервисы, позволяющие использовать S3 почти в любом веб-приложении.

Механизм S3 не предназначался для быстрого чтения-записи и лучше всего подходит для хранения архивной информации, например, для пользовательских AMI, массового копирования или хранения резервных копий данных. Для размещения активных БД лучше использовать другие технологии.

Amazon EBS

Amazon выпустила EBS в 2008 году. Для облачных вычислений это был гигантский шаг вперед. EBS — виртуальное блочное хранилище, похожее на дисковый накопитель. Оно не только имеет производительность типичного блочного устройства, предоставляя возможность быстрого чтения-записи, но и является независимым от работающего экземпляра. Это действительно важно, так как в прошлом пользователям приходилось выводить свои данные из S3 или из облака и загружать их в новый экземпляр. Но экземпляр — сущность временная, и когда он уничтожался (что могло случиться неожиданно и без предупреждения), все изменения терялись. Поэтому до появления EBS приходилось часто выгружать данные в S3 либо использовать специальные утилиты, такие как диспетчер томов.

С появлением EBS пользователи получили возможность создавать независимые устройства (называемые *томами* или *хранилищами EBS*) и прикреплять их к любому работающему экземпляру как внешний жесткий диск. Для разработчиков тома EBS выглядят стандартными блочными устрой-

ствами, чей размер варьируется от 1 Гб до 1 Тб. Размер томов EBS, как и их аналогов SAN, можно изменять «на лету», используя моментальные снимки. Это очень удобно, так как позволяет наращивать дисковое пространство с ростом объема приложений и данных.

Можно также использовать несколько томов EBS для чередования, чтобы повысить скорость ввода-вывода. Еще один плюс в том, что тома EBS реплицируются между зонами доступности Amazon EC2. Это означает, что даже при сбое в текущей зоне данные останутся доступными. Таким образом, надежность EBS выше, чем у традиционных дисковых систем хранения. Однако тома EBS, которые можно присоединить к экземпляру EC2, должны находиться в той же зоне доступности, что и сам экземпляр.

EBS также поддерживает резервное копирование на S3 с использованием мгновенных снимков. Каждая резервная копия является разностным снимком: сохраняются только блоки, измененные с момента последнего снимка. Снимки —эффективный способ создания долговечных резервных копий, оптимальный для БД MySQL.

Тома EBS, поддерживающие все функции обычных дисков и многое другое, являются отличным решением для хранения файлов БД MySQL. Вы получаете возможность использования мгновенных снимков и беспрецедентную надежность.

Как это работает

Прежде чем перейти к подробностям настройки учетной записи AWS и запуску экземпляров AMI, важно понять принципы взаимодействия с облаком Amazon и концепцию реализации виртуализированных серверов. На рис. 14-2 показана концептуальная модель реализации и выполнения экземпляров EC2.

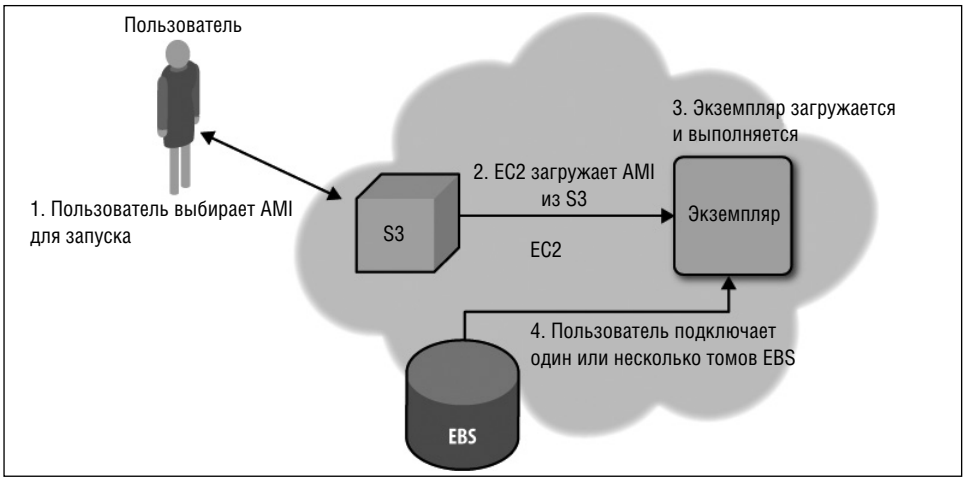


Рис. 14-2. Превращение AMI в экземпляры серверов в EC2

Утилиты облака Amazon

Существует два типа утилит, позволяющих приобретать услуги, запрашивать ресурсы и управлять ими: с графическим интерфейсом и с интерфейсом командной строки. В сущности, есть два типа графического интерфейса. Amazon предоставляет веб-консоль для доступа к своим службам, и существует несколько подключаемых модулей для веб-браузеров, которые можно использовать в качестве альтернативы. Также Amazon предоставляет набор утилит на основе EC2 API (их называют утилитами командной строки EC2).

Консоль Amazon

Amazon предоставляет веб-интерфейс для взаимодействия со всеми продуктами — AWS Management Console. Найти эту консоль можно по адресу *https://console.aws.amazon.com/ec2/home*.

Для ее запуска потребуется учетная запись AWS (см. ниже). Сначала мы расскажем о консоли, чтобы вы имели лучшее представление о работе AWS и EC2.

При помощи ее интерфейса можно создавать экземпляры, подключаться к ним, создавать тома EBS и подключать их к экземплярам, а также многое другое. Как вы еще увидите, это стандартный механизм для работы с EC2 и другими продуктами AWS. На рис. 14-3 показан интерфейс AWS Management Console для типичного пользователя.

На рис. 14-3 и других рисунках из этой главы скрыта контекстная информация, которая не важна для раскрытия темы.

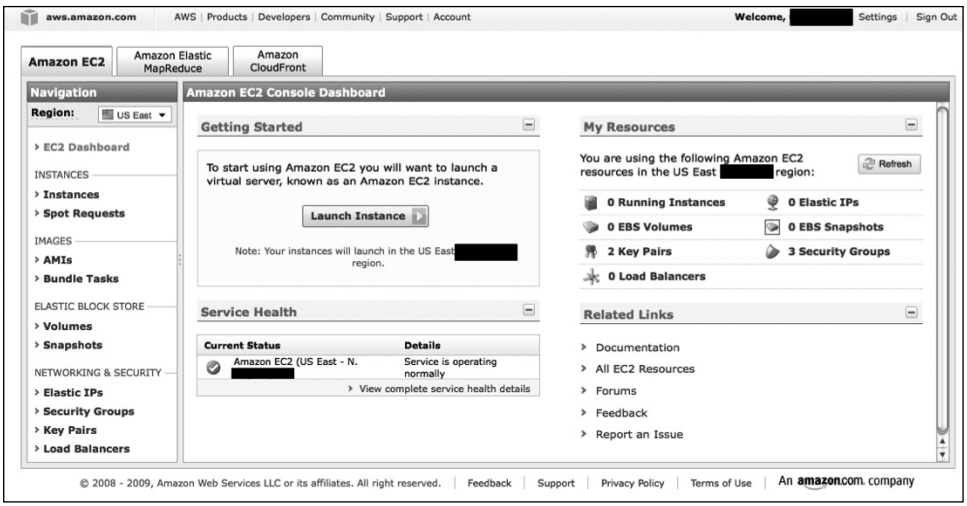


Рис. 14-3. AWS Management Console

На экране в центре слева находится кнопка **Launch Instance (Запустить экземпляр)**. Легко догадаться, что это отправная точка создания экземпляров EC2. В левой части экрана находятся ссылки на многие ресурсы облака AWS, такие как тома (EBS), эластичные IP, балансировщики нагрузки и т. д.

В верхней части находятся три вкладки, предоставляющие доступ к различным группам служб облака. По умолчанию открывается вкладка Amazon EC2 (показана). Далее идет вкладка Amazon Elastic MapReduce, позволяющая настраивать и запускать решения с использованием распределенных вычислений. Третья вкладка — Amazon CloudFront, позволяющая управлять веб-содержимым. Подробнее о создании экземпляров EC2 — в следующих разделах.

Подключаемые модули для браузеров

Если хотите управлять облаком с веб-страниц, можете пользоваться подключаемым модулем для Mozilla Firefox, носящим название Elasticfox (*http://developer.amazonweb services.com/connect/entry.jspa?externalID=609*).

Elasticfox — графический веб-интерфейс для доступа ко всем API EC2 и управления всеми аспектами экземпляров EC2 — от создания экземпляров до создания томов и подключения их к экземплярам. Elasticfox предоставляет больше возможностей, чем AWS Management Console, и к тому же позволяет быстро выполнить все необходимые действия (см. рис. 14-4).

Информацию об установке и настройке Elasticfox для получения доступа к EC2 можно найти по адресу *http://developer.amazon webservices.com/connect/entry.jspa?externalID=1797*.

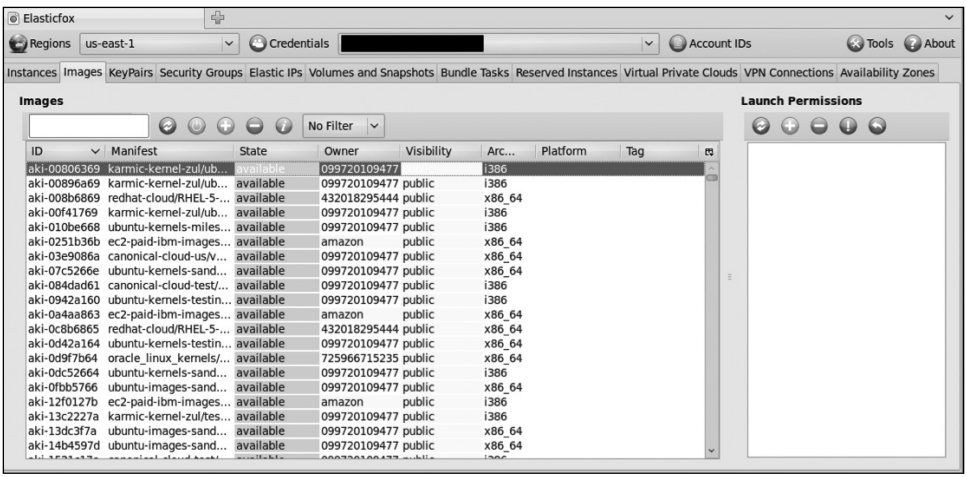


Рис. 14-4. Подключаемый модуль Elasticfox для Firefox

Альтернативы AWS Management Console с веб-интерфейсом создают и другие поставщики. Один из лучших продуктов — проект shareVM. Подробнее об этом решении см. по адресу *http://blog.sharevm.com/2009/01/09/web-based-ec2-console-alternative-to-elasticfox*.

Утилиты командной строки EC2

Взаимодействовать с EC2 можно также посредством утилит командной строки. Существует два набора: утилиты API и утилиты AMI. Первые взаимодействуют с EC2 и включают утилиты для запуска экземпляров, создания и присоединения томов, управления группами безопасности и т. д. Вторые позволяют создавать и управлять AMI.

Утилиты EC2 API можно загрузить по ссылке *http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351&categoryID=88*. Для установки и настройки следуйте инструкциям.

Документация по этим утилитам включена в документацию Getting Started (Приступая к работе), которую можно найти по адресу *http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference*. Руководство пользователя EC2 (*http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-ug.pdf*) также содержит документацию по утилитам командной строки.

Не пожалейте времени, чтобы загрузить и установить утилиты EC2 API. Они позволяют делать очень многое в EC2. Перечислим наиболее востребованные утилиты:

- *ec2-add-key-pair* — создает пару ключей SSH.
- *ec2-run-instances* — запускает экземпляры EC2. Необходимо указать как минимум имя образа и пару ключей. Можно запускать несколько экземпляров одновременно.
- *ec2-describe-images* — возвращает список доступных образов. Выходные данные включают ID образа, расположение образа в S3 и указание на то, доступен ли образ для запуска. Имеется несколько параметров, сужающих область поиска.
- *ec2-stop-instances* — останавливает или приостанавливает экземпляры. Одновременно можно остановить несколько экземпляров.
- *ec2-start-instances* — запускает или возобновляет работу экземпляров. Можно запустить несколько экземпляров одновременно.
- *ec2-terminate-instances* — уничтожает экземпляр. Можно уничтожить несколько экземпляров одновременно.

Это лишь несколько часто используемых команд. Также существуют команды для работы с группами безопасности, ключами, образами, томами и многим другим. Подробнее см. в руководстве пользователя.

Утилиты EC2 AMI можно загрузить по ссылке *http://developer.amazonwebservices.com/connect/entry.jspa?externalID=368&categoryID=88*. Для установки утилит следуйте инструкциям.

Если хотите работать с образами и создавать образы с собственными настройками, рекомендуем изучить документацию по следующим ссылкам:
http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide
http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference

Приступаем к работе

Продукты Amazon AWS платные, поэтому необходимо создать учетную запись и указать действительный способ оплаты. Начать работу с облаком Amazon очень легко. И если при обучении вы будете использовать небольшие экземпляры, и не будете надолго оставлять их работающими, то сможете научиться работать с облаком, заплатив совсем немного. Например, на подготовку примеров для этой главы мы потратили меньше, чем стоит обед в популярном ресторане быстрого питания. Это гораздо дешевле покупки небольшого сервера!

Amazon взимает плату за многие свои продукты, включая машинное время и размер хранилища. Закончив работу, обязательно уничтожьте неиспользуемые экземпляры и освободите временные хранилища. Хотя эта плата невелика, платить придется, даже если вы не работаете с активными ресурсами. Чтобы было понятнее, скажем так: если вы оставили включенным дома свет и уехали в отпуск на две недели, не стоит удивляться размеру счета за электричество.

В следующих разделах мы покажем, как получить учетную запись AWS, запустить экземпляр, создать том диска и подключить его к запущенному экземпляру. Затем мы покажем, насколько легко использовать MySQL в облаке.

Получение учетной записи

Первое, что вам понадобится — учетная запись Amazon AWS. Чтобы использовать базовые службы облака, необходимо создать учетную запись, зарегистрироваться для доступа к EC2 и зарегистрироваться для доступа к S3 и EBS. К счастью, сделать это очень просто:

1. Зайдите на сайт AWS (*http://aws.amazon.com*).
2. Перейдите по ссылке **Create an AWS Account (Создать учетную запись AWS)**.
3. Щелкните кнопку **Sign Up Now (Зарегистрироваться сейчас)**.
4. Введите предпочитаемый идентификатор регистрации (например, свой адрес электронной почты) и установите флажок **I am a new user (Я новый пользователь)**. Затем щелкните кнопку **Sign in using our secure server (Зарегистрироваться, используя безопасный сервер)**.

Если у вас уже есть учетная запись для доступа к интернет-магазину Amazon, можете использовать ее, а не создавать новую.

5. Введите и подтвердите пароль. Укажите свой адрес электронной почты и щелкните кнопку **Create account (Создать учетную запись)**.
6. Введите подробную информацию о своей учетной записи, включая адрес для отправки счетов и контактную информацию. Можно ввести адрес своего веб-сайта. Для продолжения необходимо принять соглашение пользователя AWS. Прочтите соглашение, установите флажок согласия, введите проверочные символы и щелкните кнопку **Continue (Продолжить)**.

7. На указанный адрес электронной почты придет письмо с дальнейшими инструкциями.
8. Зайдите на сайт под созданной учетной записью. На странице Account (Учетная запись) щелкните ссылку **Payment Method (Способ оплаты)** в правой части главной страницы и введите необходимые данные.


Чтобы зарегистрироваться для доступа к Amazon EC2 (или любому другому продукту), зайдите на страницу Products и щелкните ссылку в синем поле. Затем щелкните кнопку Sign up for в правом верхнем углу страницы и следуйте инструкциям. Повторите то же самое для S3 и любых других служб и продуктов, которые хотите использовать.

Получение учетных данных

Доступ к EC2 и другим продуктам AWS требует использования протоколов безопасности. Сюда входят регистрационное имя AWS и пароль, ключи доступа к AWS API, сертификаты X.509 для доступа к AWS API по протоколу SOAP и пары ключей для доступа к EC2 и CloudFront. При регистрации проверяйте требования продуктов, которые хотите использовать, чтобы убедиться, что имеете необходимые учетные данные.

Amazon рекомендует создавать отдельную папку для хранения закрытых ключей. Создайте такую папку и защитите ее от случайного обнаружения, прежде чем приступите к загрузке закрытых ключей и других учетных данных.

Логин и пароль Amazon

Чтобы использовать AWS Management Console и выполнять действия с учетной записью, вам потребуется логин и пароль, введенные при регистрации доступа к AWS.

ID ключа доступа и ключ секретного доступа

Для использования API-запросов (например, для поиска образов) и многих утилит с GUI (например, Elasticfox) потребуется идентификатор ключа доступа и ключ секретного доступа. Эти учетные данные запрашиваются чаще всего. Для создания ключей доступа выполните следующее:

1. Зайдите на главную страницу AWS (*http://aws.amazon.com*).
2. Перейдите на вкладку **Account (Учетная запись)**.
3. Щелкните ссылку **Security Credentials (Учетные данные безопасности)**.


Если вы еще не зарегистрировались, вам будет предложено пройти регистрацию.

4. В разделе Access Credentials (Учетные данные доступа) щелкните ярлык вкладки **Access Key (Ключи доступа)**.
5. Если вы создали ключи доступа при регистрации учетной записи AWS, они будут указаны на открывшейся странице. Чтобы просмотреть ключ безопасного доступа, щелкните ссылку **Show (Показать)**, расположен-

ной рядом с нужным ключом доступа. Можете скопировать этот ключ в безопасный (с ограниченным доступом) файл на своем компьютере.

6. Если ключей доступа нет или вы хотите создать новый ключ, щелкните ссылку **Create a new access key (Создать новый ключ доступа)**.

SOAP и утилиты командной строки EC2

Для работы с утилитами командной строки EC2 и подключения по протоколам SOAP требуется сертификат X.509 и закрытый ключ. Ключи доступа следует периодически менять, чтобы снизить угрозы безопасности. Amazon автоматически меняет ключи каждые 90 дней. Чтобы создать и просмотреть сертификаты X.509 выполните следующее:

1. Перейдите на главную страницу AWS (*http://aws.amazon.com*).
2. Перейдите на вкладку **Account (Учетная запись)**.
3. Щелкните ссылку **Security Credentials (Учетные данные безопасности)**.

Если вы еще не зарегистрировались, будет запрошена регистрация.

4. В секции Access Credentials (Учетные данные доступа) щелкните ярлык вкладки X.509 Certificates. Появится список имеющихся сертификатов, которые можно активировать или деактивировать.
5. Чтобы создать новый сертификат и закрытый ключ, щелкните ссылку **Create a new certificate (Создать новый сертификат)**. Будет создан новый сертификат и начнется загрузка закрытого ключа.

Пользователь может иметь два сертификата. Если у вас уже имеется два сертификата, удалите один из них, чтобы можно было создать новый.

6. Когда откроется диалоговое окно, щелкните кнопку **Download Private Key File (Загрузить файл закрытого ключа)** и сохраните закрытый ключ в ранее созданную папку.
7. Щелкните кнопку **Download X.509 Certificate (Загрузить сертификат X.509)** и сохраните сертификат в ту же папку.
8. Когда закончите, щелкните кнопку **Close (Закрыть)**.

Теперь у вас должно быть два файла, имя одного из которых начинается с *pk-*, а второго с *cert-*. Это файлы закрытого ключа и сертификата.

Учетные данные для работы с CloudFront

Для доступа к CloudFront применяется специальный механизм, использующий пары ключей. Эти ключи называют просто парами ключей, из-за чего их часто путают с парами ключей, используемыми для доступа к экземплярам. Пары ключей CloudFront работают как сертификаты X.509 в том смысле, что обращаться к ним нужно со своей страницы учетных данных безопасности и активными могут быть только две из них. Для создания и просмотра пар ключей CloudFront выполните следующее:

1. Перейдите на главную страницу AWS (*http://aws.amazon.com*).
2. Перейдите на вкладку **Account (Учетная запись)**.
3. Щелкните ссылку **Security Credentials (Учетные данные безопасности)**.


Если вы еще не зарегистрировались, придется это сделать.

5. В секции Access Credentials (Учетные данные доступа) щелкните ярлык вкладки **Key Pairs (Пары ключей)**. Появится список имеющихся пар ключей, которые можно активировать или деактивировать.
6. Чтобы создать новую пару, щелкните ссылку **Create a new key pair (Создать новую пару ключей)**. Будет создан новый сертификат и начнется загрузка закрытого ключа.


Пользователь может иметь две пары ключей. Если у вас уже имеется две пары, для создания новой одну из них придется удалить.

Учетные данные для работы с экземплярами

Для доступа к экземплярам требуется пара ключей SSH. Эти ключи создаются при помощи AWS Management Console. Можно создать столько ключей, сколько требуется. Рекомендуется создавать по одной паре на экземпляр, чтобы снизить риск случайного или намеренного раскрытия.


Создать пару ключей SSH потребуется как минимум один раз — для запуска первого экземпляра.

Эти ключи позволяют получать доступ к запущенному экземпляру, избавляя от необходимости запоминать и вводить пароли. Один из ключей внедряется в экземпляр, а второй ключ предъявляется при входе для аутентификации. Парам ключей можно давать имена, чтобы ими было проще управлять. При запуске образа необходимо указать имя пары ключей.

Чтобы создать новую пару ключей SSH, выполните следующее:

1. Перейдите на главную страницу AWS (*http://aws.amazon.com*).


Если вы еще не зарегистрировались, вам придется сделать это.

2. Щелкните ссылку **Key Pairs (Пары ключей)** в левой части страницы.
3. Щелкните кнопку **Create Key Pair (Создать пару ключей)** и введите уникальное имя.
4. Щелкните кнопку **Create (Создать)**. Пара ключей будет создана, и начнется загрузка закрытого ключа. Сохраните ключ в папку, созданную ранее.
5. Щелкните кнопку **Close (Закрыть)**.

Другие учетные данные

В нижней части страницы с учетными данными безопасности указаны учетные данные регистрации и идентификаторы учетной записи. Если хотите

изменить адрес электронной почты или пароль, это можно сделать на этой странице. Если хотите организовать взаимодействие с другими пользователями AWS, нужно внести их учетные записи в идентификаторы учетных записей, а владельцы этих учетных записей должны внести вашу учетную запись в свои идентификаторы. Идентификаторы учетных записей можно найти в нижней части этой страницы.

Можно также зарегистрироваться для использования дополнительной службы AWS Multi-Factor Authentication, использующей генератор ключей для обеспечения повышенной защиты учетной записи. Если при работе с AWS важно обеспечить высокую безопасность, подробно изучите возможности этой службы. Кнопки и ссылки на этой странице помогут вам определить, следует ли использовать эту службу.

Запуск экземпляра при помощи AMS Management Console

Запустить экземпляр в EC2 очень просто. На первый взгляд может показаться, что это не так, но Amazon проделала большую работу, чтобы максимально упростить освоение.

Чтобы запустить экземпляр в EC2, зайдите на страницу *http://aws.amazon.com* и щелкните ссылку Sign in to the AWS Management Console. В центре экрана вы увидите кнопку **Launch Instance (Запустить экземпляр)** (рис. 14-5).

Рис. 14-5. Запуск экземпляра в EC2

Теперь нужно выбрать образ для запуска. Откроется мастер запроса экземпляров (Request Instances Wizard), на первой странице которого содержится список доступных образов, предлагаемых Amazon. Вы увидите несколько вкладок: вкладку Quick Start, содержащую список образов, предлагаемых Amazon, с готовыми конфигурациями для выполнения распространенных задач, вкладку MyAMIs (Мои AMI), появляющуюся, если созданы какие-либо AMI, и Community AMIs (AMI от сообществ), содержащую список AMI, созданных разработчиками.

Некоторые AMI от сообществ могут быть платными. Обычно это AMI, содержащие специализированные конфигурации и приложения. Убедитесь, что используете бесплатные AMI, чтобы избежать лишних расходов, или проверьте стоимость перед использованием.

Для работы с MySQL хорошо подойдет образ с меткой Web Starter, так как в него включено все, что необходимо для начала работы с MySQL в облаке. В настоящее время набор LAMP работает в ОС Fedora. На рис. 14-6 показана страница для выбора образа.

Рис. 14-6. Выбор образа для запуска

Обратите внимание на то, что в списке Quick Start есть также сервер Windows. Большинство образов содержат Linux, но Windows-разработчики повышают популярность образов с Windows. Осталось дождаться, когда кто-нибудь создаст образ с Mac OX X (возможно, это не произойдет никогда, учитывая проприетарную природу этой платформы). В случае с Windows пользователю может понадобиться ввести собственные ключи продукта, чтобы использовать образы.

Выбрав образ, который хотите запустить, щелкните кнопку Select (Выбрать) справа от имени этого образа. На следующей странице (рис. 14-7) выберите число экземпляров для запуска (да, можно запустить несколько экземпляров сразу) и тип экземпляра. Типы экземпляров перечислены в табл. 14-1. Для большинства учебных задач достаточно типа small.

Также нужно выбрать зону доступности. Выбирайте ближайшую к вашему географическому положению зону. Наконец, необходимо выбрать между запуском экземпляра и запросом аренды образа (spot image).

Запрос аренды образа — это запрос экземпляра, который представляется с повременной оплатой по механизму аукциона. Стоимость использования растет, если пользователи предлагают за него большую цену, и падает, если меньшую, так что можно предложить свою цену и ждать, когда произойдет

спад и стоимость опустится ниже указанного порога. Это замечательный способ выполнения несрочных вычислений (которые можно выполнить в любое время в течение определенного срока) по приемлемой цене.

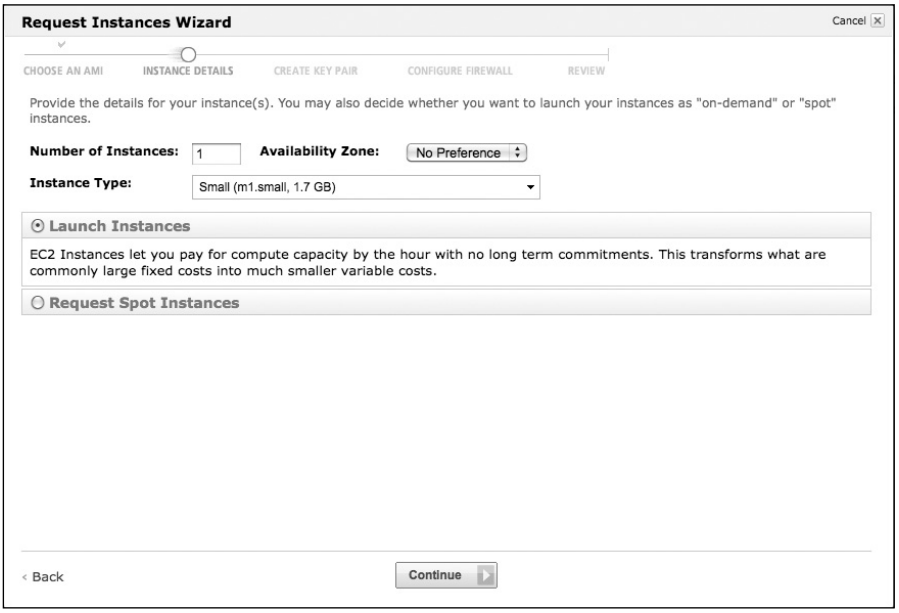


Рис. 14-7. Сведения об экземпляре

Выбрав число экземпляров, тип экземпляров и зону доступности, щелкните кнопку **Continue (Продолжить)** в нижней части страницы.

Виртуальные ресурсы, создаваемые в AWS, имеют идентификаторы, начинающиеся с сокращенного именования типа. Например, идентификаторы экземпляров начинаются с *i-*, томов — с *vol-*, мгновенных снимков — со *snap-*. Это позволяет сразу понять, с чем вы работаете.

На следующей странице (рис. 14-8) указывается ID ярда и ID диска ОЗУ. В большинстве случаев следует оставлять значения по умолчанию. Подробнее о настройках ядер и дисков можно узнать по ссылкам, представленным на этой странице. Здесь также можно включить мониторинг облака и добавить комментарии по поводу запуска.

Указав желаемые параметры, щелкните кнопку **Continue (Продолжить)**. На следующей странице (рис. 14-9) укажите пары ключей SSH, которые будут использоваться для доступа к этому экземпляру. Можно использовать существующую пару, создать новую или продолжить без ключей (не рекомендуется). Чтобы создать пару ключей и присвоить ей имя, щелкните кнопку **Create a new key pair (Создать новую пару ключей)**. Когда закончите, щелкните кнопку Continue.

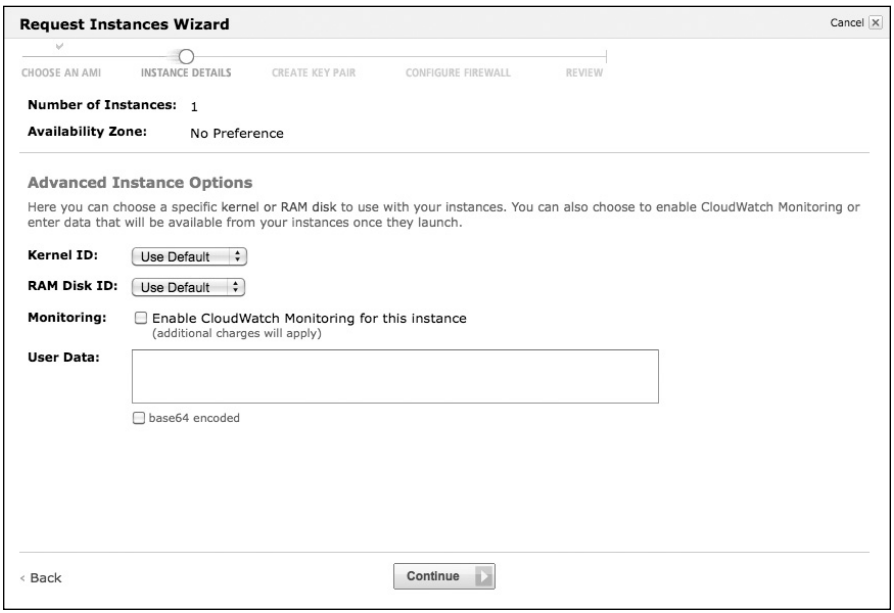


Рис. 14-8. Дополнительные параметры экземпляра

Рис. 14-9. Создание пары ключей

На следующей странице необходимо выбрать группу безопасности или создать новую группу, которая будет использоваться для данного экземпляра. Эти параметры определяют параметры брандмауэра для этого экземпляра. Сначала установите переключатель Create a new security group (Создать но-

вую группу безопасности). Введите имя группы безопасности и ее описание (рекомендуем вводить информативный текст, как показано на рис. 14-10).

Затем определите разрешенные типы подключения. По умолчанию разрешается доступ к порту SSH TCP 22 (требуется для подключения с использованием пары ключей EC2 SSH) и к другим портам, необходимым для выбранного образа. Например, образ LAMP разрешает доступ к портам HTTP TCP 80 и MySQL TCP 3306. На рис. 14-10 показан пример стандартных параметров безопасности для образа LAMP.

Выбрав параметры доступа, щелкните кнопку **Continue**.

Рис. 14-10. Настройка доступа к сети

На следующей странице (рис. 14-11) можно просмотреть все выбранные параметры и при необходимости внести изменения. Когда будете готовы, щелкните кнопку **Launch (Запустить)**, чтобы запустить экземпляр.

Можно также воспользоваться кнопкой Back (Назад), чтобы перейти к предыдущим страницам.

Появится уведомление о состоянии экземпляра, в котором говорится, что экземпляр (или экземпляры) будет отображаться в секции My Instances (Мои экземпляры) AWS Management Console. Когда вернетесь в консоль, щелкните ссылку **running instances (запущенные экземпляры)** в секции My Resources (Мои ресурсы). Вы увидите свой экземпляр на различных этапах запуска. Когда экземпляр будет запущен, в списке появятся сведения о нем.

На рис. 14-12 показано, как отображаются запущенные образы. Обратите внимание на то, что показаны сведения о подключении, а также описание образа, идентификатор AMI, его состояние и тип экземпляра.

Рис. 14-11. Последняя страница для запуска образа

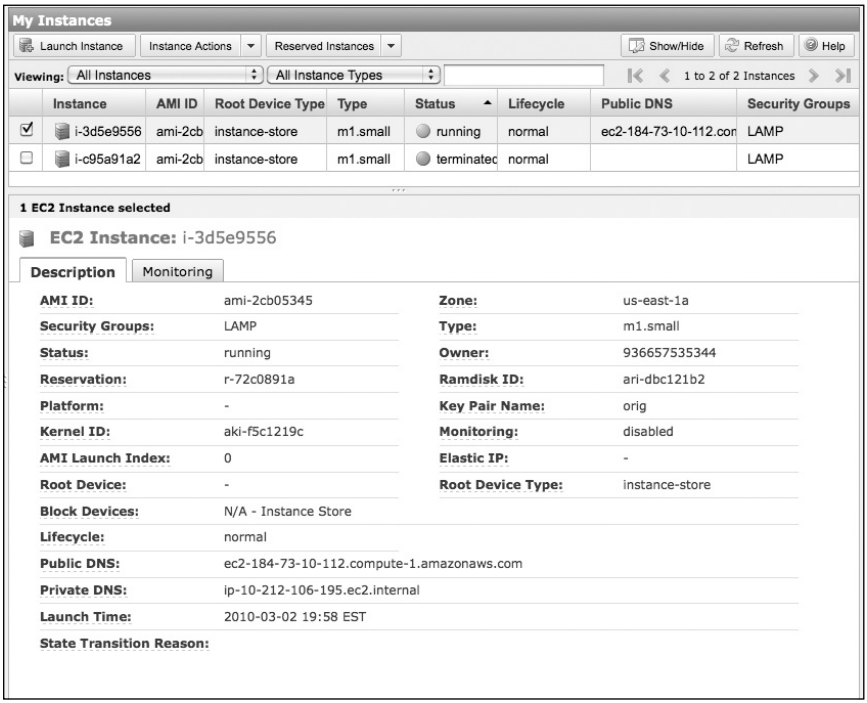


Рис. 14-12. Страница My Instances

На этой странице можно выполнять различные действия с экземплярами: запускать и останавливать (приостанавливать), уничтожать и даже подклю-

чаться к ним. Прежде чем выполнить какое-то действие, необходимо установить флажок рядом с теми экземплярами, которые это действие должно затронуть.

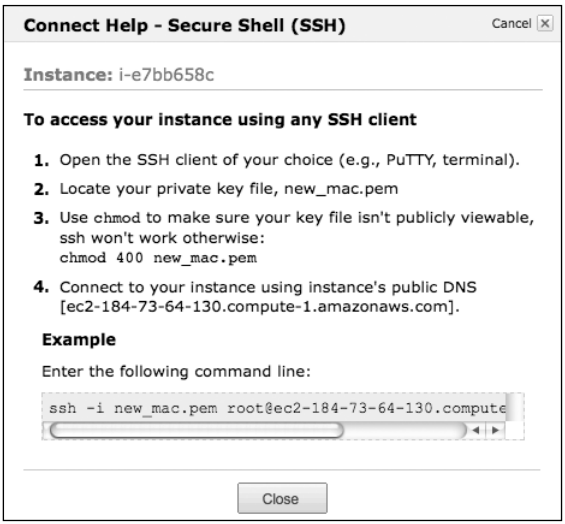


Рис. 14-13. Диалоговое окно Connect Help

Теперь попробуем подключиться к экземпляру. Установите флажок рядом с экземпляром, а затем выберите **Connect (Подключиться)** в раскрывающемся списке **Instance Actions (Действия с экземплярами)**. Откроется диалоговое окно (рис. 14-13), отображающее команды, позволяющие подключиться к экземпляру. Откройте терминал или командную строку и введите указанные команды. Можете щелкнуть кнопку **Close (Закрыть)**, чтобы закрыть диалоговое окно.

Убедитесь, что обеспечили безопасность ключей SSH. Например, для этого можно выполнить команду CHMOD 0400 <файл_ключа>.

Когда подключитесь к экземпляру, увидите примерно следующий текст сеанса терминала (SSH):

```
Chucks-MacBook-Pro:~ Chuck$ ssh -i ./ec2_credentials/new_mac.pem root@ec2-
184-73-64-130.compute-1.amazonaws.com

      __| __|_ ) Fedora 8
      _| (   /  32-bit
      |\___|___|

Welcome to an EC2 Public Image
                    :-)

   Base
```

```
--[ see /etc/ec2/release-notes ]--

[root@ip-10-245-114-64 ~]#
```

Так как у вас есть корневой доступ, вы можете выполнять со своим экземпляром любые действия. Когда закончите экспериментировать, можете уничтожить экземпляр. Для этого вернитесь на страницу My Instances консоли AWS Management Console, установите флажок рядом с экземпляром, выберите в раскрывающемся списке **Instance Actions команду Terminate (Уничтожить)** и подтвердите выбранное действие.

С экземплярами можно выполнять много различных действий: просматривать журнал системы, запускать дополнительные экземпляры, перезагружать, останавливать и запускать, включать и отключать мониторинг и т. д.

После уничтожения экземпляр будет отображаться с состоянием terminated (уничтожен). Перед выходом из консоли управления AWS рекомендуем всегда проверять, все ли ненужные экземпляры уничтожены. Экземпляры необходимо уничтожать, так как в другом состоянии экземпляры считаются активными и за них будет взиматься плата; вы будете неприятно удивлены, когда получите счет.

Запуск экземпляра при помощи утилит EC2 API

Выше мы показали, насколько легко запускать экземпляры при помощи AWS Management Console. Теперь посмотрим, как запускать экземпляры, используя утилиты EC2 API.

Если вы уже установили эти утилиты, можете повторить предложенные ниже примеры, что запустить образ и подключиться к нему.

Первый шаг — получить список доступных образов. Это выполняется при помощи команды ec2-describe-images. В следующем примере мы используем команду grep для поиска образов, поддерживающих MySQL:

```
Chucks-MacBook-Pro:~ Chuck$ ec2-describe-images -o self -o amazon | grep mysql
IMAGE  ami-225fba4b  ec2-public-images/fedora-core4-apache-
   mysql-v1.07.manifest.xml
   amazon  available public     i386  machine        instance-store
IMAGE  ami-25b6534c  ec2-public-images/fedora-core4-apache-mysql.
   manifest.xmamazon
   available  public i386  machine           instance-store
IMAGE  ami-255fba4c  ec2-public-images/fedora-core4-mysql-v1.07.
   manifest.xmlamazon
   available  public i386  machine           instance-store
IMAGE  ami-22b6534b  ec2-public-images/fedora-core4-mysql.
   manifest.xml
   available  public i386  machine           instance-storee
```

На системах Windows эти команды можно выполнять при помощи Cygwin. Подробнее см. в руководстве пользователя Amazon EC2.

Получив список доступных образов, мы можем выбрать образ LAMP и запустить его командой ec2-run-instances. Здесь мы указываем имя образа (AMI ID) из только что полученного списка, а также загруженный файл ключа SSH. Ниже показано возвращенное этой командой сообщение, в котором указывается ID экземпляра, AMI ID, состояние (pending), используемый ключ, тип экземпляра, дата и время, регион, состояние мониторинга и тип хранилища:

```
Chucks-MacBook-Pro:~ Chuck$ ec2-run-instances ami-225fba4b -k new_mac
RESERVATION  r-2249194a  936657535344  default
INSTANCE       i-75af711e  ami-225fba4b        pending  new_macm1.small
   2010-03-09T02:13:27+0000  us-east-1d            monitoring-disabled
   instance-store
```

Можете проверить состояние своих экземпляров, выполнив команду ec2-describe-instances. Будет показана та же информация, что и при выполнении предыдущей команды, но с обновленными состояниями:

```
Chucks-MacBook-Pro:~ Chuck$ ec2-describe-instances
RESERVATION  r-2249194a  936657535344  default
INSTANCE      i-75af711e  ami-225fba4b        pending  new_macm1.small
   2010-03-09T02:13:27+0000  us-east-1d         monitoring-disabled
   instance-store
```

Если экземпляр запущен, вы увидите текст, аналогичный показанному ниже. Обратите внимание на то, что в нем указан IP-адрес экземпляра. Будет показано два IP-адреса. Первый — общедоступный IP-адрес экземпляра. Его можно использовать для входа в экземпляр, как и имя экземпляра. Второй — закрытый IP-адрес, который можно использовать внутри облака для взаимодействия с другими экземплярами (своими или чужими).

```
Chucks-MacBook-Pro:~ Chuck$ ec2-describe-instances
RESERVATION  r-2249194a  936657535344  default
INSTANCE     i-75af711e  ami-225fba4b  ec2-184-73-9-65.compute-1.
   amazonaws.com
   domU-12-31-39-02-EC-E7.compute-1.internal  running  new_mac  0
   m1.small
   2010-03-09T02:13:27+0000  us-east-1d   monitoring-disabled
   184.73.9.65  10.248.243.21        instance-store
```

Чтобы получить доступ к экземпляру, необходимо авторизовать группу безопасности. В следующем примере мы используем ярлык, полученный командой ec2-authorize, чтобы указать группу безопасности по умолчанию и разрешить доступ к порту SSH TCP 22. Полное описание этой команды и примеры настройки групп безопасности при помощи утилит EC2 API можно найти в руководстве пользователя Amazon EC2.

```
Chucks-MacBook-Pro:~ Chuck$ ec2-authorize default -p 22
GROUP     default
```

```
PERMISSION     default  ALLOWS  tcp  22  22  FROM  CIDR
   0.0.0.0/0
```

Теперь мы можем подключиться к нашему экземпляру через SSH.

```
Chucks-MacBook-Pro:~ Chuck$ ssh -i ./ec2_credentials/new_mac.pem
root@ec2-184-73-9-65.compute-1.amazonaws.com

   __| __|_ ) Rev: 2 _|
   (   / ___|\___|___|

Welcome to an EC2 Public Image
                         :-)
   Apache2+MySQL4

   __ c __ /etc/ec2/release-notes.txt

[root@domU-12-31-39-02-EC-E7 ~]# exit
logout

Connection to ec2-184-73-9-65.compute-1.amazonaws.com closed.
```

Успешно! Теперь давайте выйдем из экземпляра и завершим его работу. Для этого выполним команду ec2-terminate-instances, указав ID экземпляра (взятого из результатов команды describe-instances).

```
Chucks-MacBook-Pro:~ Chuck$ ec2-terminate-instances i-75af711e
INSTANCE  i-75af711e  running  shutting-down
Chucks-MacBook-Pro:~ Chuck$ ec2-describe-instances
RESERVATION  r-2249194a  936657535344  default
INSTANCE i-75af711e ami-225fba4b ec2-184-73-9-65.compute-1.amazonaws.com
   domU-12-31-39-02-EC-E7.compute-1.internal shutting-down new_macm1.small
   2010-03-09T02:13:27+0000  us-east-1d        monitoring-disabled
   184.73.9.65  10.248.243.21        instance-store
```

Экземпляр уничтожен. Чтобы убедиться в этом, снова выполните команду describe-instances (можно несколько раз).

```
Chucks-MacBook-Pro:~ Chuck$ ec2-describe-instances
RESERVATION  r-2249194a  936657535344  default
INSTANCE  i-75af711e  ami-225fba4b          terminated  new_mac  0
      m1.small  2010-03-09T02:13:27+0000  us-east-1d
      monitoring-disabled             instance-store
```

Как видите, все задачи по запуску экземпляра, которые можно выполнить при помощи AWS Management Console, можно решить и при помощи утилит EC2 API. Многие пользователи предпочитают команды графическому интерфейсу. Решайте сами, какой вариант вам больше нравится.

Работа с диском

В предыдущем разделе мы говорили о запуске экземпляров, теперь разберемся с дисковыми хранилищами. Мы расскажем, как ведут себя дисковые хранилища (их называют хранилищами экземпляра или томами экземпляра) внутри экземпляра, как выполнять резервное копирование данных и как работать с постоянным хранилищем, создавая том EBS и подключая его к запущенному экземпляру. В конце рассмотрим пример выполнения резервного копирования тома EBS при помощи мгновенных снимков.

Тома EBS создаются быстро и просто, так что, приступая к экспериментам с ними, можете не уничтожать свои запущенные экземпляры — долгого простоя не будет.

Использование хранилища экземпляра

Каждый экземпляр имеет собственный фиксированный объем дискового хранилища, который можно использовать, как вам угодно. Однако данные оттуда следует копировать в постоянное хранилище, так как при завершении работы с экземпляром данные будут потеряны (как и любые изменения в системе).

Это особенно важно при работе с СУБД, в том числе и MySQL. Нельзя хранить данные в хранилище экземпляра, так как если экземпляр будет по какой-то причине уничтожен, вы потеряете все свои данные.

Можно либо регулярно выполнять резервное копирование данных, либо подключить к экземпляру постоянное хранилище. Как говорилось ранее, идеальным решением будет EBS. Используя EBS, можно создать том для хранения данных, который сохранится после уничтожения экземпляра.

В руководстве пользователя Amazon EC2 можно найти дополнительную информацию о работе с хранилищем экземпляра, в т. ч. о конфигурации RAID и резервном копировании данных.

Работа с томами EBS при помощи AWS Management Console

Сначала мы создадим том, а затем запустим экземпляр и подключимся к нему. Чтобы создать новый том EBS, используя AWS Management Console, выполните следующее:

1. Откройте страницу AWS Management Console (*https://console.aws.amazon.com/ec2/home*).

   Если вы еще не зарегистрировались, будет запрошена регистрация.

2. Щелкните ссылку **Volumes (Тома)** в левой части страницы.
3. Щелкните кнопку **Create Volume (Создать том)** и укажите размер тома в гигабайтах и зону доступности. Значение Snapshot (Мгновенные снимки) оставьте без изменений.

4. Щелкните кнопку **Create (Создать)**. Том будет создан, и снова откроется страница AWS Management Console. В списке EBS Volumes (Тома EBS) будет показано состояние томов. Если том готов, он находится в состоянии available (доступен).

За использование хранилища EBS придется платить, а за некоторые формы ввода-вывода из хранилища — платить отдельно. Если не используете созданные тома, удаляйте их.

Запомните, в какой зоне создали том. Тома можно подключать только к экземплярам, запущенным в той же зоне. На рис. 14-14 показано, как выглядит в списке том EBS, готовый к работе.

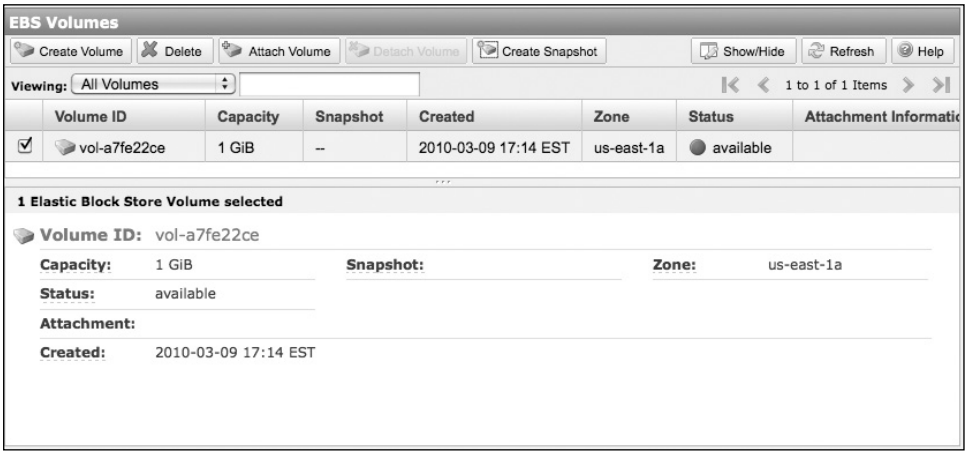


Рис. 14-14. Тома EBS

Обратите внимание на ID тома EBS. Можете также просмотреть дополнительные метаданные тома.

Теперь можно запускать экземпляр, а затем присоединять том к запущенному экземпляру. Для этого вернитесь к списку EBS Volumes, выберите том, который хотите присоединить к экземпляру, и щелкните кнопку Attach Volume (Присоединить том). Откроется диалоговое окно (рис. 14-15), в котором можно выбрать экземпляр и дисковое устройство (например, */dev/sdf*). Пользователи Windows увидят соответствующее имя диска.

Рис. 14-15. Выбор экземпляра для присоединения тома

Теперь можно войти в экземпляр и приступить к использованию тома. Сначала нужно подготовить диск. Ниже приведена типичная процедура для Linux. В Windows следует отформатировать диск, используя консоль *diskmgmt.msc*.

1. Создайте файловую систему XFS.

```
yes | mkfs -t xfs /dev/sdf
```

2. Создайте точку монтирования.

```
mkdir /mnt/mysql-data
```

3. Смонтируйте устройство.

```
mount /dev/sdf /mnt/mysql-data
```

Если в используемом образе не установлена XFS, может потребоваться установка утилит XFS (например, *xfsprogs*).

Чтобы отключить том, размонтируйте устройство, вернитесь в AWS Management Console, выберите том, щелкните кнопку Detach Volume (Отключить том) и подтвердите операцию.

Использование мгновенных снимков EBS при помощи AWS Management Console

Если для хранения данных используются тома EBS, можно в любое время быстро выполнять резервное копирование, используя мгновенные снимки EBS. При этом создается новая копия тома, содержащая все изменения, внесенные до момента выполнения копирования.

Чтобы создать снимок существующего тома, выполните следующее:

1. Выполните команду FLUSH TABLES WITH READ LOCK на сервере баз данных. Это гарантирует, что все данные записаны на диск, и обеспечит целостность данных.
2. Откройте страницу AWS Management Console (*https://console.aws.amazon.com/ec2/home*).

Если вы еще не зарегистрировались, будет запрошена регистрация.

3. Щелкните ссылку **Snapshots (Мгновенные снимки)** в левой части страницы.
4. Щелкните кнопку **Create Shapshot (Создать мгновенный снимок)**, выберите том, который хотите копировать, и введите имя мгновенного снимка. На рис. 14-16 показан диалог Create Shapshot (Создание мгновенного снимка).
5. Щелкните кнопку **Create (Создать)**. Мгновенный снимок будет создан, после чего откроется страница AWS Management Console. В списке EBS Snapshots будет показано состояние мгновенных снимков. Если том го-

тов, состояние изменится на available (доступен). На рис. 14-17 показан пример списка EBS Snapshots.

6. Выполните команду UNLOCK TABLES на сервере баз данных, чтобы разблокировать таблицы.

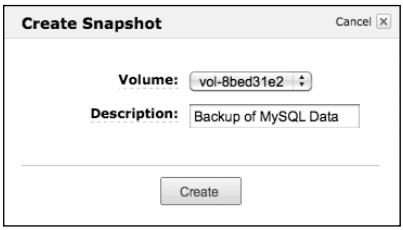


Рис. 14-16. Диалог Create Snapshot

Чтобы удалить мгновенный снимок, выберите его в списке EBS Snapshots, щелкните кнопку **Delete** и подтвердите операцию.

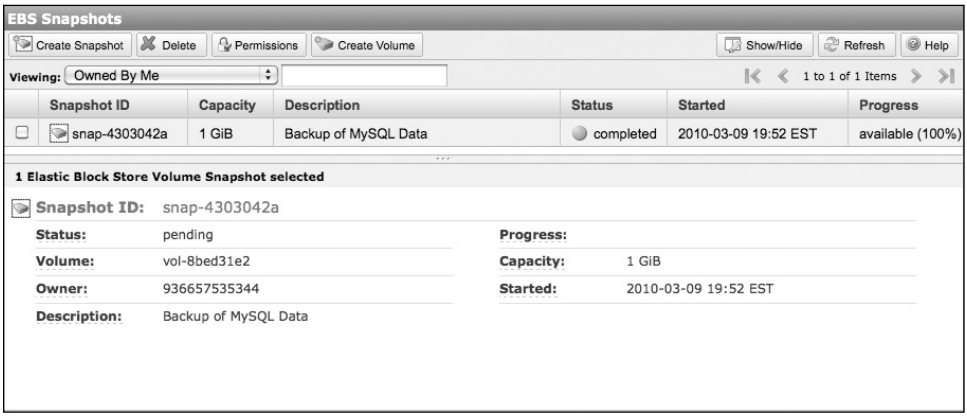


Рис. 14-17. Мгновенные снимки EBS

Работа с томами EBS с помощью утилит EC2 API

Тома можно создавать, подключать, отключать и удалять, используя утилиты EC2 API. Чтобы создать том, используйте команду ec2-create-volume, как показано ниже. Необходимо указать размер тома в Гб и зону.

```
Chucks-MacBook-Pro:~ Chuck$ ec2-create-volume -s 1 -z us-east-1a
VOLUME  vol-7fec3016  1     us-east-1a  creating
   2010-03-10T00:28:09+0000
```

Получить список всех томов можно командой ec2-describe-volumes. Будет возвращена та же информация, какая показана в секции EBS Volumes на странице AWS Management Console. Ниже приведен пример выполнения команды. Обратите внимание на ID тома, это значение понадобится для выполнения операций с томом.

```
Chucks-MacBook-Pro:~ Chuck$ ec2-describe-volumes
VOLUME  vol-7fec3016  1     us-east-1a  available
   2010-03-10T00:28:09+0000
```

Чтобы присоединить том к запущенному экземпляру, используйте команду ec2-attach-volume, указав ID тома, ID экземпляра и имя устройства:

```
Chucks-MacBook-Pro:~ Chuck$ ec2-attach-volume vol-7fec3016 -i i-414f962a
 -d /dev/sdf
ATTACHMENT  vol-7fec3016  i-414f962a  /dev/sdf  attaching
   2010-03-10T00:31:06+0000
```

Чтобы смонтировать и отформатировать том для использования, выполните процедуру, описанную выше в разделе «Использование мгновенных снимков EBS при помощи AWS Management Console». Процедура размонтирования тома также описана выше.

Чтобы отключить том от экземпляра, используйте команду ec2-detach-volume, аналогично предыдущему примеру. Необходимо будет указать ID тома.

```
Chucks-MacBook-Pro:~ Chuck$ ec2-detach-volume vol-7fec3016
ATTACHMENT  vol-7fec3016  i-414f962a  /dev/sdf  detaching
   2010-03-10T00:31:06+0000
```

Наконец, чтобы удалить том, используйте команду ec2-delete-volume, указав ID тома:

```
Chucks-MacBook-Pro:~ Chuck$ ec2-delete-volume vol-7fec3016
VOLUME  vol-7fec3016
```

Мы описали базовые операции с томами. Теперь рассмотрим, как можно создавать мгновенные снимки существующих томов.

Использование мгновенных снимков EBS при помощи утилит EC2 API

Чтобы создать мгновенный снимок при помощи утилит EC2 API, используйте команду ec2-create-snapshot, указав ID тома. Приведем пример создания мгновенного снимка тома, созданного ранее:

```
Chucks-MacBook-Pro:~ Chuck$ ec2-create-snapshot vol-a7fe22ce
SNAPSHOT  snap-ad5a5dc4  vol-a7fe22ce  pending
   2010-03-09T22:45:22+000936657535344  1
```

Создание мгновенного снимка может занять какое-то время. Для получения списка созданных мгновенных снимков используйте команду ec2-describe-snapshots:

```
Chucks-MacBook-Pro:~ Chuck$ ec2-describe-snapshots
SNAPSHOT  snap-ad5a5dc4  vol-a7fe22ce  completed
   2010-03-09T22:45:23+0000  100%  936657535344  1
```

Для удаления мгновенного снимка используйте команду ec2-delete-snapshot. Мгновенный снимок будет удален без возможности восстановления, так что будьте внимательны, выполняя эту команду.

```
Chucks-MacBook-Pro:~ Chuck$ ec2-delete-snapshot snap-ad5a5dc4
SNAPSHOT  snap-ad5a5dc4
```

Проверьте состояние мгновенных снимков, чтобы убедиться, что они удалены. Команда ec2-describe-shapshots должна возвратить пустой список.

Что дальше?

Продукты из облака Amazon очень сложны, и для их изучения может потребоваться много времени. К счастью, на веб-сайте AWS можно найти много материала для обучения. Ниже приведены некоторые из наиболее полезных ссылок:

http://aws.amazon.com/documentation/ec2.
http://aws.amazon.com/ec2.
http://aws.amazon.com/autoscaling.
http://aws.amazon.com/s3.
http://aws.amazon.com/ebs.

MySQL в облаке

Если вы беспокоитесь о том, что для запуска MySQL в облаке придется делать что-то особенное, можете расслабиться. Работа с MySQL в облаке ничем не отличается от работы на «настоящем» оборудовании. Точно так же выполняется репликация, создаются масштабируемые решения с высокой доступностью и используются те же самые инструменты для мониторинга и управления.

Единственное, что отличает использование MySQL в облаке от развертывания на обычном «железе» — возможность быстро развертывать сотни серверов в любое время, когда они понадобятся. В этом разделе мы рассмотрим некоторые решения, предложенные в предыдущих главах, и проверим, как повлияет на них реализация в облаке. Начнем с применения репликации MySQL в облаке Amazon.

Репликация MySQL и EC2

В этом разделе мы покажем простоту репликации MySQL в EC2. В приведенном примере запускается экземпляр образа LAMP Web Starter, подключение к которому производится при помощи утилит командной строки EC2. Затем мы изменяем экземпляр, чтобы он работал как главный сервер, запускаем локальный экземпляр MySQL и используем его в качестве подчиненного сервера. Этот процесс идентичен настройке репликации в локальной среде. Единственное отличие состоит в том, что главный сервер запущен в облаке.

Первый шаг — запустить экземпляр и подключиться к нему. Ниже приведен код, показывающий этапы и результаты подключения к запущенному экземпляру EC2:

```
Chucks-MacBook-Pro:~ Chuck$ ssh -i ./keys/orig.pem
root@ec2-184-73-10-112.compute-1.amazonaws.com
The authenticity of host 'ec2-184-73-10-112.compute-1.amazonaws.com
(184.73.10.112)' can't be established.
RSA key fingerprint is cd:79:eb:e5:e9:2e:d6:a2:9c:79:65:2a:27:c5:1b:ba.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-184-73-10-112.compute-1.amazonaws.com,
184.73.10.112' (RSA) to the list of known hosts.
Permission denied (publickey,gssapi-with-mic).

   __|   |  ) Fedora 8
   _|    (   /  32-bit
   __|\   |   |

Welcome to an EC2 Public Image
                    :-)

Base

--[ see /etc/ec2/release-notes ]--

[root@ip-10-212-106-195 ~]# mysql -uroot
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.0.45 Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Следующий шаг — завершить работу удаленного экземпляра и изменить файл *my.cnf* (*/etc/my.cnf*). Необходимо указать server-id = 1 и log-bin = mysql-bin. Перезапустите сервер MySQL и убедитесь, что он готов к подключению:

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |       98 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Необходимо выполнить команду GRANT, чтобы разрешить подключение подчиненному серверу:

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'rpl'@'%' IDENTIFIED BY 'rpl';
```

Следующий шаг — настроить подчиненный сервер, чтобы он подключался к главному серверу в EC2. Используйте команду CHANGE MASTER, указав адрес экземпляра EC2, сведения о двоичном журнале главного сервера и учетную запись пользователя репликации, как показано ниже:

```
mysql> CHANGE MASTER TO
   MASTER_HOST='ec2-184-73-10-112.compute-1.amazonaws.com',
   MASTER_USER='rpl',
   MASTER_PASSWORD='rpl',
   MASTER_PORT=3306,
   MASTER_LOG_FILE='mysql-bin.000001',
   MASTER_LOG_POS=98;
Query OK, 1 row affected (0.00 sec)
mysql> START SLAVE;
Query OK, 1 row affected (0.00 sec)
```

Чтобы выяснить, работает ли репликация, можно проверить состояние подчиненного сервера, используя команду SHOW SLAVE STATUS. Результаты должны показывать, что подчиненный сервер активен:

```
mysql> SHOW SLAVE STATUS \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: ec2-204-236-207-171.compute-1.amazonaws.com
                  Master_User: rpl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 325
               Relay_Log_File: mysqld-relay-bin.000002
                Relay_Log_Pos: 470
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 325
              Relay_Log_Space: 626
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
```

```
          Master_SSL_Allowed: No
          Master_SSL_CA_File:
          Master_SSL_CA_Path:
             Master_SSL_Cert:
           Master_SSL_Cipher:
              Master_SSL_Key:
       Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
               Last_IO_Errno: 0
               Last_IO_Error:
              Last_SQL_Errno: 0
              Last_SQL_Error:
1 row in set (0.00 sec)
```

Теперь создадим базу данных на главном сервере в экземпляре EC2 и проверим ее на локальном подчиненном сервере:

```
mysql> CREATE DATABASE amazon_ec2_test;
Query OK, 1 row affected (0.00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| amazon_ec2_test    |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.02 sec)
```

На подчиненном сервере мы увидим, что база данных была создана:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| amazon_ec2_test    |
| employees          |
| mysql              |
| sakila             |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

Попробуйте выполнить это, и увидите, как легко использовать MySQL в EC2. После выполнения указанных шагов у вас будет локальный подчиненный сервер, реплицирующий данные с главного сервера, находящегося в

EC2. Обратный эксперимент можете выполнить сами. Убедитесь только, что в настройках брандмауэра и маршрутизатора разрешен входящий трафик на порт TCP 3306.

Помните, что экземпляры в облаке Amazon EC2 общедоступны. Внимательно устанавливайте права доступа и пароли. В частности, не забывайте устанавливать пароль на корневой доступ к MySQL и запрещайте удаленный вход, чтобы к серверу могли подключаться только авторизованные пользователи. Кроме того, открывайте только те порты, которые необходимы для взаимодействия с приложениями в облаке.

Рекомендации по использованию MySQL в EC2

Как мы уже говорили, в облаке с MySQL можно делать почти все, что и на собственном физическом оборудовании, за исключением, конечно, создания аппаратных кластеров и аналогичных аппаратных решений для обеспечения высокой доступности. Но мы должны помнить, что оборудование может быть виртуализировано, так что если сегодня существует какое-то аппаратное решение, то есть вероятность, что завтра оно станет виртуальным ресурсом.

Использование MySQL в облаке Amazon лишь немного отличается от использования MySQL на собственном оборудовании. Если вы используете AMI с установленной и настроенной системой MySQL, то сможете быстро подключить данные, хранящиеся на томах EBS, и запустить сервер. Или создавать собственные AMI с определенными конфигурациями MySQL, чтобы автоматизировать некоторые задачи, указав в параметре *datadir* путь к томам EBS.

Каковы же тогда рекомендации по использованию MySQL в облаке? Ниже приведены советы, позволяющие получить наилучшую производительность при работе с MySQL в облаке.

- *В одном экземпляре EC2 запускайте только один сервер* MySQL будет работать быстрее, если в его распоряжении находится отдельная CU и память. Учитывая простоту и относительно низкую стоимость создания новых экземпляров, можно получить гораздо большую производительность, запуская один экземпляр MySQL в экземпляре слабого типа, чем при запуске нескольких экземпляров MySQL в экземпляре мощного типа. Помните, что вы в любое время можете добавлять столько экземпляров, сколько потребуется, расширяя и сокращая инфраструктуру при необходимости.
- *Используйте экземпляры более мощного типа для решения задач с высокой нагрузкой* Для загруженных БД используйте экземпляры крупного типа. Такие экземпляры не только имеют больше вычислительных единиц и памяти, но и обеспечивают более интенсивный ввод-вывод. В таких случаях лучше больше заплатить, но получить высокую производительность.
- *Используйте дополнительные тома EBS для таблиц InnoDB* Базы данных обычно очень требовательны к производительности ввода-вывода, которая может стать узким местом системы. Чтобы повысить скорость,

используйте отдельные тома EBS для хранения БД и таблиц InnoDB (например, по одной на каждый том EBS).

- *«Прогревайте» разделы данных* Использование дискового ввода-вывода в EC2 имеет один недостаток: низкую скорость «первой операции записи» при первоначальной записи данных в новые разделы. Чтобы избежать этого, можно «прогреть» раздел, выполнив какую-то одиночную команду, обращающуюся к нему. Например, можно использовать команду Linux `dd`, выполняющую запись на диск. Хотя снижение производительности все равно будет, и его нельзя избежать, на первой «настоящей» операции записи в БД оно не скажется.
- *Правильно настраивайте MySQL* Если просто запустить MySQL в EC2, работа не пойдет быстрее. Нельзя пропускать этап настройки системы MySQL в соответствии с нуждами конкретного приложения. Правильная настройка позволит лучше использовать ресурсы EC2.
- *Не забывайте о мониторинге* Можно и нужно выполнять мониторинг серверов MySQL, запущенных в облаке, используя методы, описанные в этой книге. Следует также использовать средства мониторинга, предлагаемые Amazon. Если серверы виртуальные, это не значит, что они не страдают от бесконечных циклов и прочих проблем с производительностью.
- *Используйте репликацию MySQL* В предыдущих главах вы видели, насколько важна репликация MySQL для масштабирования, балансировки нагрузки и высокой доступности. Облако облегчает использование этих возможностей, так как в нем можно создать столько экземпляров MySQL, сколько требуется (почти) для любого решения, обеспечивающего высокую доступность.
- *Используйте стандартные AMI* Возможность создавать собственные AMI с нуля позволяет настраивать серверы в соответствии с нуждами различных платформ. К сожалению, если ваш опыт работы с операционными системами в виртуальном окружении невелик, и вы хотите большего, чем незначительное изменение стабильного AMI, эта задача может занять много времени и превратиться в долгий процесс отлова ошибок. При любой возможности используйте существующие AMI от Amazon или стабильные AMI от сообщества.
- *Следуйте рекомендациям по обеспечению безопасности* Облако не отличается от других устройств, подключенных к Интернету. Всегда предпринимайте хорошо известные меры защиты MySQL (например, не используйте пустые пароли). Следует также ограничить доступ к экземпляру EC2, чтобы только авторизованные пользователи (и системы, такие как подчиненные серверы репликации) имели доступ к вашим виртуальным системам. EC2 включает брандмауэр, автоматически ограничивающий входящий трафик. У некоторых AMI открыты определенные порты, например 3306 для клиентов MySQL, но если требуются дополнительные порты, их нужно явно открыть. Для управления общими правилами для

групп экземпляров EC2 используйте собственные группы безопасности.

- *Монтируйте разделы с параметрами noatime и nodiratime* Монтирование разделов с любым из этих параметров добавит до 10% производительности ввода-вывода. Это произойдет потому, что Linux не будет выполнять операцию записи после каждой успешной операции чтения. noatime является расширенной версией nodiratime.
- *Используйте EBS с MySQL* Как говорилось ранее, EBS — это блочное устройство хранения с хорошей производительностью, устойчивостью к сбоям экземпляров и эластичностью. Вы получаете не только постоянное хранилище для данных и фалов журналов, но и возможность перейти на другой сервер в случае уничтожения экземпляра.
- *Выполняйте резервное копирование с использованием S3* Можно использовать возможности EC2 для создания мгновенных снимков томов и хранить их в S3 в качестве резервных копий. Мгновенные снимки — эффективный способ создания резервных копий и быстрого восстановления в случае повреждения данных.
- *Используйте балансировку нагрузки* Производительность приложений с высокой нагрузкой или большим числом одновременных подключений можно повысить при помощи балансировки нагрузки. Как говорилось в предыдущих главах, можно использовать репликацию MySQL на нескольких подчиненных серверах, чтобы улучшить производительность чтения, а производительность записи повышать при помощи сегментирования или шардинга (sharding). Можно использовать ресурс Amazon Elastic Load Balancing или собственные балансировщики нагрузки, такие как HAProxy (*http://haproxy.1wt.eu*).

Открытое ПО для облачных вычислений

Если вы не хотите тратить деньги на эксперименты и освоение облачных вычислений, подумайте об открытом ПО. В состав сервера Ubuntu входит Eucalyptus — последняя открытая система для облачных вычислений, во многом схожая с Amazon EC2 и позволяющая изучать облачные вычисления без финансовых затрат. Например, если вы хотите научиться настраивать решения с применением облака для домашних экспериментов или разработки, то можете загрузить сервер облака Ubuntu и приступить к работе. Ниже приведена последовательность команд для запуска экземпляра в облаке Ubuntu при помощи утилит командной строки. Обратите внимание на то, что эти команды похожи на команды Amazon, описанные в этой главе.

```
cbell@ubuntu-cloud:~$ euca-describe-images
IMAGE  eri-099C1159  image-store-1266350672/ramdisk.manifest.xml  admin
   available  public   x86_64  ramdisk
IMAGE  emi-DED7106D  image-store-1266350672/image.manifest.xml  admin
```

```
   available  public   x86_64  machine
IMAGE  eki-F52D10EB  image-store-1266350672/kernel.manifest.xml  admin
   available  public   x86_64  kernel
cbell@ubuntu-cloud:~$ euca-describe-volumes
VOLUME  vol-330A04B9   10     cloud9  in-use  2010-02-17T18:54:43.589Z
ATTACHMENT  vol-330A04B9  i-41EE0860  unknown,requested:/dev/sdb
   2010-02-18T17:44:11.561Z
VOLUME  vol-32DC04A3   10     cloud9  in-use  2010-02-17T18:54:41.002Z
ATTACHMENT  vol-32DC04A3  i-4DDB09AC  unknown,requested:/dev/sdb
   2010-02-18T17:44:11.561Z

cbell@ubuntu-cloud:~$ euca-describe-instances
RESERVATION r-5C060A63 admin default
INSTANCE  i-41EE0860  emi-DED7106D  172.19.1.3  172.19.1.3
   running   mykey   0   c1.medium   2010-02-18T17:32:12.441Z
   cloud9 eki-F52D10EB   eri-099C1159
RESERVATION  r-4027075E  admin  default
INSTANCE  i-4DDB09AC  emi-DED7106D  172.19.1.2  172.19.1.2
   running   mykey   0   c1.medium   2010-02-18T17:31:55.128Z
   cloud9 eki-F52D10EB   eri-099C1159
```

Подробнее об облаке Ubuntu и установке собственных решений на его основе см. *http://www.ubuntu.com/cloud* и *https://help.ubuntu.com/community/UEC*. Для установки среды облачных вычислений Ubuntu потребуется минимум два сервера, имеющих поддержку виртуализации и многоядерные процессоры.

Заключение

Немного разобравшись в облачных вычислениях и увидев возможности облака Amazon, легко представить, какие горизонты открывают старые технологии, объединенные новым способом. С появлением облачных вычислений потребность в традиционной ИТ-инфраструктуре уходит в прошлое. Облачные вычисления позволяют наращивать инфраструктуру с ростом предприятия, не нанимая целый отдел специалистов для ее обслуживания.

В этой главе мы дали определение облачным вычислениям, изучили некоторые базовые технологии и вкратце рассмотрели продукты Amazon EC2, S3 и EBS. Также мы привели пример того, как можно организовать репликацию в среде Amazon EC2.

В завершение мы обсудили возможности использования MySQL в облаке и выяснили, что в нем можно делать все, что и на собственном оборудовании, но гораздо быстрее и с меньшими затратами ресурсов и денег. Можно даже автоматически масштабировать свое решение в соответствии с текущей нагрузкой. Кому это не понравится?

— Джоэл!

Из динамика компьютера раздался сердитый гудок, когда Джоэл непроизвольно нажал несколько клавиш сразу. Он поднял глаза на улыбающегося г-на Саммерсона, стоящего в двери. Прежде чем Джоэл смог что-нибудь сказать, его босс произнес:

— Твое предложение насчет облачных вычислений — хорошая работа. Это как раз то, что нам нужно. Вечером я иду на собрание руководства. — Г-н Саммерсон отправил по воздуху копию предложения Джоэла, которая шлепнулась ему на стол. — Я сделал тут несколько замечаний. Подготовь десяток слайдов и приходи к 12:30 в зал заседаний.

Джоэл в смятении посмотрел на пометки красными чернилами.

— Э... сэр?..

— Не волнуйся ты так, Джоэл. Почти все директора когда-то были такими же сотрудниками, как мы. Надеюсь, ты любишь пиццу.

Джоэл на секунду задумался над этим «мы». Он собрался было что-то спросить, но г-н Саммерсон уже испарился. А Джоэл смотрел на свои джинсы и футболку в раздумье: успеет ли он переодеться дома и подготовить презентацию?..

ГЛАВА 15

MySQL Cluster

Услышав тихий стук в дверь, Джоэл поднял глаза и увидел г-на Саммерсона, выглядевшего обеспокоенным.

— В этот раз тебе достанется сложное задание, Джоэл, как и всем нам, впрочем.

Джоэл притих, пытаясь представить, что это может означать: прежние задания тоже были не из легких.

— У нас появился клиент, который хочет использовать наше последнее приложение в среде реального времени с доступностью 99,999%.

— То есть, никаких простоев?

— Именно. Я знаю, что СУБД MySQL очень надежна, но у нас нет времени, чтобы переписывать наше приложение для добавления поддержки отказоустойчивого сервера баз данных.

Джоэл вспомнил, что как раз дочитал книгу до главы про особую версию MySQL. Видимо, это именно то, что надо. Была не была!

— Мы могли бы использовать кластерные технологии...

— Кластерные?

— Да, есть кластерная версия MySQL. Это отказоустойчивая СУБД, ее используют в средах с довольно высокими требованиями, например, в телекоммуникациях, насколько я помню...

Г-н Саммерсон просиял и распрямился, как будто только что одержал убедительную победу.

— Отлично. Подготовь резюме к завтрашнему утру. Мне нужны цены, требования к оборудованию, ограничения, — словом, все что обычно. И не скромничай с бюджетом: мы возьмемся за эту задачу, только если реально сможем справиться с ней, при этом я ни в коем случае не хочу рисковать нашей репутацией.

— Начну прямо сейчас, — ответил Джоэл, прикидывая, во что он ввязался.

Проводив Саммерсона взглядом, он вздохнул и открыл свою любимую книгу по MySQL.

— Наверно, это самое сложное из моих заданий...

Когда высокая производительность, высокая доступность, избыточность и масштабируемость являются основными приоритетами проектировщиков БД, они часто пытаются усовершенствовать топологии репликации с помощью новейшего оборудования и решений для балансировки нагрузки. Такой

подход часто позволяет удовлетворить требования большинства организаций, но если необходимо решение без единого слабого звена, с очень высокой производительностью и надежностью 99,999%, следует обратить внимание на технологию MySQL Cluster.

В этой главе мы познакомимся с концепциями этой технологии, покажем пример запуска и остановки простого кластера и обсудим ключевые положения использования кластеров MySQL, включая высокую доступность, распределение данных и репликацию. Сначала расскажем, что такое MySQL Cluster и в чем отличия от обычных серверов MySQL.

Что такое MySQL Cluster?

MySQL Cluster — решение для хранения данных, не предусматривающее разделение ресурсов, имеющее распределенную архитектуру узлов, обеспечивающее высокую производительность и отказоустойчивость. Данные хранятся и реплицируются на отдельных узлах (иногда их называют *узлами-хранилищами*), каждый из которых работает на отдельном сервере и содержит копию данных. Каждый кластер содержит также *управляющие узлы*. При модификации данных используется изоляция с фиксацией при чтении, гарантирующая согласованность данных на всех узлах, и двухэтапная фиксация, гарантирующую идентичность данных на всех узлах (если любая операция записи не удается, вся модификация завершается ошибкой).

В исходной реализации MySQL Cluster вся информация хранилась в основной, а не постоянной памяти. В последующих выпусках появилась возможность сохранения данных на диске. Наверно, лучшая черта MySQL Cluster — использование в качестве механизма исполнения запросов сервера MySQL. Это позволяет легко переносить на MySQL Cluster приложения, написанные для MySQL.

Благодаря концепции равноправных узлов с отсутствием разделения ресурсов, обновления на одном сервере тут же становятся видимыми остальным серверам. Для передачи обновлений используется сложный механизм, предназначенный для сетей с очень высокой интенсивностью обмена данными. Цель — получить максимальную производительность путем распределения нагрузки по нескольким серверам MySQL, а также высокую доступность и избыточность (посредством хранения данных в разных местах).

Терминология и компоненты

Типичные установки MySQL Cluster включают установку компонентов кластера на разных компьютерах в сети. Поэтому MySQL Cluster называют также сетевой СУБД (network database, NDB). Когда мы используем термин «MySQL Cluster», мы имеем в виду сервер MySQL и компоненты NDB. Но, используя термины «NDB» и «кластер NDB», мы говорим только о кластерных компонентах.

MySQL Cluster — СУБД, использующая сервер MySQL в качестве внешнего интерфейса для поддержки стандартных запросов SQL. Для сопряжения сервера MySQL с кластерными технологиями используется механизм БД NDBcluster. Тут часто возникает путаница. Нельзя использовать NDBcluster без кластерных компонентов NDB. Однако можно использовать NDB без сервера MySQL, но только посредством низкоуровневого программирования с использованием NDB API.

NDB API — объектно-ориентированный интерфейс, поддерживающий индексы, операции сканирования, транзакции и обработку событий. Он позволяет писать приложения для получения, хранения и изменения данных в кластере. Этот интерфейс также предоставляет объектно-ориентированные средства обработки ошибок, позволяющие корректно завершать работу или выполнять восстановление после сбоев. Подробную информацию для разработчиков о NDB API можно найти по адресу *http://dev.mysql.com/doc/NDBapi/en/index.html*.

Отличия MySQL Cluster от MySQL

Возможно, вам будет интересно узнать, чем кластер отличается от репликации. Существует несколько определений кластеров, но, в общем, это система с возможностью управления членством в кластере, обмена сообщениями, избыточности и автоматической обработки отказов. Репликация же — это просто способ отправки сообщений (данных) от одного сервера другому. Далее в этой главе мы обсудим репликацию в кластере (также называемую *локальной репликацией*) и репликацию MySQL.

Типичная конфигурация

В MySQL Cluster можно выделить три уровня:

- приложения, взаимодействующие с сервером MySQL;
- сервер MySQL, обрабатывающий команды SQL и взаимодействующий с механизмом NDB;
- компоненты кластера NDB (иногда называемые *узлами данных*), обрабатывающие запросы и возвращающие результаты серверу MySQL.

Для повышения производительности каждый уровень можно независимо масштабировать, добавляя серверные процессы.

На рис. 15-1 показана концептуальная схема типичной кластерной установки.

Приложения подключаются к серверу MySQL, который обращается к компонентам кластера NDB через уровень механизма хранилища (а точнее механизма хранилища NDB). Ниже мы рассмотрим компоненты кластера NDB подробнее.

Существует множество возможных конфигураций. Можно использовать несколько серверов MySQL для подключения к одному кластеру NDB

и даже соединять несколько кластеров NDB посредством репликации MySQL. Подробнее об этом мы поговорим в последующих разделах.

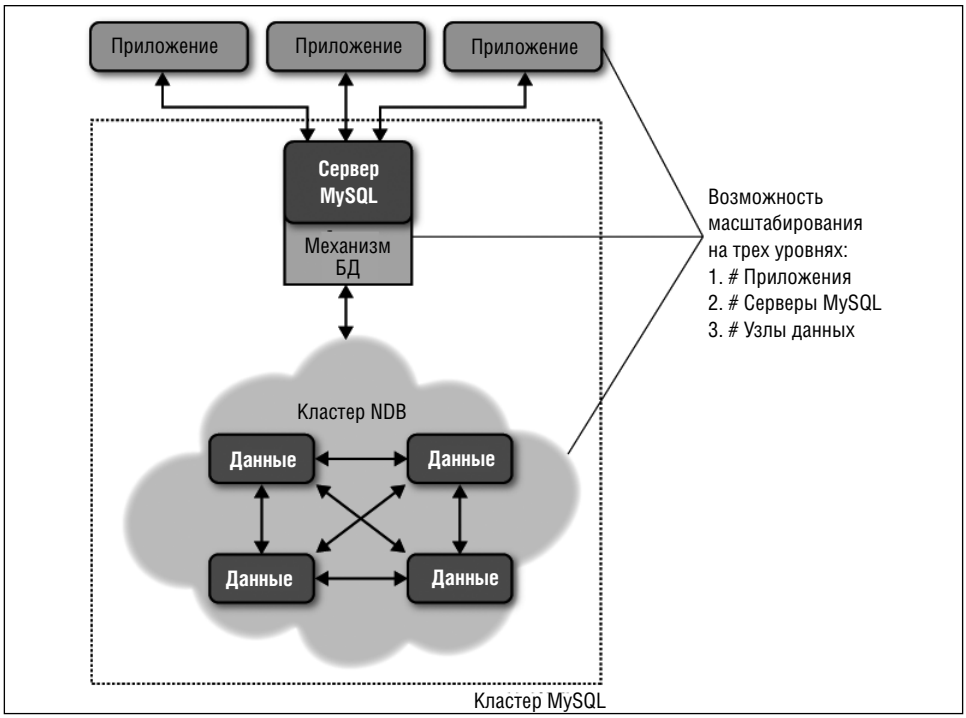


Рис. 15-1. Кластер MySQL

Возможности MySQL Cluster

Для обеспечения наивысшей производительности, доступности и избыточности данные реплицируются внутри кластера между равноправными узлами данных. Репликация выполняется с использованием синхронного механизма, в котором узлы данных подключаются ко всем другим узлам данных, и данные хранятся на нескольких узлах.

Также можно реплицировать данные между кластерами, но в этом случае используется репликация MySQL, являющаяся *асинхронной*, а не синхронной. Как говорилось в предыдущих главах, при асинхронной репликации следует ожидать задержку в обновлении подчиненных серверов, которые не сообщают о том, как проходит внесение изменений, и не следует ожидать последовательного представления на всех серверах в архитектуре репликации, как внутри одного кластера MySQL.

MySQL Cluster поддерживает несколько специализированных функций для создания систем с высокой доступностью. Перечислим наиболее важные из них:

- *Восстановление узлов.* Сбои узлов данных могут быть обнаружены по потере связи или heartbeat-импульсов, и можно настроить узлы на ав-

томатический перезапуск с использованием копий данных с оставшихся узлов. Сбой и восстановление может охватывать один или несколько узлов хранилища. Такое восстановление называют *локальным*.

- *Журналирование*. В процессе обычных обновлений копии событий, изменяющих данные, записываются в журнал, хранящийся на каждом узле данных. Эти журналы можно использовать для восстановления данных до состояния в определенный момент времени.
- *Контрольные точки*. Кластер поддерживает два вида контрольных точек: локальные и глобальные. Локальные контрольные точки удаляют «хвост» журнала. Глобальные контрольные точки создаются, когда журналы всех узлов данных переносятся на диск, создавая при этом мгновенный снимок всех данных, обладающий всеми характеристиками транзакции. Таким образом, контрольные точки позволяют выполнять полное восстановление всех узлов системы из стабильной точки синхронизации.
- *Восстановление системы*. В случае неожиданного выхода из строя всей системы можно восстановить ее при помощи контрольных точек и журналов изменений. Обычно данные копируются с диска в память из стабильных точек синхронизации.
- *Горячее резервное копирование и восстановление*. Можно одновременно создавать резервные копии всех узлов данных, не прерывая обработку транзакций. Резервная копия будет включать метаданные об объектах базы данных, сами данные и текущий журнал транзакций.
- *Отсутствие единой точки сбоя*. Архитектура кластера спроектирована так, что выход из строя любого узла не приведет к выходу из строя СУБД в целом.
- *Обработка отказов*. Чтобы гарантировать возможность восстановления узлов, все транзакции фиксируются с использованием изоляции при фиксации чтения и двухэтапных фиксаций. Это обеспечивает двойную защиту транзакций: они сохраняются в двух разных местах, прежде чем пользователь получит подтверждение транзакции.
- *Использование разделов*. Данные автоматически разделяются между узлами данных. MySQL Cluster 5.1 поддерживает разделы, определенные пользователем.
- *Онлайновые операции*. Многие операции по обслуживанию можно выполнять без остановки кластера. Такие операции обычно требуют остановки сервера или блокировки данных. Например, можно добавлять новые узлы данных, изменять структуру таблиц и даже реорганизовывать данные в кластере.

Подробнее о MySQL Cluster см. в онлайновой документации (*http://dev.mysql.com/doc/mysql-cluster-excerpt/5.1/en/index.html*).

Локальная и глобальная избыточность

Локальную избыточность (внутри отдельного кластера) можно создавать при помощи двухэтапного протокола фиксации. В принципе, каждый узел проходит цикл согласования изменений, а затем цикл фиксации транзакции. На этапе согласования каждый узел гарантирует, что имеет достаточно ресурсов для фиксации транзакции во втором цикле. В кластере NDB протокол фиксации на сервере MySQL изменяется, чтобы позволить внесение изменений на нескольких узлах. Кластер NDB также имеет оптимизированную версию двухэтапной фиксации, в которой снижено число сообщений, отправляемых с использованием синхронной репликации. Двухэтапный протокол гарантирует, что данные избыточно хранятся на нескольких узлах данных. Такое состояние называется *локальной избыточностью*.

Для обеспечения глобальной избыточности используется репликация MySQL между кластерами. При этом в топологии репликации устанавливается два узла. Как говорилось ранее, репликация MySQL является асинхронной, так как не включает подтверждение или уведомление о приеме или выполнении реплицируемых событий. Различия показаны на рис. 15-2.

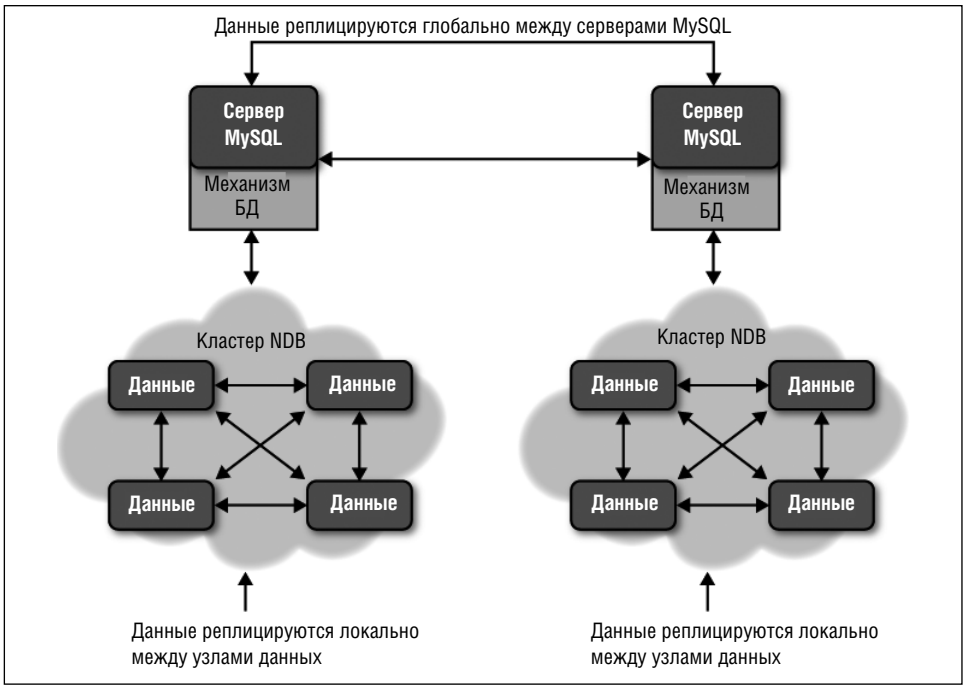


Рис. 15-2. Локальная и глобальная избыточность

Обработка журналов

В MySQL Cluster реализовано два вида контрольных точек: локальные контрольные точки для очистки части журнала повторов и глобальные кон-

трольные точки, предназначенные в основном для синхронизации разных узлов данных. Глобальные контрольные точки важны для репликации, так как формируют границы между разными наборами транзакций, называемые *эпохами*. Каждая эпоха реплицируется между кластерами как одна единица. В действительности, репликация MySQL обрабатывает наборы транзакций между двумя последовательными глобальными контрольными точками как одну транзакцию.

Избыточность и распределенные данные

Для обеспечения избыточности данных используются *реплики*. Каждая реплика содержит копию данных. Это обеспечивает отказоустойчивость кластера. Если любой узел выйдет из строя, данные все равно останутся доступными. Понятно, что чем больше реплик в кластере, тем более отказоустойчивым является этот кластер.

Синдром «двух начальников»

Если один или несколько узлов данных выходят из строя, возможно, что оставшиеся узлы данных не смогут взаимодействовать. Когда такое происходит, получается два набора узлов данных с «синдромом двух начальников». Такая ситуация нежелательна, так как каждый набор узлов теоретически может функционировать как отдельный кластер.

Для исправления такой ситуации необходим алгоритм разделения сети, позволяющий сделать выбор между конкурирующими наборами узлов данных. Решение применяется в каждом наборе независимо. Набор с наименьшим числом узлов будет перезапущен, и каждый узел из этого набора должен будет подключиться к более крупному набору.

Если два набора имеют равное число узлов, проблема остается. Если разделить четыре узла на два набора по два узла в каждом, какой из наборов должен быть расформирован? Для ответа на этот вопрос можно назначить арбитра. Если два набора имеют одинаковый размер, остается набор, которому удастся связаться с арбитром первым.

Арбитром можно назначить сервер MySQL (узел SQL) или управляющий узел. Для обеспечения наилучшей доступности следует назначать арбитром систему, не содержащую узлов данных.

В MySQL Cluster алгоритм разделения сети с использованием арбитра полностью автоматизирован, и меньшинство определяется с учетом групп узлов, что делает систему еще более доступной, чем было бы при простом переборе узлов.

Можно указать, сколько копий данных (NoOfReplicas) должно существовать в кластере. Необходимо установить столько узлов данных, сколько реплик нужно иметь. Можно также распределить данные между узлами данных, используя разделы. В этом случае каждый узел данных будет содержать только часть данных, что позволит быстрее выполнять запросы. Но,

поскольку имеется несколько копий данных, в случае выхода одного узла из строя данные останутся доступными, и можно будет выполнить восстановление отсутствующего узла (так как данные существуют в других репликах). Чтобы иметь такую возможность, необходимо иметь несколько узлов данных для каждой реплики. Например, чтобы иметь две реплики и использовать разделение, необходимо минимум четыре узла данных (два узла для каждой реплики).

Архитектура MySQL Cluster

Кластер MySQL состоит из одного или нескольких серверов MySQL, взаимодействующих через механизм NDB с кластером NDB. Сам кластер NDB состоит из нескольких компонентов: узлы данных или хранилища, хранящие и извлекающие данные, и один или несколько узлов управления, координирующих запуск, остановку и восстановление узлов данных. Большинство компонентов NBD реализовано как процессы-демоны, и MySQL Cluster предоставляет клиентские утилиты для управления функциями демонов. Ниже приведен список демонов и утилит. На рис. 15-3 показана схема взаимодействия этих компонентов.

- *mysqld* — сервер MySQL;
- *NDBd* — узел данных;
- *NDBmtd* — многопоточный узел данных;
- *NDB_mgmd* — сервер управления кластера;
- *NDB_mgm* — клиент управления кластера.

Каждый сервер MySQL с исполняемым именем *mysqld* обычно поддерживает одно или несколько приложений, выполняющих запросы SQL и получающих результаты от узлов данных. В контексте MySQL Cluster серверы MySQL иногда называют *узлами SQL*.

Узлы данных являются процессами-демонами NDB, хранящими и извлекающими данные либо из памяти, либо с диска, в зависимости от конфигурации. Узлы данных устанавливаются на каждом сервере, входящем в кластер. Существует также многопоточный демон узлов данных, называющийся NDBmtd, работающий на платформах, поддерживающих многоядерные процессоры. Используя многопоточные узлы данных на выделенных серверах с многоядерными процессорами, можно заметно повысить производительность.

Демон управления NDB_mgmd работает на сервере и отвечает за чтение файла конфигурации и передачи полученной информации всем узлам кластера. NDB_mgm — клиентская утилита управления NDB — позволяет проверять состояние кластера, делать резервные копии и выполнять другие административные функции. Этот клиент работает на системе, удобной для администратора, и взаимодействует с демоном.

Кроме того, существуют утилиты, облегчающие обслуживание. Ниже перечислены наиболее популярные из них. Полный список можно найти в документации по NDB Cluster.

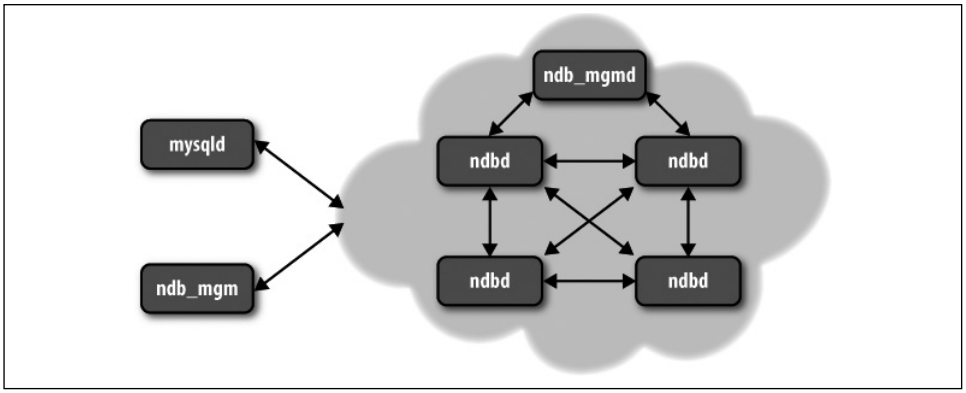


Рис. 15-3. Компоненты MySQL Cluster

- *NDB_config* — извлекает сведения о конфигурации из существующих узлов;
- *NDB_delete_all* — удаляет все строки из таблицы NDB;
- *NDB_desc* — выводит описание таблиц NDB (как SHOW CREATE TABLE);
- *NDB_drop_index* — удаляет индекс из таблицы NDB;
- *NDB_drop_table* — удаляет таблицу NDB;
- *NDB_error_reporter* — выполняет диагностику ошибок и проблем в кластере;
- *NDB_redo_log_reader* — проверяет и выводит журнал повторов кластера;
- *NDB_restore* — выполняет восстановление кластера из резервных копий, созданных с помощью клиента управления NDB.

Как хранятся данные

MySQL Cluster хранит все индексированные столбцы в основной памяти. Оставшиеся неиндексированные столбцы можно хранить в памяти или на диске, используя расположенный в памяти кэш страниц. Хранение столбцов на диске позволяет сохранять больше данных, чем позволяет размер доступной памяти.

Когда данные изменяются (командами INSERT, UPDATE, DELETE и т.д.), MySQL Cluster записывает запись изменения в журнал повторов, регулярно записывая данные в контрольных точках на диске. Как говорилось ранее, журнал и контрольные точки позволяют выполнять восстановление после сбоя. Однако, так как журналы повторов записываются асинхронно с фиксацией, при сбое может быть потеряно ограниченное число транзакций. Чтобы снизить такую вероятность, в MySQL Cluster реализована задержка

записи (по умолчанию она равна двум секундам, но ее можно настраивать). Это позволяет полностью записать контрольную точку, так что в случае сбоя последняя контрольная точка не будет потеряна. Обычно сбои отдельных узлов данных не приводят к потерям каких-либо данных благодаря синхронной репликации данных в пределах кластера.

Если таблица MySQL Cluster хранится в памяти, кластер обращается к дисковому хранилищу только для сохранения записей об изменениях в журнале повторов и для выполнения соответствующих контрольных точек. Так как запись журналов и контрольных точек выполняется последовательно, а произвольный доступ выполняется не слишком часто, MySQL Cluster может достигать более высокой производительности при использовании дисков, у которых возможности кэширования ограничены по сравнению с традиционными дисковыми хранилищами, используемыми в реляционных СУБД.

Подсчитать объем памяти, необходимый для узла данных, можно с помощью приведенной ниже формулы. Размер базы данных — это сумма размеров строк, умноженная на число строк в каждой таблице. Помните, что при использовании дискового хранилища для неиндексированных столбцов для подсчета объема памяти следует считать только индексированные столбцы.

(Размер_БД × Число_реплик × 1,1) / Число_Узлов_Данных

Это упрощенная формула для грубого подсчета. Информацию о дополнительных аспектах, которые следует учитывать при планировании использования памяти кластером, можно найти в онлайновой документации MySQL Cluster Reference Manual.

Можно также использовать сценарий Perl *NDB_size.pl*, включенный в большинство дистрибутивов. Этот сценарий подключается к работающему серверу MySQL, анализирует все существующие таблицы в базах данных и подсчитывает, сколько памяти потребуется для них в кластере MySQL. Это удобно, так как позволяет сначала создать и заполнить таблицы на обычном сервере MySQL, проверить конфигурацию памяти, а затем настроить кластер и загрузить в него данные. Полезно также периодически выполнять этот сценарий, чтобы следить за изменениями схемы, которые могут привести к проблемам с памятью. В листинге 15-1 приведен отчет для базы данных с одной таблицей. Чтобы подчитать полный размер базы данных, умножьте размер строки данных из строки Summary на число строк. В примере 15-1 (для MySQL 5.1) мы имеем 84 байта на строку для данных и индекса. Чтобы сохранить таблицу с 64000 строк, нам понадобится 5376000 байт памяти.

Если сценарий генерирует ошибку с сообщением об отсутствии модуля *Class/MethodMaker.pm*, установите этот класс в своей системе. Например, в Ubuntu это можно сделать следующей командой:

```
sudo apt-get install libclass-mothodmaker-perl.
```

Лист. 15-1. Проверка размера базы данных сценарием NDB_size.pl

```
cbell@cbell-mini:~/mysql-cluster-gpl-7.0.13-linux-i686-glibc23/bin$ ./NDB_size.pl \
--database=cluster_test --user=root
NDB_size.pl report for database: 'cluster_test' (1 tables)
----------------------------------------------------------
Connected to: DBI:mysql:host=localhost

Including information for versions: 4.1, 5.0, 5.1

cluster_test.City
-----------------

DataMemory for Columns (* means varsized DataMemory):
      Column Name    Type  Varsized    Key      4.1   5.0   5.1
         district          char(20)              20    20    20
       population           int(11)               4     4     4
            ccode           char(3)               4     4     4
             name          char(35)              36    36    36
               id           int(11)    PRI        4     4     4
                                                 --    --    --
Fixed Size Columns DM/Row                        68    68    68
 Varsize Columns DM/Row                           0     0     0

DataMemory for Indexes:
      Index Name     Type              4.1      5.0      5.1
         PRIMARY     BTREE             N/A      N/A      N/A
                                       --       --       --
       Total Index DM/Row               0        0        0

IndexMemory for Indexes:
               Index Name              4.1      5.0      5.1
                  PRIMARY              29       16       16
                                       --       --       --
           Indexes IM/Row              29       16       16

Summary (for THIS table):

                                       4.1      5.0      5.1
    Fixed Overhead DM/Row              12       12       16
           NULL Bytes/Row               0        0        0
           DataMemory/Row              80       80       84
   (Includes overhead, bitmap and indexes)

  Varsize Overhead DM/Row               0        0        8
   Varsize NULL Bytes/Row               0        0        0
       Avg Varside DM/Row               0        0        0

                No. Rows                3        3        3
```

```
        Rows/32kb DM Page              408      408      388
Fixedsize DataMemory (KB)               32       32       32

Rows/32kb Varsize DM Page                0        0        0
  Varsize DataMemory (KB)                0        0        0

Rows/8kb IM Page                     282      512      512
IndexMemory (KB)                     8        8        8

Parameter Minimum Requirements
------------------------------
* indicates greater than default

              Parameter   Default        4.1      5.0      5.1
        DataMemory (KB)     81920         32       32       32
     NoOfOrderedIndexes       128          1        1        1
            NoOfTables        128          1        1        1
       IndexMemory (KB)     18432          8        8        8
   NoOfUniqueHashIndexes       64          0        0        0
          NoOfAttributes     1000          5        5        5
            NoOfTriggers      768          5        5        5
```

Обратите внимание на то, что, хотя в листинге 15-1 используется очень простая таблица, выходные данные содержат не только размер строки, но и некоторые сведения о таблицах базы данных. В отчете также показана статистика индексирования, а это ключевой механизм, используемый в кластере для обеспечения высокой производительности.

Этот сценарий показывает разные требования к памяти для разных версий MySQL. Так можно увидеть особенности старых версий MySQL Cluster.

Секционирование данных

Один из наиболее важных аспектов MySQL Cluster — секционирование данных. Данные разделяются по горизонтали. Т.е. строки автоматически разделяются между узлами данных с использованием функции распределения строк. В основе этой функции лежит алгоритм хеширования, использующий первичный ключ таблицы. В ранних версиях MySQL использовался внутренний механизм разделения, но в MySQL 5.1 и выше можно использовать собственную функцию разделения данных. Если вы используете собственную функцию, вам следует создать функцию, гарантирующую, что данные равно распределяются между узлами данных.

Если таблица не имеет первичного ключа, MySQL Cluster добавит суррогатный первичный ключ.

Секционирование позволяют MySQL Cluster достигать более высокой производительности при выполнении запросов, так как обеспечивает распределение запросов между узлами данных. Запрос исполняется гораздо

быстрее, если будет получать данные от нескольких узлов, а не от одного. Например, можно выполнить следующий запрос на каждом узле данных, чтобы получить сумму столбцов на каждом узле и суммировать полученные результаты:

```
SELECT SUM(population) FROM cluster_db.city;
```

Данные, распределенные по узлам данных, защищены от сбоев, если имеется несколько реплик (копий) данных. Если данные распределяются по нескольким узлам данных для обеспечения параллельного выполнения запросов, следует иметь минимум две реплики каждой строки, чтобы обеспечить отказоустойчивость кластера.

Управление транзакциями

Еще один аспект поведения MySQL Cluster, отличающийся от сервера MySQL, относится к транзакционным операциям. Как говорилось ранее, MySQL Cluster координирует транзакционные изменения на всех узлах данных. Для этого используется два подпроцесса: *координатор транзакций* и *обработчик локальных запросов*.

Координатор транзакций обрабатывает распределенные транзакции и другие операции с данными на глобальном уровне. Обработчик локальных запросов управляет данными и транзакциями, обрабатываемыми на локальных узлах данных кластера, и действует как координатор двухэтапных фиксаций на узлах данных.

Каждый узел данных может быть координатором транзакций (это можно настроить). Когда приложение выполняет транзакцию, кластер подключается к координатору транзакций на одном из узлов данных. По умолчанию выбирается ближайший узел данных, определенный сетевым уровнем кластера. Если доступно несколько подключений на одном расстоянии, координатор транзакций выбирается с помощью кругового алгоритма.

После этого выбранный координатор транзакций отправляет запрос каждому узлу данных, а обработчик локальных запросов выполняет запрос, координируя двухэтапную фиксацию. Когда все узлы данных подтвердят транзакцию, координатор транзакции фиксирует ее.

MySQL Cluster поддерживает уровень изоляции транзакций с фиксацией при чтении. Это значит, что когда во время выполнения транзакции появляются изменения, только зафиксированные изменения могут быть прочитаны, пока транзакция не завершена. Так MySQL Cluster обеспечивает согласованность данных во время выполнения транзакций.

Подробные сведения о транзакциях в MySQL Cluster и список важных ограничений, относящихся к транзакциям, можно найти в главе онлайнового руководства MySQL Reference Manual, посвященной MySQL Cluster.

Онлайн-обслуживание

В MySQL 5.1 и выше можно выполнять определенные сервисные операции, не отключая сервер и не блокируя части системы или базы данных. Ниже вкратце перечислены и описаны некоторые сервисные онлайн-операции в MySQL Cluster и указаны версии, поддерживающие эти операции.

- *Резервное копирование (5.0 и выше).* При помощи консоли управления NDB можно делать резервные копии данных кластера, создавая мгновенные снимки (операция, не требующая блокирования). Эта операция включает копию метаданных (имена и определения всех таблиц), данные из таблиц и журнал транзакций (записи изменений со временем). Отличие от резервных копий mysql dump состоит в том, что не требуется блокирование и не используется сканирование таблиц для чтения записей. Для восстановления данных используется специальная утилита NDB_restore.
- *Добавление и удаление индексов (5.1 и выше).* Можно использовать ключевое слово ONLINE для выполнения команд CREATE INDEX и DROP INDEX в режиме онлайн. Когда запрашивается онлайновая операция, эта операция не является операцией копирования, т.е. она не делает копии данных для их индексации, поэтому после нее индексы не требуется создавать заново. Одним из преимуществ такого подхода является то, что можно продолжать выполнение транзакций во время операций по изменению таблиц, а изменяемые таблицы не блокируются, и к ним могут обращаться другие узлы SQL. Однако таблица блокируется для других запросов узла SQL, выполняющих изменение.

В MySQL 5.1.7 и выше операции добавления и удаления индексов выполняются в режиме онлайн, если индексы созданы только для столбцов переменной ширины.

- *Изменение таблицы (6.2 и выше).* Можно использовать ключевое слово ONLINE для выполнения команды ALTER TABLE в режиме онлайн. Эта операция тоже не является операцией копирования и имеет те же преимущества, что и онлайновое добавление индексов. Кроме того, в MySQL 7.0 и выше можно выполнять реорганизацию данных разделов в режиме онлайн, используя команду REORGANIZE PARTITION, если не используется параметр INTO (partition_definitions).

В настоящий момент онлайновое изменение типов данных и значений столбцов, назначенных по умолчанию, не поддерживается.

- *Добавление узлов данных и групп узлов (7.0 и выше).* В режиме онлайн можно управлять расширением узлов данных, выполняя масштабирование или заменяя узлы после сбоев. Эта процедура подробно описана в документации. Коротко говоря, она включает изменение файла конфигурации, перезапуск демона управления NDB, перезапуск существующих

узлов данных, запуск новых узлов данных и последующую реорганизацию разделов.

Подробнее о MySQL Cluster, его архитектуре и функциях, появившихся в версии 7.0, см. в документации по адресу *http://www.mysql.com/why-mysql/white-papers/ mysql_wp_cluster7_architecture.php*.

Пример конфигурации

В этом разделе мы представим пример конфигурации кластера MySQL, включающего два узла данных на двух системах и имеющего сервер MySQL и узел управления NDB на третьей системе. Мы представим примеры упрощенной установки узлов данных. Конфигурация системы показана на рис. 15-4.

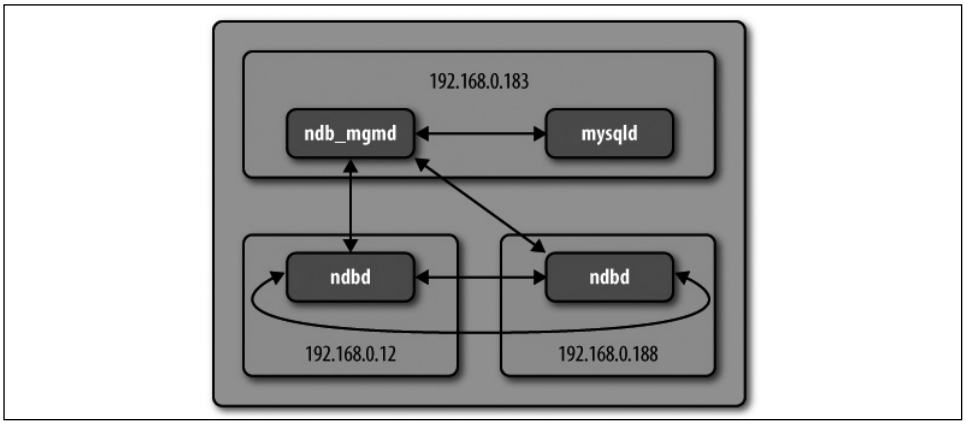


Рис. 15-4. Пример конфигурации кластера

Видно, что один узел содержит и демона управления NDB, и узел SQL (сервер MySQL). Есть также два узла данных, каждый на собственной системе. Для создания базовой конфигурации MySQL Cluster с повышенной доступностью или производительностью потребуется минимум три компьютера.

Это минимальная конфигурация MySQL Cluster и, если число реплик равняется двум, минимальная конфигурация для обеспечения отказоустойчивости. Если реплика только одна, такая конфигурация будет поддерживать разделение для обеспечения лучшей производительности, но не будет отказоустойчивой.

Обычно допустимо запускать демон управления NDB на том же узле, что и сервер MySQL, но если число узлов данных будет большим или требуется обеспечить более высокую отказоустойчивость, демон лучше запустить на отдельной системе.

Начало работы

MySQL Cluster можно получить на странице загрузок MySQL (*http://www.mysql.com/downloads/cluster*). Это ПО с открытым исходным кодом, как и сер-

вер MySQL. Можно скачать двоичный дистрибутив или файл установки для некоторых из наиболее популярных платформ. Можно также скачать сходный код и построить кластер на собственной платформе. Предварительно рекомендуем просмотреть замечания по конкретным платформам и операционным системам.

Для установки выполните обычную процедуру, описанную в онлайновом руководстве MySQL Reference Manual. Кроме одного специального каталога, утилиты NDB устанавливаются в то же место, что и двоичные файлы сервера MySQL.

Прежде чем приступать к исследованию нашего примера, рассмотрим некоторые общие концепции, относящиеся к настройке кластера MySQL. Конфигурация кластера поддерживается демоном управления NDB и считывается (первоначально) из файла конфигурации. Существует много параметров для настройки различных аспектов кластера, но сейчас мы остановимся на минимальной конфигурации.

Файл конфигурации имеет несколько разделов. Как минимум, необходимо включить в него следующие разделы:

- *mysqld* — знакомый вам раздел файла конфигурации, применяющийся к серверу MySQL, определяющий параметры узла SQL.
- *NDB DEFAULT* — раздел глобальных параметров, применяемых по умолчанию. Включайте в этот раздел все параметры, которые хотите применять ко всем узлам данных и узлам управления. Заметьте, что в названии раздела содержится пробел, а не знак подчеркивания.
- *NDB_MGMD* — раздел параметров демона управления NDB.
- *NDBD* — необходимо добавить один раздел с таким названием для каждого узла данных.

В примере 15-2 приведен минимальный файл конфигурации, соответствующий конфигурации, показанной на рис. 15-4.

Лист. 15-2. Минимальный файл конфигурации

```
[NDBD DEFAULT]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[NDB_MGMD]
Hostname=192.168.0.183
DataDir= /var/lib/mysql-cluster

[NDBD]
Hostname=192.168.0.12

[NDBD]
Hostname=192.168.0.188

[MYSQLD]
Hostname=192.168.0.183
```

Этот пример включает минимальный набор переменных для простого кластера, имеющего два узла, с поддержкой репликации. Поэтому параметру NoOfReplicas задано значение 2. Заметьте, что переменная datadir имеет значение /var/lib/mysql-cluster. Можно указать любой другой путь, но большинство установок MySQL Cluster использует этот каталог.

Наконец, обратите внимание на то, что мы указали имя хоста каждого узла. Это важно, так как демон управления NDB должен знать расположение всех узлов кластера. Если вы скачали и установили MySQL Cluster и хотите продолжать работу, внесите необходимые изменения в имена хостов, чтобы они соответствовали нашему примеру.

Поместите файл конфигурации кластера в каталог */var/lib/mysql-cluster* и дайте ему имя *config.ini* (стандартные имя и расположение этого файла).

Не обязательно устанавливать полный пакет MySQL Cluster на узлах данных. Как будет показано далее, на узлах данных требуется только демон NDBd.

Запуск MySQL Cluster

Для запуска MySQL Cluster требуется выполнить определенную последовательность команд. Мы опишем процедуры для этого примера по шагам, но сначала вкратце рассмотрим процесс в целом:

1. Запуск управляющего узла (узлов).
2. Запуск узлов данных.
3. Запуск серверов MySQL (узлов SQL).

Для нашего примера мы сначала запустим узел управления NDB на системе 192.168.0.183. Затем запустим узлы данных (192.168.0.12 и 192.168.0.188, в любом порядке). Когда узлы данных запущены, можно запустить сервер MySQL (192.168.0.183), и, после небольшой задержки на загрузку, кластер готов к использованию.

Запуск управляющего узла

Первый узел, который нужно запустить — демон управления NDB, носящий имя NDB_mgmd. Он расположен в подкаталоге *libexec* каталога MySQL. Например, в Ubuntu это будет каталог */usr/local/mysql/libexec*.

Запуска демона управления NDB выполняется с правами суперпользователя и указанием параметров --initial и -f. Параметр --initial указывает кластеру, что это первый запуск, и мы хотим удалить любые конфигурации, оставшиеся от предыдущих установок. Параметр -f указывает демону, где искать файл конфигурации. В примере 15-3 показано, как запустить демона управления NDB для нашего примера.

Лист. 15-3. Запуск демона управления NDB

```
cbell@mysql-xps-400:/usr/local/mysql/bin$ sudo ../libexec/NDB_mgmd --initial \
-f /var/lib/mysql-cluster/config.ini
```

```
2010-03-25 09:10:28 [MgmtSrvr] INFO
     -- NDB Cluster Management Server. mysql-5.1.44 NDB-7.0.14
2010-03-25 09:10:29 [MgmtSrvr] INFO
     -- Reading cluster configuration from '/var/lib/mysql-cluster/config.ini'
```

Рекомендуем всегда указывать параметр -f при первом запуске, так как в разных установках файл конфигурации может по умолчанию находиться в разных каталогах. Выяснить, где по умолчанию ищется файл конфигурации, можно, выполнив команду NDB_mgmd --help и найдя фразу «Default options are read from». При последующих запусках демона параметр -f указывать не обязательно.

Запуск консоли управления

Хотя сейчас это и не обязательно, рекомендуем запустить консоль управления NDB и убедиться, что демон управления NDB корректно считал конфигурацию. Имя консоли управления — NDB_mgm, а расположена она в подкаталоге *bin* каталога MySQL. Чтобы просмотреть конфигурацию, выполните команду SHOW, как показано в примере 15-4.

Лист. 15-4. Первый запуск консоли управления NDB

```
cbell@mysql-xps-400:/usr/local/mysql/bin$ ./NDB_mgm
-- NDB Cluster -- Management Client --
NDB_mgm> SHOW
Connected to Management Server at: 192.168.0.183:1186
Cluster Configuration
---------------------
[NDBd(NDB)]  2 node(s)
id=2 (not connected, accepting connect from 192.168.0.188)
id=3 (not connected, accepting connect from 192.168.0.12)

[NDB_mgmd(MGM)]  1 node(s)
id=1  @192.168.0.183 (mysql-5.1.44 NDB-7.0.14)

[mysqld(API)]  1 node(s)
id=4 (not connected, accepting connect from 192.168.0.183)

NDB_mgm>
```

Эта команда показывает узлы данных и их IP-адреса, а также сведения о демоне управления NDB и узле SQL. Сейчас самое время проверить, все ли узлы имеют правильные IP-адреса и все ли узлы данных загружены. Если вы изменили конфигурацию кластера, но видите здесь старые значения, скорее всего демон управления NDB не считал новый файл конфигурации.

Выходные данные из приведенного примера говорят о том, что демон управления NDB загружен и готов к работе. Если бы это было не так, команда SHOW завершилась бы с сообщением об ошибке подключения. Если появилась такая ошибка, убедитесь, что клиент управления NDB запущен

на том же сервере, что и демон управления NDB. Если они запущены на разных серверах, используйте параметр --NDB-connectstring и укажите IP-адрес или имя системы, на которой запущен демон управления NDB.

Наконец, обратите внимание на идентификаторы узлов. Они понадобятся для выполнения команд, относящихся к конкретным узлам, из консоли управления NDB. В любое время можно выполнить команду HELP, чтобы просмотреть другие доступные команды. Также необходимо знать идентификаторы узлов SQL, чтобы корректно их запустить.

В файле *config.ini* можно указать идентификаторы каждого узла кластера, используя параметр --NDB-nodeid.

Можно использовать команду STATUS, чтобы просмотреть состояние узлов. Выполните команду ALL STATUS, чтобы просмотреть состояние всех узлов, или команду id-узла STATUS, чтобы просмотреть состояние конкретного узла. Эта команда удобна для наблюдения за запуском кластера, так как ее выходные данные содержат информацию о том, на каком этапе запуска находится узел данных. Подробнее об этапах запуска узлов данных см. в документации MySQL Reference Manual.

Запуск узлов данных

После запуска демона управления NDB можно приступать к запуску узлов данных. Однако сначала рассмотрим минимальную установку, необходимую для узла данных NDB.

Все, что необходимо для установки узла данных NDB — демон узла данных NDB (NDBd), скомпилированный для соответствующей операционной системы. Сначала создайте папку */var/lib/mysql-cluster*, затем скопируйте в нее исполняемый файл NDBd, и все! Этот процесс создания узлов данных очень легко автоматизировать с помощью сценариев.

Узлы данных (NDBd) можно запускать при помощи параметра --initial-start, который указывает, что это первый запуск кластера. Также необходимо указать параметр --NDB-connectstring с IP-адресом демона управления NDB. В примере 15-5 показан первый запуск узла данных. Проделайте то же самое для каждого узла данных.

Лист. 15-5. Запуск узла данных

```
cbell@mysql-mini:~/mysql-cluster-gpl-7.0.13-linux-x86_64-glibc23/bin$
 sudo ./NDBd --initial-start --NDB-connectstring=192.168.0.183
2010-03-25 09:04:18 [NDBd] INFO
   -- Configuration fetched from '192.168.0.183:1186', generation: 1
```

Если запускается новый узел данных, или параметры узла данных сброшены, или нужно выполнить восстановление после сбоя, можно указать параметр --initial, чтобы принудительно удалить любую существующую конфигурацию и кэшированные данные и запросить новую копию у демона управления NDB.


Осторожно используйте параметр --initial. Он по-настоящему удаляет данные!

Вернитесь в консоль управления и проверьте состояние (листинг 15-6).

Лист. 15-6. Состояние узлов данных

```
NDB_mgm> SHOW
Cluster Configuration
---------------------
[NDBd(NDB)]  2 node(s)
id=2  @192.168.0.188 (mysql-5.1.41 NDB-7.0.13, Nodegroup: 0, Master)
id=3  @192.168.0.12 (mysql-5.1.41 NDB-7.0.13, Nodegroup: 0)

[NDB_mgmd(MGM)]  1 node(s)
id=1  @192.168.0.183 (mysql-5.1.44 NDB-7.0.14)

[mysqld(API)]  1 node(s)
id=4 (not connected, accepting connect from 192.168.0.183)
```

Как видите, узлы данных успешно запущены, так как показана информация о их демонах. Также видно, что один из узлов выбран главным для репликации кластера. Так как в файле конфигурации мы указали число реплик, равное 2, у нас есть две копии данных. Не путайте главный узел с главным сервером репликации MySQL. Подробнее о различиях будет рассказано далее в этой главе.

Запуск узлов SQL

Когда все узлы данных запущены, можно подключить наш узел SQL. Есть несколько параметров, которые необходимо указать, чтобы сервер MySQL мог подключиться к кластеру NDB. Чаще всего их указывают в файле *my.cnf*, но можете сделать это в командной строке при загрузке, если запускаете сервер таким способом.

- *NDBcluster* — сообщает серверу, что нужно включить механизм хранилища кластера NDB;
- *NDB_connectstring* — сообщает серверу расположение демона управления NDB;

vNDB_nodeid u server_id — обычно имеют значения идентификатора узла. Идентификатор узла можно найти в выходных данных команды SHOW в консоли управления NDB.

В примере 15-7 показана правильная последовательность запуска узла SQL для нашего примера.

Лист. 15-7. Запуск узла SQL

```
cbell@mysql-xps-400:/usr/local/mysql/bin$ sudo ../libexec/mysqld -NDBcluster \
--console -umysql
100325 9:14:21 [Note] Plugin 'FEDERATED' is disabled.
100325 9:14:21 InnoDB: Started; log sequence number 0 1112278176
```

```
100325 9:14:21 [Note] NDB: NodeID is 4, management server '192.168.0.183:1186'
100325 9:14:22 [Note] NDB[0]: NodeID: 4, all storage nodes connected
100325 9:14:22 [Note] Starting Cluster Binlog Thread
100325 9:14:22 [Note] Event Scheduler: Loaded 0 events
100325 9:14:23 [Note] NDB: Creating mysql.NDB_schema
100325 9:14:23 [Note] NDB: Flushing mysql.NDB_schema
100325 9:14:23 [Note] NDB Binlog: CREATE TABLE Event: REPL$mysql/NDB_schema
100325 9:14:23 [Note] NDB Binlog: logging ./mysql/NDB_schema (UPDATED,USE_WRITE)
100325 9:14:23 [Note] NDB: Creating mysql.NDB_apply_status
100325 9:14:23 [Note] NDB: Flushing mysql.NDB_apply_status
100325 9:14:23 [Note] NDB Binlog: CREATE TABLE Event: REPL$mysql/NDB_apply_status
100325 9:14:23 [Note] NDB Binlog: logging ./mysql/NDB_apply_status
(UPDATED,USE_WRITE)
2010-03-25 09:14:23 [NdbApi] INFO   -- Flushing incomplete GCI:s < 65/17
2010-03-25 09:14:23 [NdbApi] INFO   -- Flushing incomplete GCI:s < 65/17
100325 9:14:23 [Note] NDB Binlog: starting log at epoch 65/17
100325 9:14:23 [Note] NDB Binlog: NDB tables writable
100325 9:14:23 [Note] ../libexec/mysqld: ready for connections.
Version: '5.1.44-NDB-7.0.14-debug' socket: '/var/lib/mysql/mysqld.sock'
port: 3306 Source distribution
```

Выходные данные включают дополнительные комментарии о подключении кластера NDB, журналах и состоянии. Если вы не видите их или видите ошибки, убедитесь, что запустили узел SQL с правильными параметрами. Особо важно сообщение, указывающее ID узла и сервер управления. Если запущено несколько серверов управления, убедитесь, что узел SQL взаимодействует с нужным сервером.

Если узел SQL запустился корректно, вернитесь в консоль управления и проверьте состояние всех узлов (листинг 15-8).

Лист. 15-8. Состояние работающего кластера

```
NDB_mgm> SHOW Cluster Configuration
----------------------------------
[NDBd(NDB)]  2 node(s)
id=2  @192.168.0.188 (mysql-5.1.41 NDB-7.0.13, Nodegroup: 0, Master)
id=3  @192.168.0.12 (mysql-5.1.41 NDB-7.0.13, Nodegroup: 0)

[NDB_mgmd(MGM)]  1 node(s)
id=1  @192.168.0.183 (mysql-5.1.44 NDB-7.0.14)

[mysqld(API)]  1 node(s)
id=4  @192.168.0.183 (mysql-5.1.44 NDB-7.0.14)
```

Как видите, все узлы подключены и работают. Если выходные данные включают какую-то информацию, кроме приведенной в примере, значит, произошел сбой в последовательности запуска узлов. Проверьте журналы для каждого узла, чтобы выяснить, в чем проблема. Наиболее распростра-

ненная причина сбоев — проблемы с подключением (например, блокировка брандмауэром). Узлы NDB по умолчанию используют порт 1186.

Файлы журналов для узлов данных и демона управления NDB находятся в каталоге данных. Журналы узла SQL расположены в каталоге сервера MySQL.

Тестирование кластера

Теперь наш кластер работает, и мы можем выполнить простой текст (листинг 15-9), чтобы убедиться, что мы можем создать базу данных и таблицы, используя механизм хранилища NDBcluster.

Лист. 15-9. Проверка кластера

```
mysql> create database cluster_db;
Query OK, 1 row affected (0.06 sec)

mysql> create table cluster_db.t1 (a int) engine=NDBCLUSTER;
Query OK, 0 rows affected (0.31 sec)

mysql> show create table cluster_db.t1 \G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL
) ENGINE=NDBcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql> insert into cluster_db.t1 VALUES (1), (100), (1000);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> select * from cluster_db.t1 \G
*************************** 1. row ***************************
a: 1
*************************** 2. row ***************************
a: 1000
*************************** 3. row ***************************
a: 100
3 rows in set (0.00 sec)
```

Запустив кластер, можете поэкспериментировать с ним, загружая данные и выполняя запросы. Попробуйте сымитировать сбой на одном из узлов данных во время обновления данных и перезапустить его, чтобы убедиться, что потеря одного из узлов не влияет на доступность.

Отключение кластера

Отключение кластера, как и запуск, выполняется в определенной последовательности:

1. Если настроена репликация между кластерами, дождитесь завершения операций подчиненными серверами, и остановите репликацию.
2. Отключите узлы SQL (mysqld).
3. Выполните команду SHUTDOWN в консоли управления NDB.
4. Выйдите из консоли управления NDB.

Если между двумя и более кластерами выполняется репликация MySQL, первый этап необходим, чтобы подчиненные серверы синхронизировались с главным перед остановкой узлов SQL.

Команда SHUTDOWN, выполненная в консоли управления NDB, завершает работу всех узлов данных и демона управления NDB.

Обеспечение высокой доступности

Основная мотивация для использования средств обеспечения высокой доступности — необходимость поддержания работоспособности. Для СУБД это значит, что мы всегда должны иметь возможность доступа к данным. MySQL Cluster служит именно для этого. В MySQL Cluster для обеспечения высокой доступности используется распределение данных по узлам данных (что снижает риск потери данных при выходе из строя одного узла), репликация между репликами кластера, автоматическое восстановление потерянных узлов данных, обнаружение сбоев узлов данных при помощи тактовых импульсов, а также локальные и глобальные контрольные точки, обеспечивающие непротиворечивость данных.

Рассмотрим некоторые характеристики СУБД с высокой доступностью. Чтобы считаться высокодоступной, СУБД (как и любая система) должна удовлетворять следующим требованиям:

99,999% времени в работоспособном состоянии;

- отсутствие единой точки сбоя;
- обработка отказов;
- отказоустойчивость.

На практике, 99,999% времени в работоспособном состоянии означает, что система должна быть доступна всегда. Другими словами, сервер баз данных должен работать без перерывов. Сервер никогда не должен отключаться ни из-за сбоев компонентов, ни на обслуживание. Все операции по обслуживанию и восстановлению должны выполняться в режиме онлайн, без перебоев с доступом.

Это идеальная ситуация, редко требуемая на практике, за исключением некоторых критичных областей. Кроме того, не избежать коротких периодов профилактического обслуживания (отсюда и асимптотическое приближение к 100%). Интересно отметить принятое количество времени работы в отношении к числу девяток в процентном значении. В табл. 15-1 указано приемлемое время простоя в течение календарного года для каждого уровня доступности.

Табл. 15-1. Приемлемое время простоя

Время работы	Приемлемое время простоя
99,000%	3,65 дня
99,900%	8,76 часов
99,990%	52,56 минут
99,999%	5,26 минут

Чем больше девяток в процентном значении, тем меньше приемлемое время простоя. Для значения 99,999% все операции по обслуживанию необходимо выполнять без прерывания работы, за исключением нескольких минут в году. MySQL Cluster удовлетворяет этим требованиям несколькими способами, включая чередующиеся перезапуски узлов данных, возможность выполнять некоторые сервисные операции без остановки и множественные каналы доступа к данным (узлы SQL и приложения, подключающиеся через API NDB).

Отсутствие единой точки сбоя означает, что отказ ни одного из компонентов системы не поставить под угрозу доступность службы. Это можно обеспечить с помощью MySQL Cluster, настроив узлы каждого типа для получения избыточности. В небольшом примере из предыдущего раздела было два узла данных. Так данные были защищены от сбоя одного узла данных. Однако у нас был только один узел управления и один узел SQL. В идеале следовало добавить узлы для обеспечения избыточности этих функций. MySQL Cluster позволяет иметь несколько узлов SQL, так что если узел управления выйдет из строя, кластер сможет продолжать работу.

Обработка отказов означает, что если компонент выйдет из строя, другой может заменить его. В случае узлов данных MySQL, обработка отказов происходит автоматически, если кластер содержит несколько реплик данных. Если узел данных с одной репликой выходит из строя, доступ к данным не прекращается. После перезапуска отсутствующего узла данных данные копируются на него из другой реплики. В случае узлов SQL, так как данные в действительности хранятся на узлах данных, любой узел SQL можно заменить другим.

Если из строя выходит узел управления NDB, кластер может продолжать работу без него, а новый узел управления можно запустить в любое время (если не изменилась конфигурация).

Кроме того, можно применять обычные решения для обеспечения высокой доступности, обсуждаемые в предыдущих главах, включая репликацию и автоматическую обработку отказов между целыми кластерами. О репликации кластера см. далее в этой главе.

Отказоустойчивость обычно ассоциируется с оборудованием, таким как резервные источники питания, дополнительные каналы связи и т. д. Для программного обеспечения отказоустойчивость — побочный результат надежной обработки отказов. Для MySQL Cluster это означает, что система

может выдержать определенное число сбоев и продолжать предоставлять доступ к данным. Как и в случае с RAID-системой, теряющей два диска из одного массива, потеря нескольких узлов данных может привести к невосстановимому сбою. Однако при грамотном планировании можно настроить кластер так, чтобы снизить такой риск. Также риск можно снизить, проводя мониторинг и активное обслуживание.

Отказоустойчивость MySQL Cluster достигается активным управлением узлами кластера. Для проверки работоспособности служб используются опросы тактовыми импульсами, и когда обнаруживается вышедший из строя узел, выполняется его восстановление.

Механизмы журналирования MySQL Cluster также предоставляют возможности восстановления для обработки отказов и обеспечения отказоустойчивости. Локальные и глобальные контрольные точки обеспечивают непротиворечивость данных в кластере. Эта информация критически важна для быстрого восстановления после сбоев узлов данных. Уникальные особенности контрольных точек позволяют восстанавливать не только данные, но и узлы. Подробнее об этом рассказывается далее в этой главе.

На рис. 15-5 показана схема кластера MySQL, настроенного для обеспечения высокой доступности веб-служб.

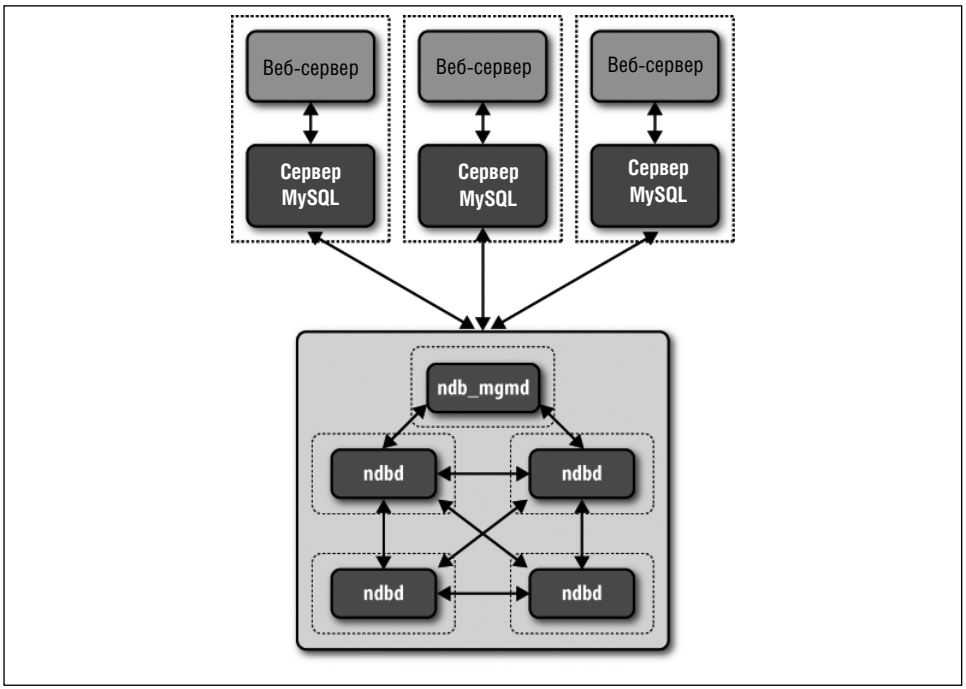


Рис. 15-5. Кластер MySQL с высокой доступностью

Пунктиром обозначены границы систем. Эти компоненты должны работать на собственном оборудовании для обеспечения избыточности. Также

следует настроить четыре узла данных для хранения двух реплик. На рисунке не показаны дополнительные компоненты, взаимодействующие с приложением, такие как балансировщик нагрузки, распределяющий нагрузку между веб-серверами и серверами MySQL.

Настраивая кластер MySQL для обеспечения высокой доступности, следует применять все приведенные ниже рекомендации. Подробнее о них мы поговорим далее в этой главе, когда будем обсуждать способы обеспечения высокой производительности MySQL Cluster.

- Используйте несколько реплик на узлах данных, работающих на выделенном оборудовании.
- Используйте дополнительные каналы связи для защиты от сбоев сети.
- Используйте несколько узлов SQL.
- Используйте несколько узлов данных для повышения производительности и децентрализации данных.

Восстановление системы

Существует два типа восстановления системы. Первый — когда сервер отключается для выполнения обслуживания и прочих запланированных событий. Второй — при неожиданной потере функциональности системы. К счастью, MySQL Cluster предоставляет механизмы для восстановления функциональности даже в самых худших ситуациях.

После корректного завершения работы MySQL Cluster запуск производится с контрольных точек из журнала. Это автоматический и нормальный этап в последовательности загрузки. При запуске система загружает наиболее свежие данные из локальных контрольных точек для каждого узла данных, тем самым восстанавливая данные до состояния на момент последнего мгновенного снимка. Когда узлы данных загрузят данные из своих локальных контрольных точек, система выполняет журнал повторов до самой свежей глобальной контрольной точки, синхронизируя данные до последнего изменения, внесенного перед завершением работы. Эта процедура одинакова для перезапуска после преднамеренного завершения работы и для полной перезагрузки системы после сбоя.

Если вам не понятно, как восстановление может происходить при загрузке, напомним, что базы данных MySQL Cluster хранятся в памяти, и поэтому при запуске данные должны быть заново загружены с диска. Для этого выполняется загрузка данных до наиболее свежей контрольной точки.

При восстановлении системы после катастрофического сбоя или в качестве корректировки можно также восстановить данные из резервной копии. Как говорилось ранее, для восстановления данных можно запустить из консоли управления NDB утилиту NDB_restore и восстановить последнюю резервную копию.

Чтобы выполнить полное восстановление системы из резервной копии, сначала следует перевести кластер в однопользовательский режим, выполнив в консоли управления NDB следующую команду:

```
ENTER SINGLE USER MODE node-id
```

node-id — идентификатор узла данных, который будет использоваться утилитой NDB_restore. Подробнее об однопользовательском режиме и подключении утилит на основе API см. в онлайновой документации.

Выполнив восстановление системы, можно восстановить данные на каждом узле данных кластера. После этого можно выйти из однопользовательского режима, и кластер будет готов к использованию. Чтобы выйти из однопользовательского режима, выполните в консоли управления NDB следующую команду:

```
EXIT SINGLE USER MODE
```

Подробнее о резервном копировании и восстановлении MySQL см. в онлайновой документации MySQL Reference Manual по следующим адресам:

- *http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-backup-using-management- client.html*
- *http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-programs-NDB-restore.html*

Восстановление узлов

Узлы могут выходить из строя по разным причинам, включая проблемы с сетью, оборудованием, памятью и операционной системой. Вкратце рассмотрим наиболее распространенные причины таких сбоев и способы восстановления узлов кластера. В этом разделе мы будем говорить об узлах данных, так как они наиболее важны с точки зрения доступности данных.

- *Оборудование*. Если оборудование компьютера выходит из строя, понятно, что узел, запущенный на этой системе, перестанет работать. В таком случае MySQL Cluster будет использовать другие реплики данных. Чтобы выполнить восстановление, замените вышедшее из строя оборудование и перезапустите узел данных.
- *Сеть*. Если связь с узлом пропадает из-за проблем с сетевым оборудованием или программных сбоев, он продолжает функционировать, но, так как не может связаться с другими узлами (посредством тактовых импульсов), MySQL Cluster помечает его как вышедший из строя и использует другие реплики, пока узел не вернется в сеть. Чтобы выполнить восстановление, замените вышедшее из строя сетевое оборудование и перезапустите узел данных.
- *Память*. Если в системе недостаточно памяти, у кластера может кончиться место для размещения данных. Это приведет к выходу из строя узла данных. Чтобы решить эту проблему, добавьте больше памяти или уве-

личьте значения параметров конфигурации, относящихся к выделению памяти, и выполните чередующийся перезапуск узла данных.

- *Операционная система*. Если конфигурация ОС приводит к проблемам в работе узла данных, устраните причины проблем и перезапустите узел данных.

Подробнее о высокой доступности баз данных и серверов MySQL см. по следующим адресам:

- *http://www.mysql.com/why-mysql/white-papers/mysql_db_high_availability.php*
- *http://www.mysql.com/why-mysql/white-papers/mysql_ha_solutions.php*

Репликация

Мы уже обсуждали вкратце различия между репликацией MySQL и репликацией внутри кластера. Репликацию MySQL Cluster иногда называют *внутренней репликацией кластера* или просто *внутренней репликацией*, чтобы подчеркнуть, что это не репликация MySQL, которую иногда называют *внешней репликацией*.

В этом разделе мы поговорим о внутренней репликации MySQL Cluster. Также мы рассмотрим, что происходит во время репликации MySQL (внешней репликации), когда она выполняется между кластерами MySQL, а не отдельными серверами MySQL.

Репликация внутри кластера и репликация MySQL

Мы уже говорили, что MySQL Cluster использует синхронную репликацию внутри кластера. Это нужно для поддержки двухэтапного протокола фиксации, обеспечивающего целостность данных. Репликация MySQL выполняется асинхронно, данные передаются в одну сторону, без подтверждения их получения перед фиксацией.

Репликация внутри кластера

Внутренняя репликация MySQL Cluster обеспечивает избыточность путем хранения нескольких копий данных (называемых *репликами*). Данные записываются на несколько узлов, прежде чем происходит подтверждение (фиксация) запроса. Это выполняется с использованием двухэтапной фиксации.

Такая репликация является синхронной и гарантирует непротиворечивость данных после подтверждения запроса или завершения фиксации.

Данные реплицируются *фрагментами*. Фрагмент — подмножество строк таблицы. Фрагменты распределяются между узлами данных в результате разделения, и копия фрагмента существует на другом узле данных в каждой реплике. Один из фрагментов назначается первичным и используется для выполнения запросов. Все остальные копии тех же данных считаются вторичными фрагментами. Во время обновления первым обновляется первичный фрагмент.

Репликация MySQL между кластерами

Репликация между кластерами выполняется очень просто. Если вы можете настроить репликацию между двумя серверами MySQL, то сможете настроить ее и между двумя кластерами, так как для этого нет специальных этапов настройки, дополнительных команд или параметров. Репликация MySQL выполняется так же, как между отдельными серверами, только данные хранятся в кластерах NDB. Однако есть некоторые ограничения на внешнюю репликацию. Перечислим их, чтобы вы знали о них, когда будете планировать внешнюю репликацию. Подробнее о внешней репликации см. в разделе «MySQL Cluster Replication» онлайновой документации MySQL Reference Manual.

- Внешняя репликация должна быть построчной.
- Внешняя репликация не может быть круговой.
- Внешняя репликация не поддерживает параметры auto_increment_*.
- Размер двоичного журнала может быть больше, чем при обычной репликации MySQL.

Использование репликация MySQL для репликации данных из одного кластера в другой позволяет использовать преимущества MySQL Cluster в каждом кластере и передавать данные между кластерами.

Можно ли использовать репликацию MySQL в кластере?

Данные с сервера MySQL Cluster можно реплицировать на сервер, не входящий в кластер (и наоборот). Для этого не требуется специальной настройки, кроме решения некоторых потенциальных конфликтов механизмов хранилища, аналогичных конфликтам при репликации между серверами MySQL с разными механизмами хранилища. В таких случаях используйте механизм хранилища, назначенный по умолчанию, и откажитесь от указания механизма хранилища в командах CREATE.

Репликация из кластера MySQL в систему, не входящую в кластер MySQL, требует создания специальной таблицы с именем *ndb_apply_status* для репликации зафиксированных эпох. Если эта таблица отсутствует на подчиненном сервере, репликация остановится с ошибкой и сообщением о том, что таблица *ndb_apply_status* не существует. Создать эту таблицу можно следующей командой:

```
CREATE TABLE `mysql`.`ndb_apply_status` (
  `server_id` INT(10) UNSIGNED NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  `log_name` VARCHAR(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
  `start_pos` BIGINT(20) UNSIGNED NOT NULL,
  `end_pos` BIGINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY (`server_id`) USING HASH
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
```

Репликация данных из кластера MySQL с использованием внешней репликации должна быть построчной, а главный узел SQL должен быть запущен с параметром --binlog-format=ROW или --binlog-format=MIXED. Действуют также все обычные требования к репликации MySQL (например, все узлы SQL должны иметь уникальные идентификаторы).

Внешняя репликация также требует некоторых специальных дополнений к процессу репликации, включая использование двоичного журнала кластера, потока вставки двоичного журнала и специальных системных таблиц для поддержки обновлений между кластерами. Кроме того, при внешней репликации немного по-другому обрабатываются транзакционные изменения. Подробнее об этом рассказывается в следующем разделе.

Архитектура репликации MySQL Cluster

Базовые принципы операций внешней репликации такие же, как у репликации MySQL. В частности, мы определяем главную и подчиненные роли для определенных кластеров. Главный кластер содержит исходную копию данных, а подчиненные получают копии этих данных в виде потока изменений данных.

В репликации MySQL Cluster используется несколько специальных таблиц из базы данных *mysql* на каждом узле SQL главной и подчиненных систем (независимо от того, является ли подчиненная система кластером или отдельным сервером). Эти таблицы создаются во время установки MySQL. Это таблица *NDB_binlog_index*, хранящая данные индекса для двоичного журнала (локального по отношению к узлу SQL), и таблица *NDB_apply_status*, хранящая записи операций, реплицированных на подчиненные системы. Таблица *NDB_apply_status* имеется на всех узлах SQL и синхронизируется, так что все ее копии одинаковы во всем кластере. Ее можно использовать для восстановления PITR вышедшей из строя подчиненной системы, входящей в кластер MySQL.

Эти таблицы обновляются новым потоком, который называют *потоком вставки двоичного журнала*. Этот поток обновляет главную систему при появлении изменений в механизме хранилища кластера NDB, записывая изменения, произошедшие в кластере. Также этот поток отвечает за перехват всех событий кластера, записанных в двоичный журнал, и запись в таблицу *NDB_binlog_index* всех событий, изменяющих, вставляющих или удаляющих данные. Поток выгрузки на главной системе отправляет события потоку ввода-вывода на подчиненной системе, используя репликацию MySQL.

Важное отличие внешней репликации MySQL Cluster состоит в том, что каждая эпоха обрабатывается как транзакция. Так как эпоха — это промежуток времени между контрольными точками, а MySQL Cluster обеспечивает непротиворечивость данных в каждой контрольной точке, эпохи считаются неделимыми и реплицируются с использованием того же механизма, что и транзакции в репликации MySQL. Информация о последней примененной

эпохе хранится в системных таблицах NDB, поддерживающих внешнюю репликацию между кластерами MySQL.

Одноканальная и многоканальная репликация

Подключение между главной и подчиненной системой в репликации MySQL называется *каналом*. По сути, канал — это сетевой протокол и среда, используемая для связи главной системы с подчиненными. Обычно создается только один канал, но для повышения доступности можно настроить вторичный канал, обеспечивающий отказоустойчивость. Такая репликация называется *многоканальной*. На рис. 15-6 показана схема многоканальной внешней репликации.

Многоканальная репликация значительно улучшает возможности по восстановлению после сбоев в сетевых каналах связи. В идеале следует использовать активный мониторинг для получения оповещений при выходе из строя сетевого подключения. Это можно выполнять разными способами, от сценариев, использующих простой механизм опроса, до оповещений и советников, доступных в MySQL Enterprise Monitor.

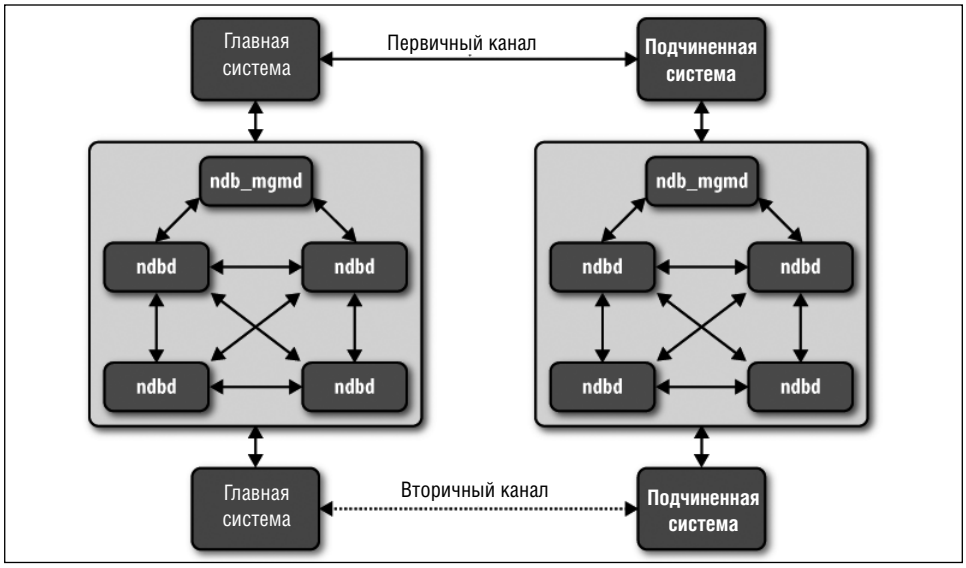


Рис. 15-6. Многоканальная внешняя репликация

Обратите внимание на то, что всего на рисунке четыре узла SQL. Главный кластер имеет два узла, выполняющих роли главных: первичный и вторичный. Аналогично, подчиненный кластер имеет первичный подчиненный узел и вторичный подчиненный узел. Первичная пара главный-подчиненный взаимодействует по одному каналу связи, вторичная пара — по-другому.

Не следует полностью доверять сетевому оборудованию. Даже коммутатор может выйти из строя. Использование разных кабелей в одной коммутируемой сети даст очень мало преимуществ. Лучше использовать полностью независимые наборы избыточных подключений и промежуточные сетевые компоненты для обеспечения настоящей избыточности сети.

Настройка многоканальной репликации не особо отличается от одноканальной (обычной) репликации MySQL. Однако обработка отказов выполняется немного по-другому. Идея в том, что не нужно запускать подчиненную систему во вторичном канале. Переход на использование вторичного канала требует некоторых специальных действий.

Используйте описанную ниже процедуру для запуска многоканальной внешней репликации с активным первичным каналом и вторичным каналом в режиме ожидания. Предполагается, что избыточное сетевое оборудование и каналы связи имеются и нормально работают.

1. Запустите первичную главную систему.
2. Запустите вторичную главную систему.
3. Подключите первичную подчиненную систему к первичной главной.
4. Подключите вторичную подчиненную систему к вторичной главной.
5. Запустите первичную подчиненную систему.

Не запускайте вторичную подчиненную систему. Если сделать это, появляется риск возникновения конфликтов первичных ключей и дублирования данных. Следует, однако, подключить вторичную подчиненную систему к вторичной главной, чтобы можно было быстро запустить вторичный канал, если первичный выйдет из строя.

Обработка отказов с переходом на вторичный канал требует другой процедуры. Недостаточно просто запустить вторичную подчиненную систему. Чтобы избежать повторной репликации тех же данных, необходимо сначала установить последнюю реплицированную эпоху и использовать ее для запуска репликации. Это выполняется следующим образом (обратите внимание, для хранения промежуточных результатов мы используем переменные):

Предварительно следует убедиться, что первичный канал действительно не работает. На всякий случай можно остановить первичную подчиненную систему.

1. Определите время последней глобальной контрольной точки, полученной подчиненной системой. Для этого нужно найти последнюю эпоху в таблице *ndb_apply_status* на первичной подчиненной системе.

```
SELECT @latest := MAX(epoch) FROM mysql.ndb_apply_status;
```

2. Получите строки, сохраненные в таблице *ndb_binlog_index* на первичной главной системе после сбоя. Это можно сделать следующим запросом:

```
SELECT @file := SUBSTRING_INDEX(File, '/', -1), @pos := Position
FROM mysql.ndb_binlog_index
WHERE epoch > @latest ORDER BY ASC LIMIT 1;
```

3. Синхронизируйте вторичный канал. Выполните на вторичной подчиненной системе следующую команду, где file — имя файла, а pos — позиция:

```
CHANGE MASTER TO MASTER_LOG_FILE = 'file', MASTER_LOG_POS = pos;
```

4. Запустите репликацию по второму каналу, выполнив на вторичной подчиненной системе следующую команду:

```
START SLAVE
```

Эта процедура обработки отказов переключит репликацию на использование вторичного канала. Если из строя вышли также какие-то из узлов SQL, их необходимо восстановить перед выполнением это процедуры.

Обеспечение высокой производительности

MySQL Cluster обеспечивает не только высокую доступность, но и высокую производительность. Мы уже рассмотрели многие из функций для повышения производительности, так как они повышают и доступность. В этом разделе рассмотрим некоторые другие возможности, обеспечивающие высокую производительность. В конце приведем список рекомендаций по настройке системы для обеспечения высокой производительности.

Ниже перечислены функции, поддерживающие высокую производительность MySQL Cluster. Многие из них мы рассмотрели в предыдущих разделах:

- *репликация между кластерами (глобальная избыточность)* — все данные реплицируются на удаленную систему, которую можно использовать для снижения нагрузки на основную систему;
- *репликация внутри кластера (локальная избыточность)* — позволяет считывать данные параллельно с нескольких узлов;
- *хранилище в основной памяти* — отсутствие необходимости ожидания записи на диск обеспечивает быструю обработку обновлений данных.

Повышение производительности

Существует три основных способа настройки системы для повышения производительности:

- убедитесь, что приложения работают максимально эффективно. Иногда для этого требуется изменение серверов баз данных (например, оптимизация конфигурации или изменение схемы базы данных), но часто само приложение можно изменить для обеспечения высокой производительности;

Запросы с объединениями часто требуют много времени на выполнение.

- расширьте возможности доступа к базам данных. Убедитесь, что серверов MySQL хватает для существующего числа подключений, и распределите данные для повышения доступности, например, при помощи репликации;
- рассмотрите возможности повышения производительности MySQL Cluster, например, добавление дополнительных узлов данных.

Могут потребоваться некоторые компромиссы между желаемыми уровнями доступности и производительности. Например, добавление реплик повышает доступность. Но чем больше реплик, тем больше требуется вычислительной мощности и тем ниже производительность во время обновле-

ний. Операции чтения выполняются быстро, так как для получения данных не требуется считывать все реплики. Производительность будет выше, если число узлов будет большим, а число реплик маленьким.

Еще один важный аспект — распределенная природа MySQL Cluster. Наилучшая производительность достигается тогда, когда каждый узел работает на отдельном сервере, поэтому критически важна производительность каждого сервера. Важна также производительность сетевых компонентов. Так как координирующие команды и данные передаются от узла к узлу, производительность сетевого подключения должна быть высокой. Также следует учитывать такие аспекты, как транспортный протокол (например, TCP/IP, SHM и SCI), задержка, пропускная способность и географическая удаленность.

MySQL Cluster можно установить и запустить в облаке. Одно из преимуществ этого — очень высокая скорость сетевого взаимодействия. Так как для узлов данных, в основном, требуется быстрый процессор, достаточный объем памяти и быстрая сеть, технологии виртуальных серверов более чем достаточны для использования MySQL Cluster в облаке.

Подробнее об обеспечении высокой производительности см. в разделе «MySQL Cluster» руководства MySQL Reference Manual.

Рекомендации по обеспечению высокой производительности

Для обеспечения максимальной производительности MySQL Cluster можно предпринимать разные действия. Здесь мы приведем несколько наиболее полезных рекомендаций с кратким объяснением. Некоторые из них довольно общие, но их не следует пропускать, когда речь идет о достижении максимально возможной производительности.

Настраивайте шаблоны доступа. Изучите методы, используемые приложениями для доступа к данным. Так как MySQL Cluster хранит индексированные столбцы в памяти, доступ к этим столбцам выполняется даже быстрее, чем к неиндексированным столбцам на одиночном сервере MySQL. MySQL Cluster требует наличия первичного ключа в каждой таблице, так что приложения, обращающиеся к данным по первичному ключу, почти гарантированно будут работать быстро.

Учитывайте возможность распределения данных в приложениях. Лучший вариант доступа к данным из хранилища, использующего разделы, — изолировать запрос до отдельного узла кластера. По умолчанию MySQL Cluster использует первичный ключ для хеширования строк в разделах. К сожалению, это не всегда оптимально по отношению к поведению запросов с использованием главных и дополнительных таблиц (часто приложения обращаются к главной таблице, а затем к другим таблицам за дополнительной информацией, которая связана с данными из главной таблицы). В таком

случае следует изменить функцию хеширования, чтобы главная и дополнительные строки находились в одном узле. Один из способов сделать это — *сокращение раздела*, когда удаляется вторичное поле, используемое в хеше раздела дополнительной таблицы, а дополнительные строки разделяются только при помощи первичного ключа главной таблицы (который является внешним ключом в дополнительной таблице). Это позволяет разместить главную и дополнительные строки в одном узле дерева разделов.

Используйте пакетные операции. Каждая пересылка данных на сервер и возврат результатов при исполнении запроса сильно увеличивает нагрузку на сервер. В некоторых операциях, таких как вставка, можно избежать лишней нагрузки, используя запросы с одновременной вставкой нескольких строк (команды INSERT, вставляющие несколько строк). Можно также объединять операции, включая параметр transaction_allow_batching и добавляя разные операцию в одну транзакцию (между блоками BEGIN и END). Это позволяет указывать несколько запросов, обрабатывающих данные (INSERT, UPDATE и т. д.), и снижать нагрузку.

Параметр transaction_allow_batching не работает с командами SELECT и командами UPDATE, включающими переменные.

Оптимизируйте схемы. Оптимизация схем баз данных имеет в MySQL Cluster тот же эффект, что и в обычных СУБД. В MySQL Cluster используйте эффективные типы данных (например, выбирайте минимальный размер для экономии памяти; 30 байт на строку в таблице с миллионом строк позволят сэкономить значительный объем памяти). Также следует рассмотреть возможность денормализации в некоторых схемах для использования преимуществ методов параллельного доступа (разделения).

Оптимизируйте запросы. Понятно, что чем лучше оптимизирован запрос, тем быстрее он будет выполняться. Этот совет применим ко всем базам данных, и при повышении производительности приложения его следует выполнять одним из первых. Для MySQL Cluster рассматривайте оптимизацию запросов с точки зрения получения данных. В частности, объединения особенно чувствительны к производительности MySQL Cluster. Запросы с плохой производительностью иногда могут приводить к появлению аномалий, которые легко принять за неэффективную работу других частей системы.

Оптимизируйте параметры сервера. Оптимизируйте конфигурацию кластера, чтобы он работал максимально эффективно. Для этого может потребоваться потратить время, чтобы разобраться с множеством параметров конфигурации, а также подобрать подходящее оборудование. Для этой задачи нет волшебного общего рецепта: каждая установка становится все более уникальной с изменением дополнительных параметров. Выполняйте оптимизацию осторожно, изменяя один параметр за раз, и всегда сравнивайте результаты с известными базовыми показателями, чтобы проверить эффект изменения.

Используйте пулы подключений. По умолчанию узлы SQL используют только один поток для подключения к кластеру NDB. Если потоков несколько, узлы SQL могут выполнять несколько запросов одновременно. Чтобы использовать пулы подключений для узлов SQL, добавьте в файл конфигурации параметр NDB-cluster-connection-pool. Поместите его в раздел [mysqld] и установите значение больше 1 (например, 4). Поэкспериментируйте с этим параметром, так как установленное значение может быть слишком большим для приложения или оборудования.

Используйте многопоточные узлы данных. Если узел данных имеет многоядерный процессор или несколько процессоров, можно повысить производительность, запустив демон многопоточных узлов данных NDBmtd. Этот демон может использовать до восьми ядер или потоков. Использование нескольких потоков позволяет узлу данных выполнять параллельно много операций, таких как локальный обработчик запросов и процессы коммуникации, что позволяет получить еще более высокую производительность.

Используйте NDB API для пользовательских приложений. Тогда как сервер MySQL (узел SQL) предоставляет быстрый внешний интерфейс обработчика процессов, MySQL имеет встроенный механизм прямого доступа, написанный на C++, называемый NDB API. Для некоторых операций, таких как взаимодействие с MySQL Cluster по LDAP, это может быть единственным способом подключения приложения к кластеру MySQL (в данном случае просто к кластеру NDB). Если для приложения критически важна производительность, и вы имеете необходимые ресурсы для разработки собственного решения NDB API, вы можете получить значительное повышение производительности.

Используйте подходящее оборудование. Конечно, быстрое оборудование дает более высокую производительность (в общем случае). Однако следует рассматривать каждый аспект конфигурации кластера. Следует учитывать не только быстрые процессоры и модули памяти, но и высокоскоростные решения для внутренних соединений, такие как SCI, а также быстрые избыточные сетевые подключения. Во многих случаях такие аппаратные решения продаются готовыми и не требуют перенастройки сервера.

Отключайте кэш запросов. Так как MySQL Cluster не использует механизм хранилища MyISAM, кэш запросов не даст никаких преимуществ, и его можно отключить.

Не используйте виртуальную память. Убедитесь, что узлы данных используют физическую память, а не файл подкачки. В противном случае производительность значительно упадет. К тому же, это не только снижает производительность, но и может угрожать стабильности кластера.

Используйте привязку к процессорам для узлов данных. В многопроцессорных системах привязывайте процессы узла данных к процессорам, не участвующим в сетевых взаимодействиях. На некоторых платформах (например, в системах Sun CMT) это можно сделать при помощи параметров

LockExecuteThreadToCPU и LockMaintThreadsToCPU из раздела [NDBd] файла конфигурации.

Если следовать этим рекомендациям, можно сделать MySQL Cluster лучшим высокопроизводительным решением с высокой доступностью. Подробнее об оптимизации MySQL Cluster см. в документе «Optimizing Performance of the MySQL Cluster Database» по адресу *http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_performance.php*.

Заключение

В этой главе мы обсудили уникальное решение для обеспечения высокой доступности MySQL Cluster. Среди преимуществ MySQL Cluster можно выделить секционирование таблиц и распределение их по нескольким узлам, а также параллельную архитектуру MySQL Cluster, поддерживающую базы данных со многими главными узлами. Это позволяет системе выполнять много операций чтения и записи одновременно. Все обновления мгновенно становятся доступными всем узлам приложений (посредством команд SQL или NDB API), обращающимся к данным, хранящимся на узлах данных.

Так как операции записи распределяются по всем узлам данных, можно достигать очень высокого уровня производительности записи и масштабируемости для транзакционных задач. Наконец, создание нескольких узлов — серверов MySQL (узлов SQL), работающих параллельно, где каждый сервер обслуживает несколько подключений, и использование репликации MySQL для передачи данных в географически удаленные системы позволяет создавать очень эффективные приложения, использующие транзакции и параллельные вычисления.

Хотя такие строгие требования предъявляются нечасто, MySQL Cluster — отличное решение для приложений, которым необходима наивысшая доступность MySQL.

— Джоэл!

Джоэл улыбнулся, когда его руководитель чуть не прошел мимо, но все-таки вернулся.

— Да, Боб?

Саммерсон вошел в кабинет, закрыл дверь, пододвинул стул и сел напротив. Растерявшись, Джоэл только улыбнулся и спросил:

— Чем могу быть полезен, Боб?

— Ты уже принес пользу, Джоэл. Ты очень быстро разбирался со всеми этими задачами с MySQL, не сбавляя темпа при расширении системы. А теперь ты помог нам заработать кучу денег на последней сделке. Я знаю, что много требовал от тебя, и ты заслужил что-нибудь взамен.

После неловкой паузы, он добавил:

— Джоэл, ты играешь в гольф?

Джоэл пожал плечами:

— Не играл со времен колледжа, да и тогда звезд с неба не хватал...

— Это не проблема. Я люблю эту игру, но любовь не взаимна: каждый раз теряю полкоробки мячей. Как насчет того, чтобы в субботу сыграть на девять лунок?

Джоэл не понимал, к чему клонит босс, но что-то подсказывало ему, что надо соглашаться.

— Конечно, я готов.

— Тогда встретимся в десять. Сыграем, а потом обсудим твое будущее за обедом.

— О'кей, увидимся, Боб.

Г-н Саммерсон встал, открыл дверь, но задержался.

— Я сказал в бухгалтерии, чтобы тебе выделили бюджет, в том числе на подписку MySQL Enterprise и двух помощников.

— Спасибо, — ответил пораженный Джоэл. Он не был готов к столь быстрому принятию его предложения, не говоря уже о внезапно свалившейся на него ответственности.

Когда Саммерсон исчез в коридоре, к Джоэлу зашла Эми.

— Ты в порядке? — спросила она обеспокоенно.

— Да, а что?

— Я никогда не видела, чтобы он закрывал дверь для разговора с кем-то. Может, расскажешь, что тут было?

Махнув рукой над бумагами, разложенными на столе, Джоэл ответил:

— Он предложил сыграть в гольф и сказал, что выделил денег на покупку MySQL Enterprise.

Эми улыбнулась и тронула его руку.

— Это хорошо, Джоэл, просто отлично.

Джоэл смутился. Вряд ли расширение бюджета или покупка рабочей программы заслуживали такой реакции.

— Почему ты так думаешь?..

— Последний, кто играл в гольф с Саммерсоном, получил повышение. Саммерсон может казаться строгим, но он всегда награждает за преданность делу и целеустремленность.

— Правда?

Джоэл осмотрел бумаги на столе и сказал себе, что не стоит радоваться раньше времени.

— Может, пообедаем? — спросила Эми, чуть сжав его руку.

Джоэл посмотрел на руку, лежащую на его руке, и улыбнулся.

— Конечно. Давай сходим в какое-нибудь уютное местечко.

Соглашаясь, Джоэл знал, что сегодня он будет долго планировать их следующее свидание, прежде чем уснуть.

ПРИЛОЖЕНИЕ

Репликация: секреты и советы

Ниже даются полезные советы и рекомендации по запуску, диагностике, восстановлению и оптимизации репликации MySQL.

Это дополнение к основному тексту книги, здесь нет всех подробностей, необходимых для полного руководства. Их вы найдете в онлайновой документации по ссылке *http://dev.mysql.com/doc/refman/5.4/en/index.html*.

Свежую информацию о передовых методах репликации можно найти по адресу *http://dev.mysql.com/replication*.

Последние два раздела этого приложения посвящены функциям, которые скоро появятся в MySQL, но на момент написания книги еще не доступны.

Что делать, если подчиненный сервер остановился?

Если подчиненный сервер останавливается без предупреждения, и есть сообщение об ошибке, поищите в документации возможные причины этой ошибки и выполните необходимые действия по восстановлению.

Исправив ошибку, выполните следующее, чтобы определить, с какого момента перезапускать подчиненный сервер после события, приведшего к ошибке:

1. Проверьте позицию, на которой произошла остановка, выполнив следующую команду:

   ```
   SHOW SLAVE STATUS
   ```

2. Используйте переменные Master_Log_File и Read_Master_Log_Pos для определения следующего события для передачи с главного сервера.
3. Используйте переменные Relay_Master_Log_File и Exec_Master_Log_Pos для определения следующего события, которое нужно применить к журналу главного сервера.
4. Используйте переменные Relay_Log_File и Relay_Log_Pos для определения следующего события, которое нужно применить к журналу ретрансляций.
5. Используйте mysqlbinlog для чтения содержимого:

```
mysqlbinlog master-log.000001
mysqlbinlog relay-log.000001
```

6. Изучите проблему и, если требуется, удалите строки из базы данных:

```
SET SQL_SLAVE_SKIP_COUNTER=1; START SLAVE
```

Проверка двоичного журнала с параметром verbose

Если ведется построчный журнал, для его анализа можно применять параметр --verbose, чтобы восстановить запросы в событиях журнала. Ниже показан примерный вывод для двоичного журнала:

```
$ mysqlbinlog --verbose master-bin.000001 BINLOG '
qZnvSRMBAAAAKQAAAAYCAAAAABAAAAAAAAAAAABHRlc3QAAnQxAAEDAAE= qZnvSRcBAAAAJwAAAC
OCAAAQABAAAAAAAAEAAf/+AwAAAP4EAAAA '/*!*/;
### INSERT INTO test.t1
### SET
### @1=3
### INSERT INTO test.t1
### SET
### @1=4
```

Обратите внимание на то, что значения столбцам присваиваются с использованием имен типа `@n`. Причина этого в том, что при репликации на основе строк для передачи и применения записей используются позиции столбцов, но игнорируются их имена.

Использование репликации для повторного заполнения таблицы

Если таблица на подчиненном сервере повреждена в результате ошибки или по случайности (например, пользователь удалил данные), данные можно восстановить при помощи репликации. Для этого создайте на главном сервере таблицу, являющуюся копией исходной таблицы, удалите исходную таблицу, а затем воссоздайте ее из копии. Такой способ хорошо работает, если столбцы не имеют таких типов данных, как autoincrement. В процессе репликации таблица на починенном сервере будет восстановлена. Существует два вида этой процедуры, зависящих от того, какой журнал ведется.

Логический журнал

Если в журнале регистрируются команды, выполите для каждой таблицы следующие команды:

```
SELECT * INTO OUTFILE 't1.txt' FROM t1;
DROP TABLE IF EXISTS t1;
```

```
CREATE TABLE t1 ...;
LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

Построчный журнал

Если в журнале регистрируются модификации строк, временная таблица не будет передана на подчиненный сервер. Поэтому отправляйте только данные, необходимые для инициализации таблицы, используя команду INTO:

```
CREATE TEMPORARY TABLE t1_tmp LIKE t1;
INSERT INTO t1_tmp SELECT * FROM t1;
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 SELECT * FROM t1_tmp;
```

MySQL Proxy и репликация с несколькими главными серверами

Ни один подчиненный сервер не может иметь более одного главного сервера, но у главного сервера может быть несколько подчиненных. Обычно это не проблема, но что, если требуется объединить данные с двух главных серверов и реплицировать результаты на подчиненные серверы?

Один из способов — использование MySQL Proxy в качестве промежуточной подчиненной системы. На рис. А-1 показана концептуальная схема такой конфигурации.

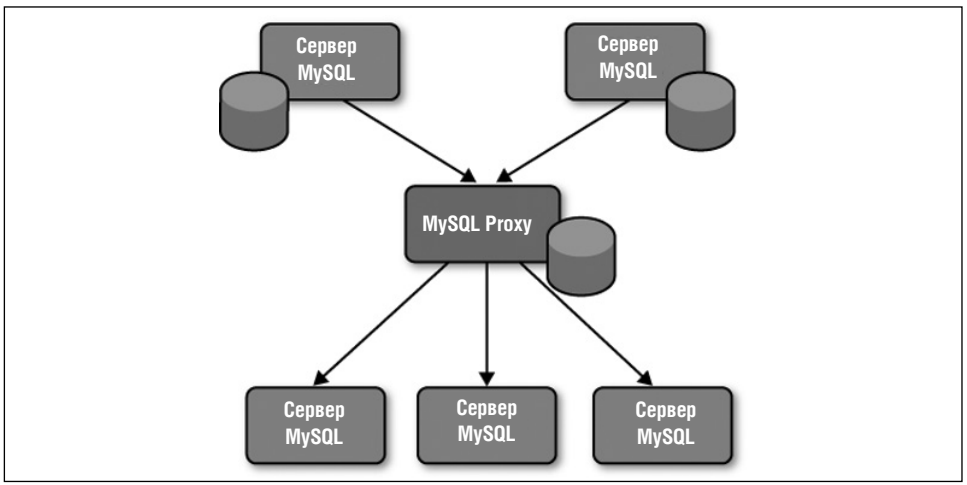


Рис. А-1. Использование MySQL Proxy для обращения к нескольким главным серверам

MySQL Proxy получит изменения (события) с обоих главных серверов и запишет новый двоичный журнал, не сохраняя данные (не записывая их в базу данных). Подчиненные серверы будут использовать MySQL Proxy как главный сервер. В результате объединенные данные с двух главных серверов

будут реплицированы на набор подчиненных серверов. По сути, это репликация с несколькими главными серверами.

Подробнее о MySQL Proxy см. в соответствующем разделе документации MySQL Reference Manual по адресу *http://dev.mysql.com/doc/refman/5.4/en*.

Механизм БД по умолчанию

Если задать на главном сервере механизм БД по умолчанию командой SET GLOBAL STORAGE ENGINE, эта команда не будет реплицирована. Следовательно, команды CREATE, в которых явно не указан механизм БД, будут использовать заданный на подчиненных серверах механизм БД по умолчанию. Так таблицы InnoDB могут быть реплицированы в таблицы MyISAM. Если проигнорировать предупреждения, появляющиеся при этом, репликация может остановиться из-за проблем с совместимостью.

Одно из решений — убедиться, что глобальный механизм БД установлен на каждом подчиненном сервере, включив в файл конфигурации параметр default-storage-engine. Однако рекомендуется указывать механизм и в команде CREATE.

Это не исключает проблем, связанных с отсутствием на подчиненном сервере тем, механизма БД, используемого на главном сервере. В таком случае единственное решение — установить недостающий механизм на подчиненном сервере.

Репликация MySQL Cluster из нескольких источников

В MySQL Cluster можно настроить репликацию из нескольких источников (рис. А-2). Для этого настройте главные серверы, реплицирующие разные данные. Хранение данных на разных главных серверах позволяет избежать дублирования ключей и других значений, зависящих от контекста.

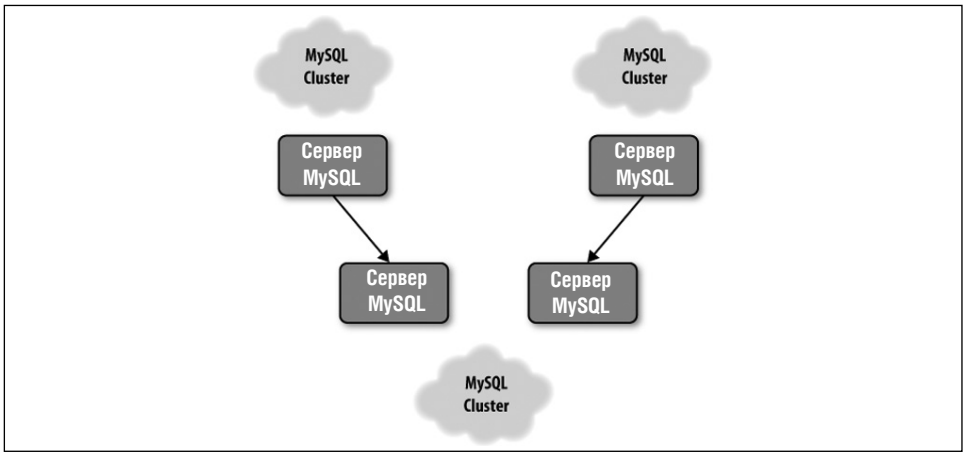


Рис. А-2. Репликация из нескольких источников

Многоканальная репликация с обработкой отказов

Для повышения надежности и скорости восстановления можно использовать двойную конфигурацию репликации (рис. А-3), в которой репликация выполняется между двумя наборами топологий в двух кластерах. Если один поток выйдет из строя, можно быстро переключиться на второй.

Обратите внимание на то, что имена серверов указываются в формате *mysql.X*, а сохраненные позиции — в формате *mysql.X.savepos*.

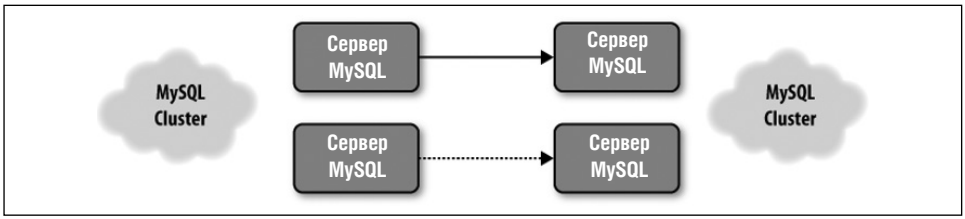


Рис. А-3. Многоканальная репликация

Фильтрация в текущей БД

Фильтры двоичного журнала могут быть весьма удобны для удаления определенных команд.

Некоторые команды имеют смысл только на сервере, Листинг — выбор механизма БД для таблицы, если механизмы на главном и подчиненных серверах должны различаться. Существует много причин для использования на подчиненных серверах другого механизма БД, включая:

- MyISAM больше подходит для создания отчетов; двоичный журнал содержит транзакции целиком, поэтому можно использовать InnoDB на главном сервере для обработки транзакций и InnoDB на подчиненных серверах для анализа и создания отчетов;
- для экономии памяти можно удалить на подчиненном сервере некоторые таблицы, используемые для аналитической обработки, заменив механизм БД на Blackhole.

Можно приостановить запись в двоичный журнал, задав переменной сервера SQL_LOG_BIN значение 0. Например, в примере А-1 журналирование отключается (SQL_LOG_BIN = 0) перед сменой механизма БД на MyISAM, чтобы оно выполнялось только на главном сервере, а затем снова включается (SQL_LOG_BIN = 1). Однако для изменения значения SQL_LOG_BIN требуется привилегия SUPER, которую нежелательно предоставлять обычным пользователям БД. Поэтому на листинге 5-4 показан альтернативный способ замены механизма хранилища на главном сервере без записи этой операции в журнал. (В этом примере предполагается, что таблица my_table находится в базе данных my_db.) Любой пользователь, желающий выполнить команду,

которая не должна реплицироваться, может использовать (USE) базу данных no_write_db, и команда будет отфильтрована. Так как для использования базы данных требуются привилегии доступа, можно определить, кому разрешено скрывать команды, не предоставляя привилегию SUPER.

Листинг А-1. Два способа фильтрации команды из двоичного журнала

```
master> SET SQL_LOG_BIN = 0;
Query OK, 0 rows affected (0.00 sec)

master> ALTER TABLE my_table ENGINE=MyISAM;
Query OK, 92 row affected (0.90 sec)
Records: 92 Duplicates: 0 Warnings: 0

master> SET SQL_LOG_BIN = 1;
Query OK, 0 rows affected (0.00 sec)

master> USE no_write_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
master> ALTER TABLE my_db.my_table ENGINE=MyISAM;
Query OK, 92 row affected (0.90 sec)
Records: 92 Duplicates: 0 Warnings: 0
```

На подчиненном сервере больше полей, чем на главном

Когда требуется добавить на подчиненном сервере поля, которых нет на главном сервере (например, чтобы записать временны*е* отметки или добавить локализованные данные), можно просто добавить эти поля в таблицу на подчиненном сервере, не добавляя на главном. Репликация MySQL поддерживает такой сценарий, игнорируя дополнительные поля. Чтобы вставить данные в дополнительные поля на подчиненном сервере, определите их так, чтобы они принимали значения по умолчанию (например, так легко вставлять временные отметки), или используйте для вставки значений триггер, определенный на подчиненном сервере.

Когда ведется журнал команд, столбцы можно создать следующим образом:

1. Создайте таблицу на главном сервере:

   ```
   CREATE TABLE t1 (a INT, b INT);
   ```

2. Измените таблицу на подчиненном сервере:

   ```
   ALTER TABLE t1 ADD ts TIMESTAMP;
   ```

3. Выполните вставку на главном сервере:

```
INSERT INTO t1(a,b) VALUES (10,20);
```

Если ведется построчный журнал, новые поля должны находиться в конце строки и иметь значения по умолчанию. Если добавить поля в середину или начало строки, репликация на основе строк не будет выполнена. Если поля добавлены в конец таблицы и имеют значения по умолчанию, нет особой разницы, какие команды используются при репликации, так как в процессе репликации на основе строк поля будут извлечены напрямую из обновляемой, вставляемой или удаляемой строки.

На подчиненном сервере меньше полей, чем на главном

Если на подчиненном сервере должно быть меньше полей, чем на главном (например, для защиты конфиденциальной информации или уменьшения объема реплицируемых данных), можно удалить поля из таблицы на подчиненном сервере, не удаляя на главном. Репликация MySQL на основе строк поддерживает такой сценарий, игнорируя отсутствующие поля. Однако отсутствующие на подчиненном сервере поля должны находиться в конце строки на главном сервере.

Для репликации на основе строк удалить поля можно следующим образом:

1. Создайте таблицу на главном сервере:

```
CREATE TABLE t1 (a INT, b INT, comments TEXT);
```

2. Измените таблицу на подчиненном сервере:

```
ALTER TABLE t1 DROP comments;
```

3. Выполните вставку на главном сервере:

```
INSERT INTO t1 VALUES (1,2,»Do not store this on slave»);
```

При построчной репликации можно реплицировать произвольный набор полей таблицы, независимо от того, находятся ли они в конце или нет. Для этого нужен триггер на главном сервере, обновляющий таблицу из базовой таблицы в отдельной БД, так вместо базовой таблицы реплицируется таблица из реплицируемой базы данных с меньшим числом полей. На рис. А-4 показана концептуальная схема этого решения. Таблица с тремя полями при помощи триггера преобразуется в таблицу, расположенную в реплицируемой базе данных.

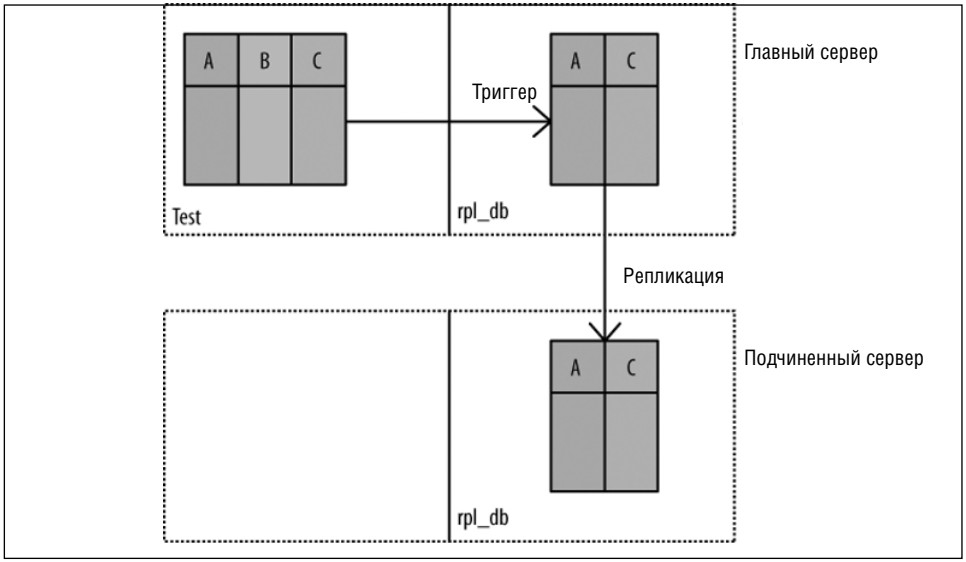


Рис. А-4. Репликация подмножества столбцов

Ниже показано, как использовать триггер для обновления таблицы в реплицируемой базе данных. На главном сервере выполните следующее:

```
CREATE DATABASE rpl_db;
USE test;
CREATE TABLE t1 (a INT, b BLOB, c INT);
CREATE TRIGGER tr_t1 AFTER INSERT ON test.t1 FOR EACH ROW
INSERT INTO rpl_db.t1_v(a,c) VALUES(NEW.a,NEW.c);
USE rpl_db;
CREATE TABLE t1_v (a INT, c INT);
```

Когда выполняется обычная вставка, триггер извлекает подмножество полей и записывает его в таблицу базы данных *rpl_db*. Эта база данных будет реплицирована, а исходная таблица нет.

```
USE test;
SET @blob = REPEAT('beef',100);
INSERT INTO t1 VALUES (1,@blob,3), (2,@blob,9);
```

Репликация выбранных строк на подчиненный сервер

Репликацию можно сегментировать, чтобы реплицировать только строки, удовлетворяющие определенным условиям. Для этого используется специальная база данных, реплицируемая с главного сервера на подчиненный (это делается с помощью фильтра), и триггер, который определяет реплицируемые строки, вставляя их в реплицируемую таблицу (рис. А-5).

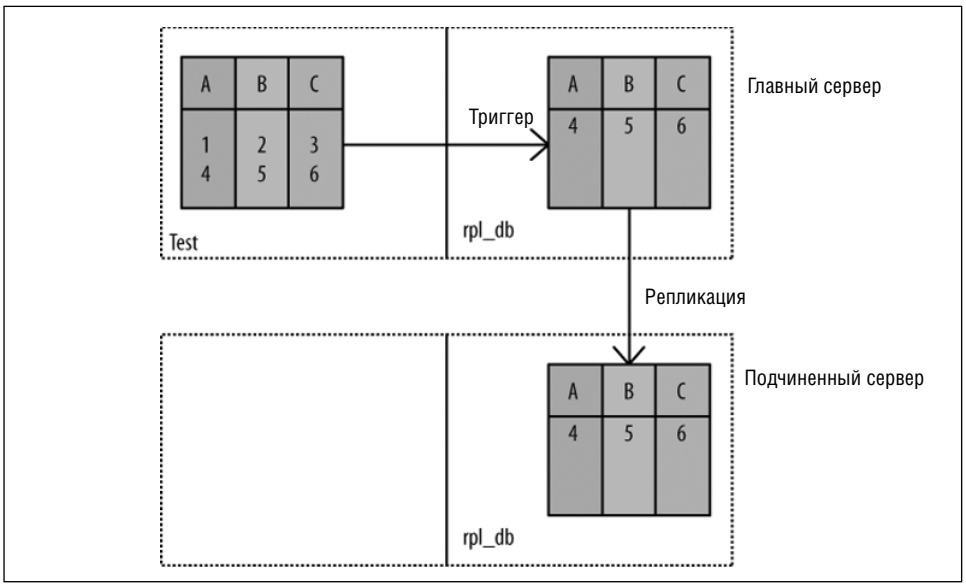


Рис. А-5. Репликация подмножества строк

Так можно использовать триггер для вставки в таблицу реплицируемой БД только строк с нечетным значением поля a:

```
# Реплицируем только строки, имеющие нечетные числа в первом стполе
USE rpl_db;
CREATE TABLE t1_h (a INT, b BLOB, c INT);
--delimiter //
CREATE TRIGGER slice_t1_horiz AFTER INSERT ON test.t1
FOR EACH ROW
BEGIN
  IF NEW.a MOD 2 = 1 THEN INSERT
    INTO rpl_db.t1_h VALUES (NEW.a, NEW.b, NEW.c);
  END IF;
END//
--delimiter ;
```

Heartbeat-сигналы репликации

Можно сделать репликацию более надежной и повысить отказоустойчивость, используя механизм heartbeat-импульсов. Когда он включен, подчиненный сервер периодически опрашивает главный. Если подчиненный сервер получает ответ, он продолжает функционировать как обычно и опрашивает главный сервер через заданный промежуток времени. Ведется статистика числа полученных импульсов, чтобы отслеживать доступность главного сервера. На рис. А-6 показана концептуальная схема механизма heartbeat-импульсов.

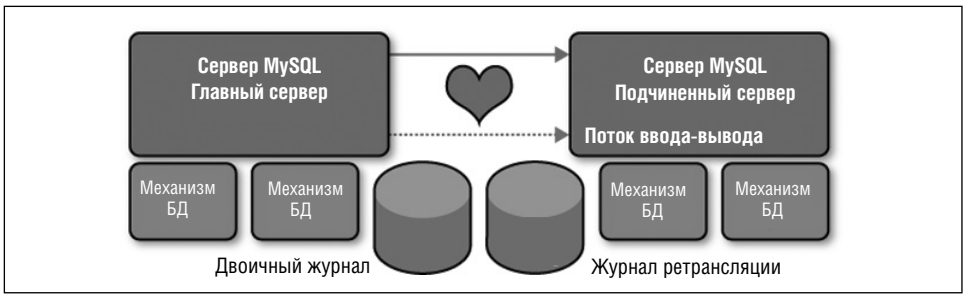


Рис. А-6. Heartbeat-импульсы во время репликации

Механизм heartbeat-импульсов позволяет подчиненному серверу определить, доступен ли главный сервер, что дает следующие преимущества и возможности:

- автоматическая проверка состояния подключения;
- журнал ретрансляции не меняется при недоступности главного сервера;
- определение момента отключения главного сервера с точностью до миллисекунд.

Таким образом, можно использовать механизм heartbeat-импульсов для автоматической обработки ошибок и аналогичных мер по обеспечению высокой доступности. Для установки интервала heartbeat-импульсов на подчиненном сервере выполните следующую команду:

```
CHANGE MASTER SET master_heartbeat_period= val;
```

Следующие команды позволяют проверить параметры и статистику механизма heartbeat-импульсов:

```
SHOW STATUS like 'slave_heartbeat period';
SHOW STATUS like 'slave_received_heartbeats';
```

Это относительно новая функция, появившаяся в MySQL 5.4.4.

Игнорирование серверов в круговой репликации

Если в топологии круговой репликации один из серверов выходит из строя, необходимо изменить топологию, исключив этот сервер. На рис. А-7 сервер А вышел из строя, и его необходимо удалить из круга. В этом случае можно настроить сервер В, чтобы он завершил события сервера А в новом круге.

Это можно сделать, выполнив на сервере В следующую команду:

```
CHANGE MASTER TO MASTER_HOST=C ... IGNORE_SERVER_IDS=(A)
```

Эта функция появилась в MySQL 5.5.

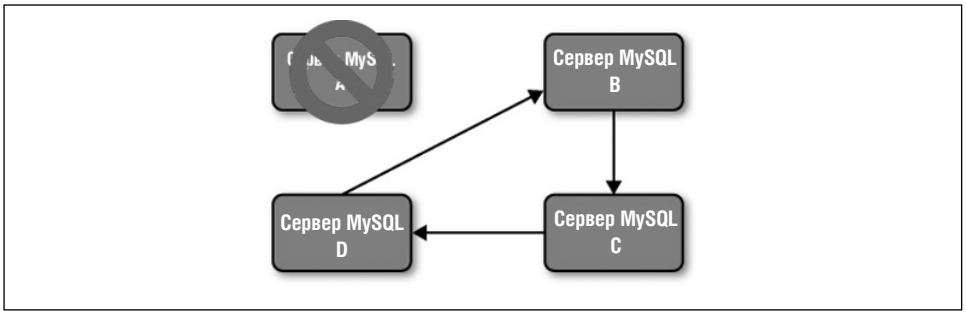


Рис. A-7. Замена сервера в круговой репликации

Анонс: репликация с задержкой

Иногда требуется задержать репликацию, например, чтобы не перегружать подчиненные серверы изменениями. Данная функция предоставляет возможность выполнять события репликации на подчиненном сервере так, что он всегда будет отставать от главного на *n* секунд. Этот параметр устанавливается следующим образом:

```
CHANGE MASTER TO MASTER_DELAY= n
```

где *n* — неотрицательное целое число секунд, меньшее MAX_ULONG (попытки установить большее значение завершатся ошибкой).

Чтобы проверить настройку задержки, выполните команду SHOW SLAVE STATUS и проверьте в выходных данных столбец Seconds_behind_master.

Подробнее о репликации с задержкой можно почитать по адресу *http://forge.mysql.com/wiki/ReplicationFeatures/DelayedReplication*.

Анонс: репликация с помощью сценариев

Когда требуется выполнить трансформацию, специфическую низкоуровневую фильтрацию или аналогичные операции в потоке репликации, пригодятся сценарии, которые могут выполняться:

- когда событие записывается в двоичный журнал на главном сервере;
- когда событие записывается в поток выгрузки на главном сервере;
- после чтения события потоком ввода-вывода, но перед его записью в журнал ретрансляции на подчиненном сервере;
- когда событие считывается из журнала ретрансляции на подчиненном сервере.

В зависимости от ваших потребностей, можете использовать одну или несколько из этих точек для выполнения операций с событиями. На рис. A-8 показаны положения точек, в которых возможно выполнение сценариев. Эта возможность позволяет выполнять сценарии на языке Lua.

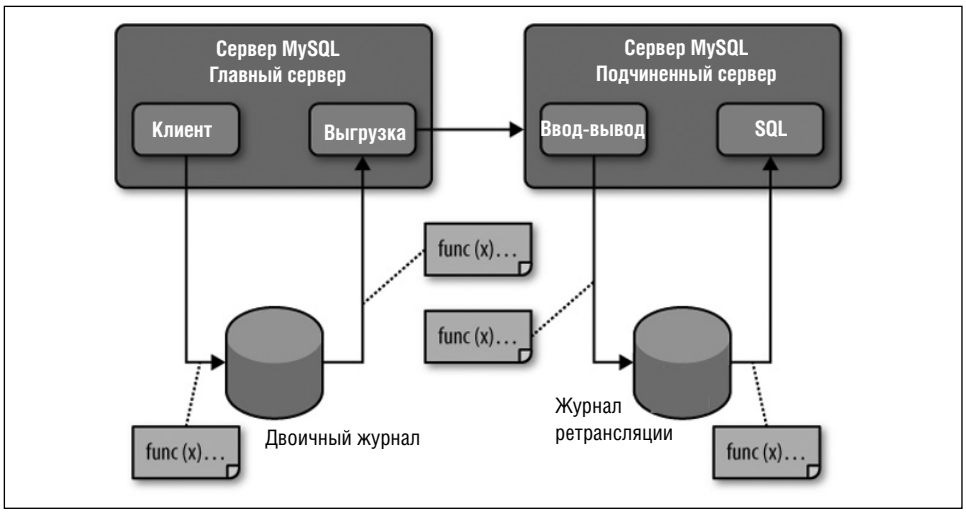


Рис. А-8. Точки для вставки сценариев

Подробнее о сценариях для выполнения репликации можно почитать по адресу *http://forge.mysql.com/wiki/Replica tionFeatures/ScriptableReplication*.

Анонс: алгоритм Oracle

Пол Такфилд (Google/YouTube) разработал алгоритм фильтрации событий, который можно использовать в решениях для репликации по сценарию (рис. А-9).

```
module(..., package.seeall); require "luasql.mysql"
pattern = {
        [ "UPDATE%s+(%w+) .*%s(WHERE.*)"] = "SELECT * FROM %1 %2",
        ["DELETE%s+FROM%s+(%w+) .*%s(WHERE.*)"] = "SELECT * FROM %1 %2",
}
env = luasql.mysql ()
con = env:connect("test", "root", "", "localhost", mysql.port)

function before_write(event)
        local line = event.query
        if not line then return end
        for pat,repl in pairs(pattern) do
                local str = string.gsub(line, pat, repl)
                if str then con:execute(str) ; break; end
        end
end
```

Рис. А-9. Алгоритм Oracle Пола Такфилда

Интересно, что название этого алгоритма (oracle — *англ.* оракул) отражает «предвидение» событий и заблаговременной подготовки кэша. Оно не имеет никакого отношения к компании Oracle или алгоритму обработки текста Oracle.

> Если вы работаете с Maatkit, можете использовать mk-slave-prefetch для получения таких же возможностей при помощи команд SQL. Подробнее см. по адресу *http://maatkit.org/doc/mk-slave-prefetch.html*.

Предметный указатель

Символы

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

А

Б

В

Г

Д

Ж

З

И

К

M

Н

О

П

P

C

T

У

Ф

Х

Ц

Ш

Э

Об авторах

Доктор Чарльз Белл — главный специалист по разработке ПО в компании Oracle. В настоящий момент он является ведущим разработчиком систем резервного копирования и членом команды MySQL Backup and Replication (Резервное копирование и репликация MySQL). Они с женой живут в небольшом городке в сельской местности штата Вирджиния. Чарльз получил степень доктора философии в области инженерии в Университете Содружества Вирджинии в 2005 году. Его научные интересы включают теорию СУБД, системы управления версиями, семантические веб-технологии и динамическую разработку.

Доктор Мэтс Киндал — главный разработчик ПО, работающий над MySQL Server. Он является основным архитектором и разработчиком функций построчной репликации MySQL и отвечает за стратегию разработки репликации, ре-инжиниринга и подключаемой архитектуры. Перед началом работы над MySQL он проводил исследования в области формальных методов, анализа кода и распределенных систем, за что получил докторскую степень. Также он провел много лет за разработкой компиляторов C/C++. Чтобы пересчитать языки программирования, которыми он владеет, не хватит пальцев.

Доктор Ларс Талманн — руководитель разработки средств резервного копирования и репликации MySQL. Он отвечает за стратегию и разработку этих функций и возглавляет соответствующие команды разработчиков. Талманн связан с разработкой MySQL с 2001 года, когда он занимался разработкой MySQL Cluster. Затем он работал над MySQL Enterprise Backup, задавал ход «эволюции» функций репликации MySQL с 2004 года, и был одним их ключевых разработчиков репликации MySQL Cluster. Талманн получил в шведском университете Упсала докторскую степень по информатике.

Эмблема книги

Животное на обложке — североамериканский странствующий дрозд (Turdus migratorius). Этот представитель семейства дроздов — одна из наиболее распространенных в Америке птиц. Хотя он называется так же, как его европейский собрат, который тоже имеет красную грудь, эти два вида не являются близкородственными.

Площадь ареала странствующего дрозда — шесть миллионов квадратных миль в Северной Америке, он встречается круглый год в большинстве штатов. Дроздов считают вестниками весны, они начинают петь рано утром и последними замолкают ближе к ночи. Питаются они беспозвоночными (чаще всего дождевыми червями), фруктами и ягодами. Дрозды предпочитают открытые пространства и низкую траву, поэтому их легко встретить на задних дворах, в парках, садах и на газонах.

O'REILLY®

Обеспечение высокой доступности систем на основе MySQL

В реальной жизни вы непременно столкнетесь с отказами серверов БД и «узкими местами», но можно сделать так, чтобы эти досадные неприятности не приводили к полной остановке ИТ-системы. В MySQL есть функции, которые помогут вам защитить вверенные вам системы от простоев независимо от того, работают ли они на физических серверах, в виртуальной среде или в облаке. В этой книге рассматривается использование функций репликации, поддержки кластеров и мониторинга в самых разных жизненных ситуациях. Авторы сами являются разработчиками множества описанных в ней функций MySQL, поэтому они как никто другой могут объяснить читателям недокументированные и неочевидные возможности, обеспечивающие надежность и высокую доступность БД на основе MySQL. Книга дает знания, совершенно необходимые для работы в любых организациях, использующих эту СУБД.

Прочитав эту книгу, вы научитесь:

- анализировать двоичный журнал для настройки репликации, восстановления после аварий и устранения неполадок;
- сокращать время отклика системы при обработке больших объемов данных;
- клонировать серверы с использованием репликации;
- вести мониторинг операций с БД, производительности системы и ключевых параметров ОС;
- отслеживать действия главных и подчиненных серверов, бороться с отказами, а также с последствиями перезагрузки, повреждения данных и прочих происшествий;
- автоматизировать ключевые операции с помощью библиотеки, написанной авторами книги;
- использовать MySQL в виртуализированных средах, таких как Amazon Web Services.

«Репликацию MySQL часто используют на практике, но никогда не разбирают толком в теории. Эта книга наверстывает упущенное».

Марк Каллаган (Mark Callaghan), один из разработчиков MySQL, глава инженерных групп по MySQL в нескольких крупнейших интернет-компаниях

ISBN 978-5-7502-0409-0

9 785750 204090

Издательство
Русская редакция
Москва, 123298
ул. 3-я Хорошевская, 11
E-mail: info@rusedit.com
Internet: www.rusedit.com
Тел./факс (495) 638-5-638

O'REILLY®
www.oreilly.com