8

Репликация

Встроенные в MySQL средства репликации составляют основу для построения крупных высокопроизводительных приложений. Они позволяют сконфигурировать один или несколько серверов в качестве подчиненных другому серверу; такие серверы называют репликами. Это полезно не только при создании высокопроизводительных приложений, но и во многих других случаях, например для совместного использования данных с удаленным офисом, для поддержания «горячей замены» или для хранения копии актуальных данных на другом сервере с целью тестирования или обучения.

В этой главе мы рассмотрим все аспекты репликации. Начав с обзора принципов работы, мы затем перейдем к простейшей настройке сервера и далее продемонстрируем более сложные конфигурации и расскажем о том, как управлять реплицированными серверами и оптимизировать их. Хотя эта книга посвящена, прежде всего, вопросам производительности, в деле репликации не менее важны корректность и надежность, поэтому мы остановимся и на том, как организовать правильную работу репликации. Мы обсудим также планируемые изменения и улучшения в механизме репликации MySQL, например любопытные заплаты, созданные в компании Google.

Обзор репликации

Основная задача, которую призвана решить репликация, – это синхронизация данных одного сервера с данными другого. К одному главному серверу можно подключить несколько подчиненных (slave), причем подчиненный сервер может, в свою очередь, выступать в роли главного. Топология сети главных и подчиненных серверов зачастую сильно различается. Можно реплицировать сервер целиком, или только некоторые базы данных, или даже определенные таблицы.

MySQL поддерживает две разновидности репликации: покомандную и построчную. Покомандная (или «логическая») репликация существует еще со времен версии 3.23, в настоящее время именно она обычно используется в промышленной эксплуатации. Построчная репликация появилась в версии MySQL 5.1. В обоих случаях изменения записываются в двоичный журнал[1] на главном сервере и воспроизводятся на подчиненном, причем оба варианта *асинхронны*, то есть не гарантируется, что копия данных на подчиненном сервере хотя бы в какой-то момент полностью актуальна[2]. Относительно величины отставания также не дается никаких гарантий. Если запросы сложны, то подчиненный сервер может отставать от главного на секунды, минуты и даже часы.

Репликация в MySQL в основном обратно совместима. Это означает, что сервер более поздней версии может быть подчинен серверу, на котором установлена ранняя версия MySQL. Однако старые версии обычно не могут выступать в роли подчиненных для более свежих; они не распознают новые синтаксические конструкции SQL, да и форматы файлов репликации могут отличаться. Например, невозможно реплицировать главный сервер версии MySQL 5.0 на подчиненный версии 4.0. Мы рекомендуем протестировать схему репликации до перехода на новую версию, в которую были внесены серьезные изменения, например при переходе с 4.1 на 5.0 или с 5.0 на 5.1.

Вообще говоря, накладные расходы репликации на главном сервере невелики. Правда, на нем требуется включить двоичный журнал, что само по себе весьма ощутимо, но это так или иначе нужно сделать, если вы хотите снимать нормальные резервные копии. Помимо записи в двоичный журнал небольшую нагрузку (в основном, на сеть) дает добавление каждого подчиненного сервера.

Репликация вполне применима для масштабирования операций чтения, которые можно адресовать подчиненному серверу, но для масштабирования записи она не очень подходит, если только не учесть это требование при проектировании системы. Подключение большого количества подчиненных серверов просто приводит к тому, что запись выполняется многократно, по одному разу на каждом из них. Система в целом ограничена количеством операций записи, которые может выполнить самое слабое ее звено.

При наличии нескольких подчиненных серверов репликация становится расточительным удовольствием, так как данные без нужды дублируются несколько раз. Например, если к одному главному серверу подключено 10 подчиненных, то на главном сервере оказывается 11 одинаковых копий данных, которые дублируются в 11 разных кэшах. Это можно считать аналогом RAID 1 с 11 дисками. Подобное использова-

---

[1] Информацию о двоичном журнале вы можете найти в главе 6, ниже в этой главе и в главе 11.

[2] Подробнее см. раздел «Синхронная репликация в **MySQL» на стр. 558.**

ние оборудования неэкономно, тем не менее такая конфигурация встречается на удивление часто. Ниже мы обсудим различные способы сгладить эту проблему.

Проблемы, решаемые репликацией

Перечислим несколько типичных случаев применения репликации.

Распространение данных

Обычно репликация в MySQL потребляет не очень большую часть пропускной способности сети[1], к тому же ее можно в любой момент остановить и затем возобновить. Это полезно, если хранение копии данных происходит в географически удаленном пункте, например в другом центре обработки данных. Удаленный подчиненный сервер может работать даже с непостоянным (намеренно или по другим причинам) соединением. Однако если вы хотите обеспечить минимальное отставание реплики, то следует использовать надежный канал с малым временем задержки.

Балансировка нагрузки

С помощью репликации можно распределить запросы на чтение между несколькими серверами MySQL; в приложениях с интенсивным чтением эта тактика работает очень хорошо. Реализовать несложное балансирование нагрузки можно, внеся совсем немного изменений в код. Для небольших приложений достаточно просто «зашить» в программу несколько доменных имен или воспользоваться циклическим (round-robin) разрешением DNS-имен (когда с одним доменным именем связано несколько IP-адресов). Возможны и более изощренные решения. Стандартные технологии балансирования нагрузки, в частности сетевые балансировщики, прекрасно послужат для распределения нагрузки между несколькими серверами MySQL. Неплохо зарекомендовал себя и проект Linux Virtual Server (LVS). Подробнее о балансировании нагрузки мы будем говорить в главе 9.

Резервное копирование

Репликация – это ценное подспорье для резервного копирования. Однако подчиненный сервер все же не может использоваться в качестве резервной копии и не является заменой настоящему резервному копированию.

Высокая доступность и аварийное переключение на резервный сервер (failover)

Репликация позволяет исправить ситуацию, при которой сервер MySQL является единственной точкой отказа приложения. Хорошая система аварийного переключения при отказе, имеющая в со-

[1] Впрочем, как мы увидим ниже, построчная репликация, появившаяся в версии MySQL 5.1, может порождать гораздо больший сетевой трафик, чем традиционная покомандная репликация.

ставе реплицированные подчиненные серверы, способна существенно сократить время простоя. Мы рассмотрим эту тему в главе 9.

Тестирование новых версий MySQL

Очень часто на подчиненный сервер устанавливают новую версию MySQL и перед тем как ставить ее на промышленные серверы, проверяют, что все запросы работают нормально.

Как работает репликация

Перед тем как вплотную заняться настройкой репликации, посмотрим, как же на самом деле MySQL реплицирует данные. На самом верхнем уровне репликацию можно описать в виде процедуры, состоящей из трех частей.

1. Главный сервер записывает изменения данных в двоичный журнал. Эти записи называются *событиями двоичного журнала*.
2. Подчиненный сервер копирует события двоичного журнала в свой журнал ретрансляции (relay log).
3. Подчиненный сервер воспроизводит события из журнала ретрансляции, применяя изменения к собственным данным.

Это лишь общая картина, в реальности каждый шаг весьма сложен. На рис. 8.1 схема процедуры репликации изображена более подробно.

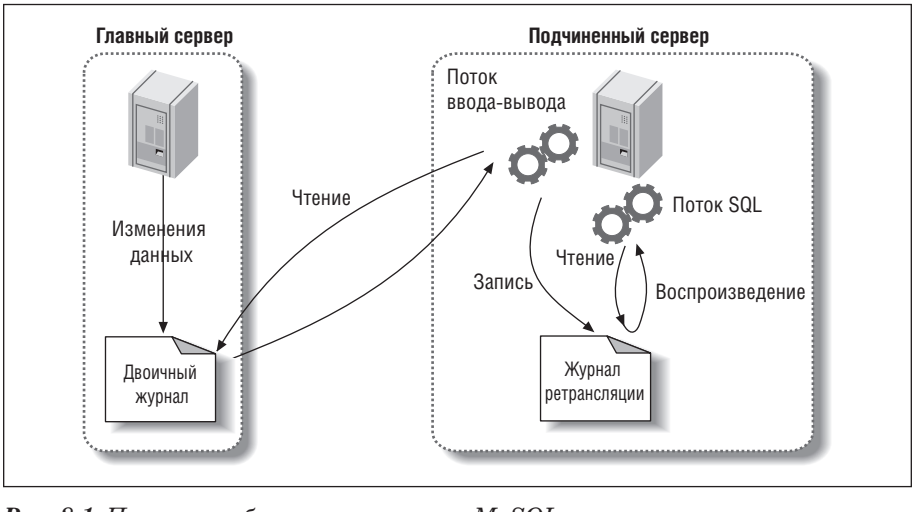
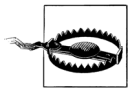


Рис. 8.1. *Принцип работы репликации в MySQL*

Первый этап данного процесса – запись в двоичный журнал на главном сервере (как ее настроить, мы объясним чуть позже). Непосредственно перед тем, как завершить транзакцию, обновляющую данные, главный сервер заносит изменения в свой двоичный журнал. MySQL записывает транзакции последовательно, даже если во время выполнения пере-

межаются команды из разных транзакций. Записав события в двоичный журнал, главный сервер просит подсистему хранения зафиксировать транзакцию.

На следующем этапе подчиненный сервер копирует двоичный журнал главного сервера на свой жесткий диск, в так называемый *журнал ретрансляции*. Первым делом он запускает *поток ввода/вывода*. Этот поток открывает обычное клиентское соединение с главным сервером, а затем запускает специальный процесс *дампа двоичного журнала* (*binlog dump*) (соответствующей команды в языке SQL не существует). Этот процесс читает события из двоичного журнала главного сервера. Он не опрашивает события активно. Обнаружив конец журнала, процесс выгрузки дампа засыпает и ждет, пока главный сервер не просигнализирует о появлении новых событий. Прочитанные события поток ввода/вывода записывает в журнал ретрансляции на подчиненном сервере.



До выхода версии MySQL 4.0 репликация во многих отношениях работала по-другому. Например, первоначально никакого журнала ретрансляции не было, поэтому для репликации использовалось два, а не три потока. Но сейчас, как правило, эксплуатируются более поздние версии сервера, поэтому рассказывать об уже неактуальных деталях мы не будем.

На последнем этапе в дело вступает *поток SQL*. Он читает и воспроизводит события из журнала ретрансляции, приводя данные на подчиненном сервере в соответствие с главным сервером. При условии, что поток SQL успевает за потоком ввода/вывода, журнал ретрансляции обычно остается в кэше операционной системы, так что накладные расходы на работу с этим журналом очень низкие. События, исполняемые потоком SQL, могут также записываться в собственный двоичный журнал подчиненного сервера, что бывает полезно в некоторых сценариях, которые мы рассмотрим ниже в этой главе.

На рис. 8.1 показано только два потока репликации на подчиненном сервере, но есть и еще один поток на главном сервере – тот, что ассоциирован с соединением, которое открыл подчиненный сервер.

Такая архитектура репликации позволяет развязать процессы выборки и воспроизведения событий на подчиненном сервере и сделать их асинхронными. Иными словами, поток ввода/вывода может работать независимо от потока SQL. Кроме того, она налагает определенные ограничения на процедуру репликации, из которых важнее всего то, что *репликация сериализуется на подчиненном сервере*. Это означает, что обновления, производившиеся на главном сервере, возможно, параллельно (в разных потоках), на подчиненном сервере распараллелены быть не могут. Как мы увидим ниже, при некоторых характеристиках рабочей нагрузки это способно стать узким местом.

Настройка репликации

В MySQL настройка репликации не вызывает особых сложностей, но у основных шагов есть много вариаций, зависящих от конкретного сценария. Самый простой случай – когда главный и подчиненный серверы только что установлены и еще не введены в эксплуатацию. На верхнем уровне процедура выглядит следующим образом:

1. Завести учетные записи репликации на каждом сервере.
2. Сконфигурировать главный и подчиненный сервера.
3. Сказать подчиненному серверу, чтобы он соединился с главным и начал реплицировать данные с него.

Здесь подразумевается, что многие принимаемые по умолчанию параметры удовлетворительны, и это действительно так, если главный и подчиненный серверы только что установлены и обладают в точности одними и теми же данными (стандартной базой `mysql`). Мы последовательно опишем действия на каждом шаге, предполагая, что серверы называются `server1` (IP-адрес 192.168.0.1) и `server2` (IP-адрес 192.168.0.2). Затем мы объясним, как инициализировать подчиненный сервер с помощью уже эксплуатируемого, и подробно рассмотрим рекомендуемую конфигурацию репликации.

Создание учетных записей репликации

В MySQL предусмотрено несколько специальных привилегий, необходимых для запуска репликации. Поток ввода/вывода, работающий на подчиненном сервере, устанавливает TCP/IP-соединение с главным сервером. Это означает, что на главном сервере должна существовать учетная запись, наделенная соответствующими привилегиями, чтобы поток ввода/вывода мог соединиться от ее имени и читать двоичный журнал главного сервера. Ниже показано, как создать такую учетную запись, – мы назвали ее *repl*:

```
mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.*
    -> TO repl@'192.168.0.%' IDENTIFIED BY 'p4ssword';
```

Такая запись создается как на главном, так и на подчиненном сервере. Отметим, что мы разрешили пользователю устанавливать соединение только из локальной сети, поскольку учетная запись репликации не защищена (дополнительную информацию о безопасности см. в главе 12).

Учетной записи репликации на самом деле нужна только привилегия `REPLICATION SLAVE` на главном сервере, а `REPLICATION CLIENT` не нужна ни на главном, ни на подчиненном сервере. Так зачем же мы их дали на обоих серверах? Тому есть две причины.

- Учетной записи, которая применяется для мониторинга и управления репликацией, нужна привилегия `REPLICATION CLIENT`, и луч-

ше не усложнять себе жизнь, а использовать одну и ту же запись для обеих целей.

- Если вы заведете учетную запись на главном сервере, а затем клонируете его для настройки подчиненного сервера, то подчиненный сервер уже будет подготовлен для роли главного на случай, если вы захотите поменять серверы ролями.

Конфигурирование главного и подчиненного серверов

Следующий шаг – настроить несколько параметров на главном сервере, в качестве которого у нас будет выступать `server1`. Необходимо включить двоичный журнал и задать идентификатор сервера. Введите (или убедитесь в наличии) такие строчки в файл *my.cnf* на главном сервере:

```
log_bin = mysql-bin
server_id = 10
```

Конкретные значения выберите по своему усмотрению. Мы пошли по простейшему пути, но вам, возможно, захочется сделать что-то похитрее.

Необходимо явно назначить серверу уникальный идентификатор. Мы задали 10, а не 1, так как значение 1 сервер обычно выбирает по умолчанию, если не задано никакое другое (это зависит от версии, некоторые версии MySQL в этом случае вообще не работают). Поэтому выбор 1 легко может привести к конфликтам между серверами, которым идентификатор явно не назначен. Часто в качестве идентификатора выбирают последний октет IP-адреса сервера, предполагая, что он уникален и в будущем не изменится (то есть все серверы находятся в одной подсети).

Если двоичный журнал ранее не был включен на главном сервере, то MySQL придется перезапустить. Чтобы проверить, создан ли на главном сервере двоичный журнал, выполните команду `SHOW MASTER STATUS` и сравните ее результат с приведенным ниже (MySQL добавляет к имени файла несколько цифр, поэтому истинное имя будет отличаться от заданного вами):

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |       98 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Файл *my.cnf* на подчиненном сервере выглядит примерно так же, как на главном, но с некоторыми дополнениями; подчиненный сервер также необходимо перезапустить:

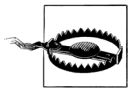
```
log_bin     = mysql-bin
server_id   = 2
```

```
relay_log         = mysql-relay-bin
log_slave_updates = 1
read_only         = 1
```

Некоторые из этих параметров, строго говоря, необязательны, а для других мы явно задали значения, совпадающие со значениями по умолчанию. На самом деле, на подчиненном сервере обязательным является только параметр `server_id`, но мы включили также `log_bin` и присвоили файлу журнала явное имя. По умолчанию имя этого файла совпадает с именем хоста, но при такой конфигурации могут возникнуть проблемы, если в будущем имя хоста изменится. Кроме того, мы хотим, чтобы журналы на обоих серверах назывались одинаково на случай, если возникнет желание превратить подчиненный сервер в главный. Исходя из этого мы не только завели на обоих серверах одноименные учетные записи, но и остальные параметры задали одинаково.

Еще мы добавили два необязательных параметра: `relay_log` (определяет имя и местоположение журнала ретрансляции) и `log_slave_updates` (чтобы подчиненный сервер записывал реплицированные события в собственный двоичный журнал). Последнее добавляет подчиненному серверу работы, но, как мы вскоре убедимся, имеются обоснованные причины для того, чтобы задавать эти параметры на всех подчиненных серверах.

Некоторые предпочитают включать только двоичный журнал, не задавая параметр `log_slave_updates`, с целью сразу же увидеть, изменяются ли какие-нибудь данные на подчиненном сервере (например, из-за неправильно сконфигурированного приложения). Если возможно, то лучше использовать параметр `read_only`, который не дает модифицировать данные никому, кроме потоков со специальными привилегиями (не выдавайте пользователям больше привилегий, чем реально необходимо для работы!) Однако часто параметр `read_only` оказывается непрактичным, особенно если некое приложение должно иметь возможность создавать таблицы на подчиненных серверах.



Не помещайте такие конфигурационные параметры, как `master_host` и `master_port`, в файл *my.cnf* на подчиненном сервере. Это устаревший способ конфигурирования подчиненного сервера. Он может привести к неприятностям и не дает никаких преимуществ.

Запуск подчиненного сервера

Следующий шаг – сообщить подчиненному серверу о том, как соединиться с главным и начать воспроизведение двоичных журналов. Для этой цели используется не файл *my.cnf*, а команда `CHANGE MASTER TO`. Она полностью заменяет соответствующие настройки в файле *my.cnf*. Кроме того, она позволяет впоследствии указать подчиненному серверу другой главный без перезапуска. Ниже приведена простейшая форма команды, необходимой для запуска репликации на подчиненном сервере:

```
mysql> CHANGE MASTER TO MASTER_HOST='server1',
    -> MASTER_USER='repl',
    -> MASTER_PASSWORD='p4ssword',
    -> MASTER_LOG_FILE='mysql-bin.000001',
    -> MASTER_LOG_POS=0;
```

Параметр MASTER_LOG_POS устанавливается в 0, потому что это начало журнала. После того как эта команда отработает, выполните команду SHOW SLAVE STATUS и проверьте, что параметры подчиненного сервера установлены правильно:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
       Slave_IO_State:
          Master_Host: server1
          Master_User: repl
          Master_Port: 3306
        Connect_Retry: 60
      Master_Log_File: mysql-bin.000001
  Read_Master_Log_Pos: 4
       Relay_Log_File: mysql-relay-bin.000001
        Relay_Log_Pos: 4
Relay_Master_Log_File: mysql-bin.000001
     Slave_IO_Running: No
    Slave_SQL_Running: No
          ...опущено...
Seconds_Behind_Master: NULL
```

Столбцы Slave_IO_State, Slave_IO_Running и Slave_SQL_Running показывают, что процессы репликации на подчиненном сервере не запущены. Внимательный читатель заметит также, что позиция указателя журнала равна 4, а не 0. Это объясняется тем, что 0 – это не столько истинное значение указателя, сколько признак «в начале файла журнала». MySQL знает, что данные первого события начинаются в позиции 4[1].

Чтобы запустить репликацию, выполните следующую команду:

```
mysql> START SLAVE;
```

Она не должна выводить никаких сообщений. Снова проверьте состояние подчиненного сервера:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
       Slave_IO_State: Waiting for master to send event
          Master_Host: server1
          Master_User: repl
```

[1] В действительности, из результата приведенной выше команды SHOW MASTER STATUS следует, что указатель находится в позиции 98. Подчиненный сервер разберется в этом вопросе, когда установит соединение с главным, чего пока не произошло.

```
          Master_Port: 3306
        Connect_Retry: 60
      Master_Log_File: mysql-bin.000001
  Read_Master_Log_Pos: 164
       Relay_Log_File: mysql-relay-bin.000001
        Relay_Log_Pos: 164
Relay_Master_Log_File: mysql-bin.000001
     Slave_IO_Running: Yes
    Slave_SQL_Running: Yes
       ...пропущено...
Seconds_Behind_Master: 0
```

Теперь на подчиненном сервере работают потоки ввода/вывода и SQL, а переменная состояния `Seconds_Behind_Master` отлична от `NULL` (что она означает, мы расскажем ниже). Поток ввода/вывода ожидает события от главного сервера, то есть в настоящий момент он прочитал из двоичного журнала все записи, которые там были. Позиции указателей в обоих журналах сместились, иными словами, какие-то события были прочитаны и обработаны (на вашем сервере картина может быть иной). Если сейчас произвести какое-нибудь изменение на главном сервере, то указатели позиций на подчиненном увеличатся. Кроме того, изменения будут применены к подчиненному серверу!

Потоки репликации должны быть видны в списках процессов на главном и подчиненном серверах. На главном вы увидите соединение, созданное потоком ввода/вывода, который работает на подчиненном сервере:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 55
   User: repl
   Host: slave1.webcluster_1:54813
     db: NULL
Command: Binlog Dump
   Time: 610237
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
```

На подчиненном сервере должно быть два потока: ввода/вывода и SQL:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 611116
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 2
   User: system user
```

```
     Host:
       db: NULL
  Command: Connect
     Time: 33
    State: Has read all relay log; waiting for the slave I/O thread to update it
     Info: NULL
```

Приведенные нами примеры распечаток получены на серверах, которые проработали достаточно долго, поэтому в столбце `Time` для потока ввода/вывода на главном и подчиненном серверах выводится большое значение. SQL-поток на подчиненном сервере простаивает 33 секунды, то есть в течение этого времени не было воспроизведено ни одного события.

Эти процессы всегда работают от имени учетной записи «system user», но значения в остальных столбцах могут отличаться от показанных. Например, когда поток SQL воспроизводит событие на подчиненном сервере, в столбце `Info` отображается исполняемый запрос.

Если вы хотите просто поэкспериментировать с репликацией, попробуйте сценарий MySQL Sandbox, написанный Джузеппе Максиа (Giuseppe Maxia) (*http://sourceforge.net/projects/mysql-sandbox/)*. Он позволяет быстро создать из дистрибутивного tgz-файла новую инсталляцию MySQL, которую потом можно будет безболезненно удалить. Чтобы получить работающие главный и два подчиненных сервера, достаточно нескольких нажатий клавиш и 15 секунд.

```
$ ./set_replication.pl ~/mysql-5.0.45-linux-x86_64-glibc23.tar.gz
```

Инициализация подчиненного сервера на основе существующего

Выше мы предполагали, что главный и подчиненный серверы только что установлены, поэтому данные на них практически одинаковы и позиция указателя в файле двоичного журнала известна. Но на практике так обычно не бывает. Как правило, уже существует главный сервер, который проработал какое-то время, и требуется синхронизировать с ним новый подчиненный сервер, на котором еще нет копии данных с главного.

Существует несколько способов инициализировать, или «клонировать», подчиненный сервер из имеющегося: копирование данных с главного, клонирование другого подчиненного сервера и загрузка на подчиненный сервер данных из свежей резервной копии. Чтобы синхронизировать подчиненный сервер с главным, необходимы три вещи.

- Мгновенный снимок данных главного сервера в некоторый момент времени.
- Текущий файл журнала главного сервера и смещение от начала этого файла в точности на тот момент времени, когда был сделан мгно-

венный снимок. Вместе они называются *координатами репликации*, так как однозначно идентифицируют позицию в двоичном журнале. Найти координаты репликации вам поможет команда `SHOW MASTER STATUS`.

- Файлы двоичных журналов главного сервера с момента мгновенного снимка до текущего момента.

Существует несколько способов клонировать подчиненный сервер с помощью другого сервера.

Холодная копия

Самый простой способ запустить подчиненный сервер состоит в том, чтобы остановить сервер, который впоследствии станет главным, и скопировать файлы с него на подчиненный сервер (об эффективных способах копирования файлов см. приложение А). После перезапуска главный сервер откроет новый двоичный журнал и можно будет воспользоваться командой `CHANGE MASTER TO`, указав на подчиненном сервере начало файла в качестве позиции в двоичном журнале. Недостаток такого решения очевиден: в течение всего времени копирования главный сервер должен быть остановлен.

Горячая копия

Если все таблицы имеют тип MyISAM, то можно воспользоваться командой *mysqlhotcopy*, которая копирует файлы с работающего сервера. Подробную информацию см. в главе 11.

Использование mysqldump

Если все таблицы имеют тип InnoDB, то чтобы выгрузить данные с главного сервера в дамп, загрузить их на подчиненный и изменить координаты репликации на подчиненном сервере в соответствии с позицией в двоичном журнале главного сервера, можно воспользоваться такой командой:

```
$ mysqldump --single-transaction --all-databases
--master-data=1 --host=server1 | mysql --host=server2
```

Флаг *--single-transaction* говорит, что при выгрузке нужно читать те данные, которые существовали на момент начала транзакции. Возможно, он работает и для других транзакционных систем хранения, но мы этого не проверяли. Если имеются нетранзакционные таблицы, то для получения согласованного дампа всех таблиц следует задать флаг *--lock-all-tables*.

С помощью мгновенного снимка LVM или резервной копии

Если известны координаты в нужном двоичном журнале, то для инициализации подчиненного сервера можно воспользоваться мгновенным снимком LVM или резервной копией (в последнем случае необходимо иметь все двоичные журналы главного сервера с момента снятия этой копии). Восстановите данные из мгновенного снимка или копии на подчиненном сервере, а затем задайте координаты ре-

пликации с помощью команды CHANGE MASTER TO. Дополнительную информацию об этом способе см. в главе 11.

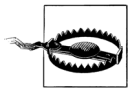
Технология InnoDB Hot Backup, которая также рассматривается в главе 11, – еще один удобный способ инициализировать подчиненный сервер в ситуации, когда все таблицы имеют тип InnoDB.

На основе другого подчиненного сервера

Любой из вышеупомянутых методов годится для клонирования одного подчиненного сервера из другого. Однако флаг *--master-data* в команде *mysqldump* работать не будет.

Еще отметим, что вместо того, чтобы получать координаты репликации в двоичном журнале главного сервера командой SHOW MASTER STATUS, следует воспользоваться командой SHOW SLAVE STATUS для отыскания позиции в двоичном журнале главного сервера, на которой подчиненный сервер остановился в момент снятия мгновенного снимка.

Серьезный недостаток клонирования другого подчиненного сервера состоит в том, что если подчиненный сервер рассинхронизирован с главным, то вы будет клонировать неактуальные данные.



Не пользуйтесь командами LOAD DATA FROM MASTER и LOAD TABLE FROM MASTER! Они устарели, работают медленно и крайне опасны. К тому же они применимы только к таблицам типа MyISAM.

На каком бы методе вы ни остановились, потратьте время на то, чтобы освоиться с ним, и документируйте свои действия или напишите сценарий. Не исключено, что эту процедуру придется проделать не раз, и вы не должны растеряться, если что-то пойдет не так.

Рекомендуемая конфигурация репликации

Параметров репликации много, и большинство из них так или иначе влияют на безопасность данных и производительность. Ниже мы расскажем о том, какие правила можно нарушать и когда. А в этом разделе приведем рекомендуемую «безопасную» конфигурацию, которая сводит к минимуму шансы нарваться на неприятность.

На главном сервере самым важным параметром для двоичного журналирования является sync_binlog :

```
sync_binlog=1
```

При таком значении MySQL сбрасывает двоичный журнал на диск в момент фиксации транзакции, поэтому события журнала не потеряются в случае сбоя. Если отключить этот режим, то работы у сервера станет меньше, но при возникновении сбоя записи в журнале могут оказаться поврежденными или вообще отсутствовать. На подчиненном сервере, который не выступает в роли главного, этот режим приводит к излишним накладным расходам. Он применяется только к двоичному журналу, а не к журналу ретрансляции.

Если повреждение таблиц после сбоя неприемлемо, то мы рекомендуем работать с InnoDB. Подсистема хранения MyISAM хороша, если с поврежденной таблицей можно примириться, но имейте в виду, что после сбоя подчиненного сервера таблицы типа MyISAM могут оказаться в несогласованном состоянии. Есть вероятность, что некая команда будет применена к одной или нескольким таблицам не полностью, поэтому данные останутся несогласованными даже после исправления таблиц.

При использовании InnoDB мы настоятельно рекомендуем задавать на главном сервере следующие параметры:

```
innodb_flush_logs_at_trx_commit=1 # Сброс после каждой записи в журнал
innodb_support_xa=1               # Только в версии MySQL 5.0 и более поздних
innodb_safe_binlog                # только в версии MySQL 4.1, примерный
                                  # эквивалент innodb_support_xa
```

Эти значения подразумеваются по умолчанию в версии MySQL 5.0. На подчиненном сервере мы рекомендуем включить следующие параметры:

```
skip_slave_start
read_only
```

Параметр `skip_slave_start` предотвращает автоматический перезапуск подчиненного сервера после сбоя, это оставляет вам возможность восстановить сервер при наличии проблем. Если же подчиненный сервер перезапускается автоматически после некорректного завершения, а база данных находится в несогласованном состоянии, то повреждение может дойти до такой степени, что все данные придется выбросить и начать все сначала. Даже если вы установите все параметры так, как мы предлагаем, подчиненный сервер все равно может выйти из строя после сбоя, поскольку журналы ретрансляции и файл *master.info* не защищены от повреждений. Они даже не сбрасываются принудительно на диск, и не существует никакого параметра, который управлял бы этим поведением. Разработанная компанией Google заплата, о которой мы поговорим ниже, решает эту проблему.

Параметр `read_only` не дает большинству пользователей изменять какие-либо таблицы, кроме временных. Исключение составляют поток SQL и потоки, работающие с привилегией `SUPER`. Это единственная причина, по которой обычной учетной записи имеет смысл давать привилегию `SUPER` (о привилегиях см. главу 12).

Если подчиненный сервер очень сильно отстает от главного, то поток ввода/вывода может создать множество журналов ретрансляции. Поток SQL удаляет их сразу после воспроизведения (это поведение можно изменить с помощью параметра `relay_log_purge`), но если отставание велико, то поток ввода/вывода вполне может заполнить весь диск. Справиться с этой проблемой поможет конфигурационный параметр `relay_log_space_limit`. Если совокупный размер всех журналов ретрансляции больше значения этого параметра, то поток ввода/вывода приостанавливается и ждет, пока поток SQL освободит место на диске.

На первый взгляд, все хорошо, но здесь таится одна проблема. Если подчиненный сервер не скопировал в журнал ретрансляции все события с главного сервера, то в случае сбоя последнего они могут быть навсегда потеряны. Если места на диске достаточно, то лучше дать возможность подчиненному серверу создавать журналы ретрансляции без ограничений. Именно поэтому мы не включили параметр `relay_log_space_limit` в рекомендуемую конфигурацию.

Взгляд на репликацию изнутри

Теперь, когда мы познакомились с основами репликации, можно копнуть и поглубже. Мы рассмотрим, как на самом деле работает механизм репликации, познакомимся с его сильными и слабыми сторонами и изучим некоторые дополнительные конфигурационные параметры.

Покомандная репликация

Версии MySQL 5.0 и более ранние поддерживали только *покомандную репликацию* (она также называется *логической*). В мире СУБД это необычно. Принцип работы такого механизма заключается в том, что протоколируются все выполненные главным сервером команды изменения данных. Когда подчиненный сервер читает из своего журнала ретрансляции событие и воспроизводит его, на самом деле он отрабатывает в точности ту же команду, которая была ранее выполнена на главном сервере. У такого решения есть свои плюсы и минусы.

Очевидный плюс – относительная легкость реализации. Простое журналирование и воспроизведение всех предложений, изменяющих данные, теоретически поддерживает синхронизацию подчиненного сервера с главным. Еще одно достоинство покомандной репликации состоит в том, что события в двоичном журнале представлены компактно. Иначе говоря, покомандная репликация потребляет не слишком большую часть пропускной способности сети – запрос, обновляющий гигабайты данных, занимает всего-то несколько десятков байтов в двоичном журнале. Кроме того, уже упоминавшийся инструмент *mysqlbinlog* удобнее использовать именно для покомандной репликации.

На практике, однако, покомандная репликация не так проста, как кажется, поскольку многие изменения на главном сервере могут зависеть от факторов, не выражаемых в тексте запроса. Например, моменты выполнения команд на главном и подчиненном сервере могут слегка – или даже сильно – различаться. Поэтому в двоичном журнале MySQL присутствует не только текст запроса, но и кое-какие метаданные, такие как временная метка. Но все равно некоторые команды невозможно реплицировать корректно, в частности, запросы, в которых встречается функция `CURRENT_USER()`. Проблемы возникают также с хранимыми процедурами и триггерами.

С покомандной репликацией связана еще одна неприятность – модификации должны быть сериализуемы. Для этого приходится обрабатывать в коде сервера различные особые случаи, вводить специальные параметры и неоправданные функции. Например, в этой связи можно упомянуть блокировку следующего ключа в InnoDB и блокировку при выполнении автоинкремента. Не все подсистемы хранения корректно поддерживают покомандную репликацию, хотя подсистемы, включенные в официальный дистрибутив MySQL вплоть до версии 5.1, с этим справляются.

Полный перечень недостатков покомандной репликации можно найти в соответствующей главе руководства по MySQL.

Построчная репликация

В версии MySQL 5.1 добавилась поддержка *построчной репликации*, при которой в двоичный журнал записываются фактические изменения данных, как это делается в большинстве других СУБД. У такой схемы есть свои плюсы и минусы. Самое существенное достоинство заключается в том, что теперь MySQL может корректно реплицировать любую команду, причем в некоторых случаях это происходит гораздо более эффективно. Основной недостаток – это то, что двоичный журнал стал намного больше и из него непонятно, какие команды привели к обновлению данных, так что использовать его для аудита с помощью программы *mysqlbinlog* уже невозможно.

Построчная репликация не является обратно совместимой. Утилита *mysqlbinlog*, входящая в дистрибутив MySQL 5.1, может читать двоичные журналы, содержащие события в формате построчной репликации (он не предназначен для чтения человеком). Однако версии *mysqlbinlog* из предыдущих версий MySQL такой журнал не распознают и при попытке прочитать его завершаются с ошибкой.

Некоторые изменения, хранящиеся в формате построчной репликации, MySQL воспроизводит более эффективно, так как ему не приходится повторять запросы, выполненные на главном сервере. А ведь воспроизведение определенных запросов обходится весьма дорого. Например, следующий запрос агрегирует данные из большой таблицы, помещая результаты в таблицу, меньшую по размерам:

```
mysql> INSERT INTO summary_table(col1, col2, sum_col3)
    -> SELECT col1, col2, sum(col3)
    -> FROM enormous_table
    -> GROUP BY col1, col2;
```

Предположим, что в таблице `enormous_table` есть всего три уникальных комбинации столбцов `col1` и `col2`. В этом случае запрос просматривает очень много строк в исходной таблице, но результатом является вставка лишь трех строк в конечную таблицу. Если это событие реплицировать

как команду, то подчиненный сервер должен будет повторить всю работу для того, чтобы вычислить эти три строки, тогда как построчная репликация обходится до смешного дешево. В данном случае построчная репликация многократно эффективнее.

С другой стороны, следующее событие гораздо дешевле обрабатывается методом покомандной репликации:

```
mysql> UPDATE enormous_table SET col1 = 0;
```

Применение построчной репликации для данного запроса обходится очень дорого, так как изменяется каждая строка, следовательно, каждую строку нужно будет записать в двоичный журнал, который вырастет до неимоверных размеров. В результате нагрузка на главный сервер резко возрастает как на этапе сохранения данных в журнале, так и на этапе репликации, а из-за медленной записи в журнал может пострадать конкурентность.

Поскольку ни тот, ни другой формат не идеален, MySQL 5.1 динамически переключается с одного на другой по мере необходимости. По умолчанию применяется покомандная репликация, но если обнаруживается событие, которое невозможно корректно реплицировать командой, то сервер переходит на построчную репликацию. Разрешается также явно управлять форматом с помощью сеансовой переменной `binlog_format`.

Если двоичный журнал представлен в формате построчной репликации, то восстановить данные на конкретный момент времени в прошлом становится более сложно, но все-таки возможно. В этом может помочь сервер журнала, но подробнее об этом мы поговорим ниже.

Теоретически построчная репликация решает некоторые из вышеупомянутых проблем. На практике же многие наши знакомые, работающие с MySQL 5.1, по-прежнему предпочитают покомандную репликацию на промышленных серверах. Поэтому пока о построчной репликации трудно сказать что-нибудь определенное.

Файлы репликации

Теперь рассмотрим некоторые файлы, участвующие в процессе репликации. О двоичном журнале и журнале ретрансляции вы уже знаете, но есть и другие файловые объекты. Конкретное место их хранения зависит в основном от конфигурационных параметров MySQL. В разных версиях СУБД умолчания различны. Чаще всего их можно найти в каталоге данных или в каталоге, где находится *pid*-файл сервера (в UNIX-системах это обычно каталог */var/run/mysqld/*). Перечислим их.

mysql-bin.index

Если запись в двоичный журнал включена, то сервер создает файл с таким же именем, как у двоичных журналов, но с расширением

.index. В нем регистрируются все файлы двоичных журналов, имеющиеся на диске. Это не индекс в том смысле, в каком мы говорим об индексах по таблицам; он просто состоит из текстовых строк, в каждой из которых указано имя одного файла двоичного журнала.

Вероятно, у вас возникла мысль, что этот файл лишний и может быть удален (в конце концов, MySQL может просто найти все файлы на диске). Не делайте этого! MySQL игнорирует двоичные журналы, не указанные в индексном файле.

mysql-relay-bin.index

Этот файл играет ту же роль для журналов ретрансляции, что рассмотренный выше файл для двоичных журналов.

master.info

В этом файле хранится информация, необходимая подчиненному серверу для соединения с главным. Формат текстовый (по одному значению в строке) и изменяется в зависимости от версии MySQL. Не удаляйте его, иначе после перезапуска подчиненный сервер не будет знать, как подключиться к главному. Поскольку в этом файле может храниться пароль пользователя в открытом виде, будет разумно максимально ограничить права доступа к нему.

relay-log.info

В этом файле на подчиненном сервере хранятся имя его текущего двоичного журнала и координаты репликации (то есть место в двоичном журнале главного сервера, до которого дошел подчиненный). Не удаляйте этот файл, иначе после перезапуска подчиненный сервер не будет знать, с какого места продолжить репликацию, и попытается воспроизвести уже выполненные команды.

Совокупность этих файлов представляет собой довольно прямолинейный способ сохранить состояние репликации и журналов. К сожалению, запись в них производится не синхронно, поэтому если произойдет сбой питания в момент, когда файлы не были сброшены на диск, то после перезапуска данные в них окажутся некорректными.

По умолчанию имя двоичного журнала образуется из имени хоста, к которому добавляется цифровой суффикс, но лучше задать базовое имя явно в файле *my.cnf*:

```
log_bin             # Не делайте этого, если не хотите,
                    # чтобы имя образовывалось из имени хоста
log_bin = mysql-bin # Так безопасно
```

Это существенно, поскольку в случае изменения имени хоста репликация может перестать работать. Мы также не рекомендуем выбирать в качестве базового имени имя хоста; иными словами, не делайте умолчание явным решением. Лучше выберите какое-то одно имя для двоичных журналов и используйте его на всех серверах. Тогда будет гораздо

проще перемещать файлы сервера с одной машины на другую и автоматизировать переключение в случае сбоя (failover).

Следует также явно выбирать имена для журналов ретрансляции (которые тоже по умолчанию образуются из имени хоста) и соответствующих *index*-файлов. Вот как мы рекомендуем задавать все эти параметры в файле *my.cnf*:

```
log_bin         = mysql-bin
log_bin_index   = mysql-bin.index
relay_log       = mysql-relay-bin
relay_log_index = mysql-relay-bin.index
```

Вообще-то *index*-файлы по умолчанию наследуют имена от соответствующих файлов журналов, но не будет никакого вреда, если задать их явно.

На *index*-файлы влияет также параметр `expire_logs_days`, определяющий, сколько времени MySQL должен сохранять уже закрытые двоичные журналы. Если в файле *mysql-bin.index* упоминаются файлы, отсутствующие на диске, то автоматическое удаление работать не будет; даже команда `PURGE MASTER LOGS` ничего не даст. В общем случае для решения этой проблемы нужно поручить управление двоичными журналами самому серверу MySQL, уж он-то не запутается.

Необходимо выработать стратегию удаления старых журналов – с помощью параметра `expire_logs_days` или иными средствами, – иначе рано или поздно MySQL заполнит двоичными журналами весь диск. При этом следует учитывать и принятую в организации политику резервного копирования. Дополнительную информацию о двоичном журнале см. в разделе «Формат двоичного журнала» на стр. 601.

Отправка событий репликации другим подчиненным серверам

Параметр `log_slave_updates` позволяет использовать подчиненный сервер в роли главного для других подчиненных. Он заставляет сервер MySQL записывать события, выполняемые потоком SQL, в собственный двоичный журнал, доступный подчиненным ему серверам. Эта схема изображена на рис. 8.2.

В данном случае любое изменение на главном сервере приводит к записи события в его двоичный журнал. Первый подчиненный сервер извлекает и исполняет это событие. Обычно на этом жизнь события и завершилась бы, но, поскольку включен режим `log_slave_updates`, подчиненный сервер записывает его в свой двоичный журнал. Теперь второй подчиненный сервер может извлечь это событие и поместить в свой журнал ретрансляции. Такая конфигурация означает, что изменения, произведенные на главном сервере, распространяются по цепочке подчиненных серверов, не подключенных напрямую к главному. Мы предпо-

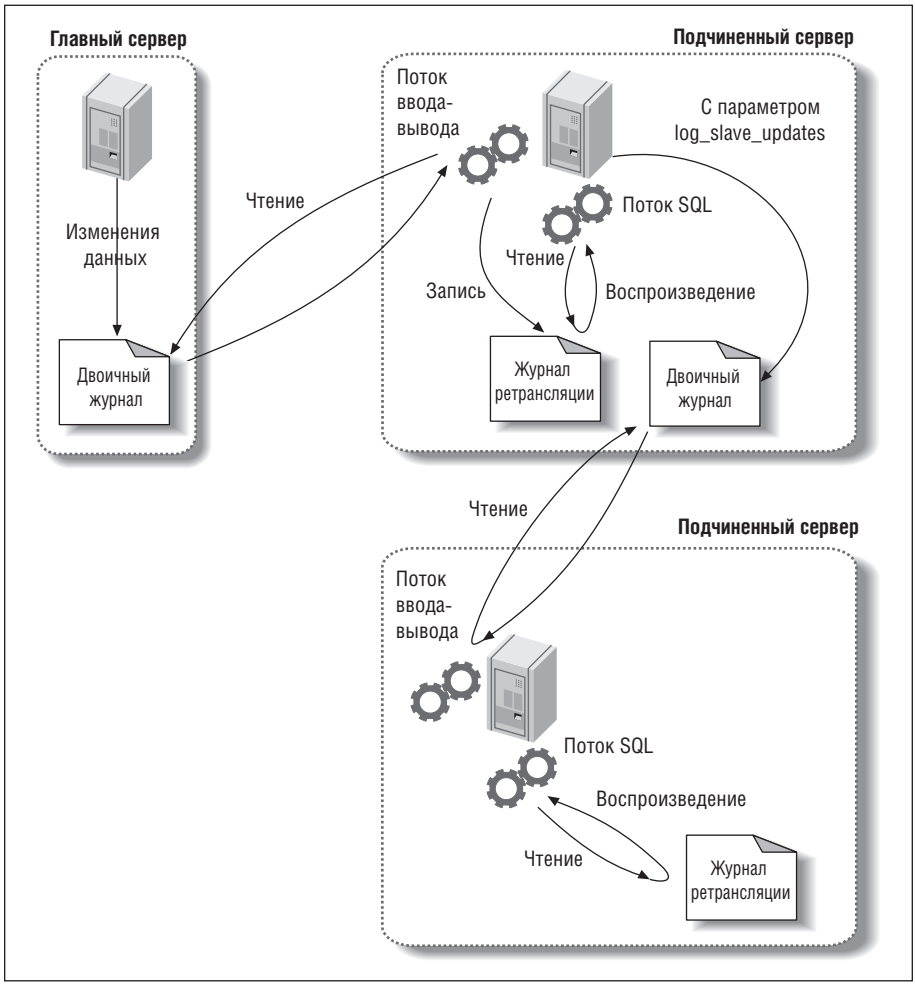


Рис. 8.2. *Передача события репликации по цепочке подчиненных серверов*

читаем по умолчанию оставлять режим `log_slave_updates` включенным, так как это позволяет подключать подчиненный сервер без перезапуска сервера.

Когда первый подчиненный сервер переписывает событие из двоичного журнала главного сервера в свой двоичный журнал, его позиция в журнале почти наверняка изменится, то есть она может оказаться либо в другом файле, либо по другому смещению. Поэтому нельзя предполагать, что все серверы, логически находящиеся в одной и той же точке репликации, имеют одинаковые координаты репликации. Позже мы увидим, что это существенно осложняет решение некоторых задач,

например подключение подчиненного сервера к другому главному или преобразование подчиненного сервера в главный.

Если не позаботиться о том, чтобы у каждого сервера был уникальный идентификатор, то подобное конфигурирование подчиненного сервера может послужить причиной трудноуловимых ошибок и даже привести к полной остановке репликации. Часто спрашивают, зачем нужно присваивать серверу идентификатор. Разве MySQL не может реплицировать команды, не зная их происхождения? Почему MySQL хочет, чтобы идентификатор сервера был глобально уникальным? А все дело в том, чтобы предотвратить бесконечные циклы в процедуре репликации. Когда поток SQL на подчиненном сервере читает журнал ретрансляции, он отбрасывает все события, в которых идентификатор сервера совпадает с его собственным. Тем самым бесконечный цикл разрывается. Предотвращение бесконечных циклов важно в таких важных топологиях репликации, как главный–главный (master–master).

При возникновении затруднений с настройкой репликации первым делом обратите внимание на идентификатор сервера. Недостаточно просто проверить переменную `@@server_id`. У нее всегда есть какое-то значение по умолчанию, но репликация не будет работать, если значение не задано явно – в файле *my.cnf* или командой `SET`. Если вы пользуетесь командой `SET`, не забудьте также обновить конфигурационный файл, иначе измененные настройки пропадут после перезапуска сервера.

Фильтры репликации

Параметры фильтрации позволяют реплицировать только часть данных, хранящихся на сервере. Есть два вида фильтров репликации: одни применяются при записи событий в двоичный журнал на главном сервере, а другие – при чтении событий из журнала ретрансляции на подчиненном сервере. Те и другие изображены на рис. 8.3.

Фильтрацией двоичного журнала управляют параметры `binlog_do_db` и `binlog_ignore_db`. Но, как мы скоро поясним, их как раз использовать *не стоит*.

Параметры `replicate_*` управляют фильтрацией событий, считываемых потоком SQL из журнала ретрансляции на подчиненном сервере. Можно реплицировать или игнорировать одну или несколько баз данных, переписывать одну базу данных в другую и реплицировать или игнорировать определенные таблицы, задаваемые с помощью предиката `LIKE`.

Очень важно понимать, что параметры `*_do_db` и `*_ignore_db` – как на главном, так и на подчиненном сервере, – работают не так, как можно было бы ожидать. Естественно полагать, что фильтрация производится по имени базы данных объекта, но на самом деле анализируется имя