MySQL Replication(复制)

用 MySQL5.5 提高可扩展性和可用性

MySQL 技术白皮书

2010 年 10 月

目录

1. 简介

MySQL Replication(复制)已经在一些著名的网站和企业广泛应用以将数据库的扩展性提升到极限水平。对用户而言可以简单快速地为数据库创建多个副本，超越单个数据库实例容量的限制，弹性扩展数据库系统以满足快速增长的数据库负载。

复制也可以作为高可用数据库服务的基础，提供多主机的镜像数据来避免单点故障。另外，许多用户把数据复制到专门用于备份或者数据分析的系统，提高资源利用率，减轻应用系统的这些负担。

随着MySQL5.5 版本的发布，MySQL 复制性能提升了许多，提供更高级别的数据整合、性能和应用灵活性。我们会在白皮书中讨论这些提升的特性。

在本白皮书中我们还会探索部署MySQL 复制的业务和技术的好处；描述复制背后的基本技术并且给出一个简单的指南教你如何一步步的安装并配置一个主/从拓扑，以及处理故障转移事件。

最后，本白皮书概述了用MySQL Cluster数据库与用MySQL5.5 做复制的主要区别。

2. 复制基础

本文中我们把“复制”定义为将数据复制到另一个或多个地方。接下来的部分我们将涉及流行的几种复制类型之间的差异。

复制使一个数据库能从一个物理地址或者系统拷贝/复制数据库的变化到另一个物理地址或者系统（通常是从“主”到“从”系统）。这经常被用来增加数据库的可用性和扩展性，用户需要经常在从系统上执行备份操作或者运行分析查询语句，使主系统的不需做这些功能。

MySQL本身支持复制并把这當成数据库的一项标准特征。依靠配置，你可以复制全部数据库、选中的数据库、甚至是在一个数据库中选中的表。

MySQL 复制的工作方式很简单，一台服务器作为主机，一台或多台服务器作为从机。主机会把数据库的变化记录到日志。一旦这些变化被记录到日志，就会立刻（或者以设定的时间间隔）被送到从机。

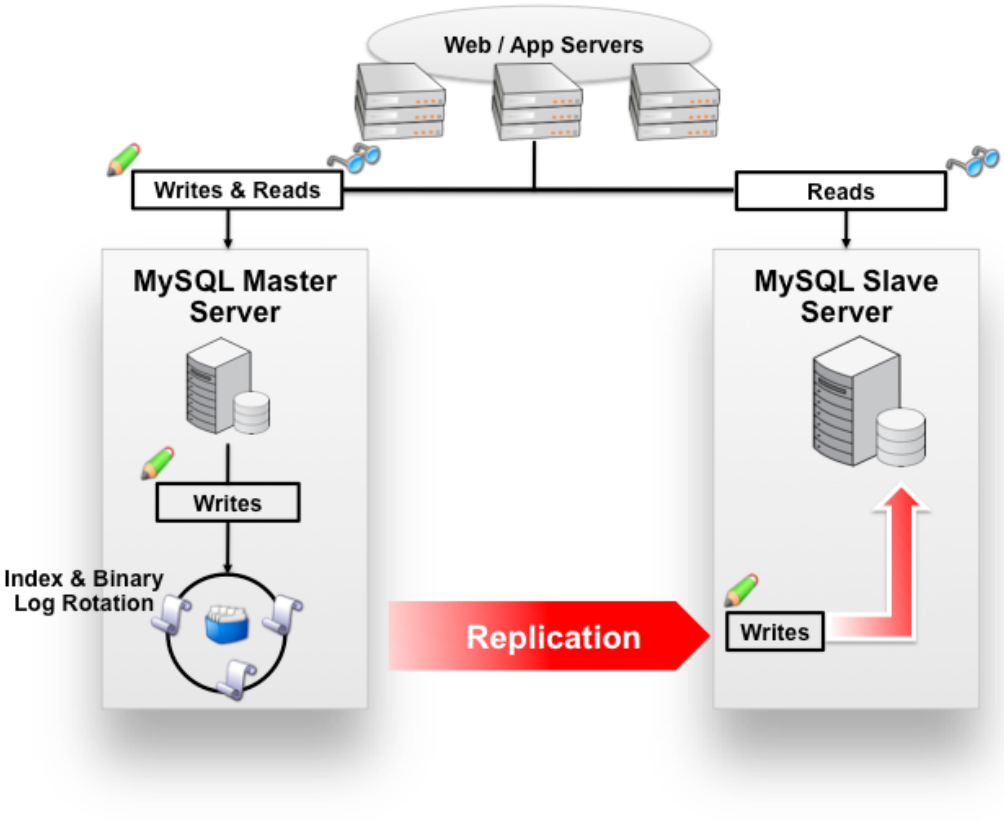
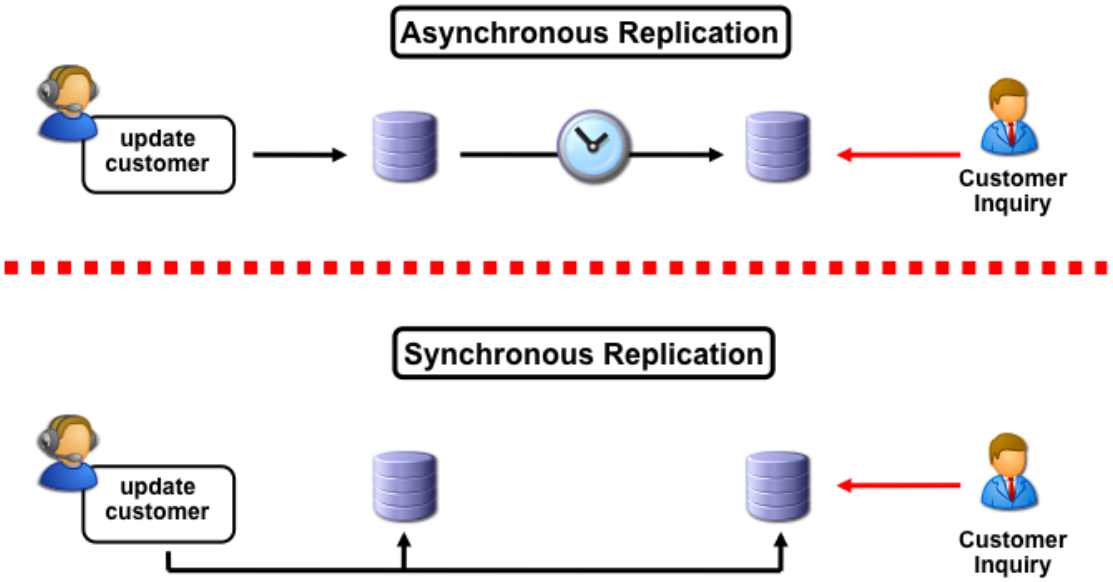


图 1 MySQL 复制支持高可用和扩展性

使用MySQL 复制提供扩展大型网站的能力，这些大型网站的数据库主要是读操作(SELECTs)。从机用於复制主机的銷耗是很少的（通常每个从机 1%的开销），在大型网站中每个主机部署 30 个从机也是常见的。

异步复制

MySQL本身支持单向的、异步的复制。异步复制意味着在把数据从一台机器拷贝到另一台机器时有一个延时 - 最重要的是这意味着当应用系统的事务提交已经确认时数据并不能在同一时刻拷贝/应用到从机。通常这个延时是由网络带宽、资源可用性和系统负载决定的。然而，使用正确的组件并且调优，复制能做到接近瞬时完成。



图二：异步Vs.同步复制

同步复制

同步复制可以定义为数据在同一时刻被提交到一台或多台机器，通常这是通过众所周知的“两阶段提交”做到的。虽然这确实给你在多系统中保持一致性，但也由于增加了额外的消息交换而造成性能下降。

使用MyISAM或者InnoDB存储引擎的MySQL本身并不支持同步复制，然而有些技术，例如分布式复制块设备（简称DRBD），可以在下层的文件系统提供同步复制，允许第二个MySQL服务器在主服务器丢失的情况下接管（使用第二服务器的复本）。要了解更多信息，请参见：http://www.drbd.org/

如果使用MySQL Cluster（集群），数据在集群内的各数据节点同步复制然后在不同的集群内异步复制。

半同步复制

MySQL5.5 的新功能之一就是半同步复制这意味着如果主机开启了半同步复制并且至少配置了一台半同步从机，主机上就有一个线程执行事务提交并且等待至少一台半同步从机返回一个确认收到事务所有事件的消息给主机或者一直等到出现超时。在超时发生时，主机仍然会提交事务，但是以异步模式提交。

使用异步复制，如果主机崩溃，那么不能立刻知道主机提交的事务是否已经复制给了从机。因此，从主机到从机的故障迁移可能会导致服务器丢失主机相关

的事务。半同步复制将主机上“因崩溃而未复制的”事务的潜在可能降到最低，因为所有提交的事务都由从机接收 - 换句话说，对任何一个客户端线程，作为“提交中”事务一部分的主机改变的数据会丢失（所以客户端会重试），但当事务提交已经被确认时，改变的数据就会保留下来。

半同步复制确实有些影响性能，由于需要等待从机，提交事务会变得缓慢。这是增加数据整合度的代价。减慢的速度时间量至少是把事务提交给从机的一个TCP/IP往返时间，包括从机把提交的事务记录到中继日志并等待从机收到的确认时间。这意味着半同步复制服务器在同一地点、通过高速的网络通信的情况下工作效率最高。

基于语句的复制

默认情况下，MySQL使用基于语句的复制，也就是SQL语句（不是实际的数据改变）被复制到从机。从 3.23 版本开始，基于语句的复制就是MySQL服务器的一部分。

在一些情况下基于语句的复制的一个好处是，写到日志文件的数据减少了，比如在修改或者删除指令影响很多行的时候。对于仅仅影响几行数据的简单语句，基于行的复制能节约更多的空间。

基于语句的复制也有一些缺点，最明显的是它不能支持具有不确定行为的语句 - 比如当前时间的函数。

基于行的复制

MySQL5.1 引入了基于行的复制，记录单个表的行变化到日志而不是语句。用基于行的复制，主机把消息或者事件写到二进制日志，记录单个表的行是怎么改变的。这更像是其他关系型数据库的传统的复制形式。通常基于行的复制需要更少的主机和从机锁，这意味着可能达到更高的并发数。基于行的复制的一个缺点是通常可能需要记录比必要的数据更多的数据量。例如，如果一句语句改变了一张表内的一百行，用基于行的复制就要记录 100 条改变，而基于语句的复制只要记录一条SQL语句。

在MySQL5.5 版之前，用基于行的复制时，主和从的列类型必须相同。而对于MySQL5.5，你可以配置是否允许类型升级并且/或者降级，或继续执行严格的类型检查。

如果用MySQL Cluster，那么必须使用基于行的复制。

混合格式复制

从MySQL5.1.8 开始，在混合格式日志模式下，二进制日志格式能够根据记录日志的事件实时改变。开启混合格式，默认使用基于语句的复制，但在某些情况下自动切换成基于行的复制 - 例如：

- 一句更新MySQL Cluster表的DML语句
- 一句包含UUID（）的语句
- 两个或更多使用了AUTO_INCREMENT列的表被更新
- 当执行任何 INSERT DELAYED时
- 当视图的主体需要基于行的复制时，创建视图的语句也包含这些情况 - 例如，当使用UUID（）函数创建视图时
- 调用用户定义函数（UDF）时

要得到这些情况的全部列表请参见：
http://dev.mysql.com/doc/refman/5.1/en/binary-logmixed.html

3. 在 MySQL5.5 中改变的复制内容概要

MySQL5.5 对复制做了很多提高；本章节提供了这些提高的概要，其中一些提高的内容在本文档的其他地方有更详细的描述：

- **半同步复制**：对于持续事件，让主机等待从机以提高适应能力。
- **从机fsync调优和自动的中继日志恢复**：用选项来决定中继日志何时写入磁盘而不是依靠默认的操作系统行为。设置sync_relay_log=1 来确保在崩溃后中继日志最多只丢失一条事务语句。从机就可以通过请求主机重新发送损坏的日志入口来从损坏的中继日志恢复数据。引入了三种新的选项（sync-master-info,sync-relay-log和sync-relay-log-info），它们的用法在MySQL参考手册中有描述：
  http://dev.mysql.com/doc/refman/5.5/en/replication-optionsslave.html#sysvar_sync_master_info
- **复制心跳**：自动检查主从机之间的连接状态，允许更精确的故障检测机制。能够检查到毫秒级（可配置）的连接丢失。避免当主机空闲时不必要的中继日志轮换。

- **每服务器复制过滤：**当一台服务器从一个复制环中移除时，可以选择一台正常工作的服务器移除已经应用到所有服务器的复制消息。
- **精确的从机类型转换：**允许不同的类型用在主机和从机上，当使用基于行的复制时自动升降类型（基于语句的复制也已经能够做到）
- **单独的日志刷新：**当使用‘FLUSH LOGS’时可选择性地刷新服务器日志以达到更好的控制
- **安全的混合事务日志：**复制同时包含了InnoDB和MyISAM变化的事务。

4．复制使用案例

选择MySQL 复制的技术和业务理由多种多样。在这一章我们探索使用MySQL 复制的各种使用案例和应用场景。

水平扩展

这是用户选择使用复制最常见的原因。在水平扩展的拓扑下，主要的目的就是将负载分散到一台或多台从机来提高性能。

对用户来说，快速地创建多个数据库副本，超越单个数据库实例的容量限制来弹性扩展是很简单的，这使得他们能够应付日益增长的数据库负载。

水平扩展是相对于垂直扩展的，垂直扩展的思路是在机器上增加资源（通常是RAM和CPU）。垂直扩展可以想象成是垂直的，“叉车式”的方式来满足增长的容量需求。

在水平扩展的架构下，读和写在主机和从机之间是分离的。

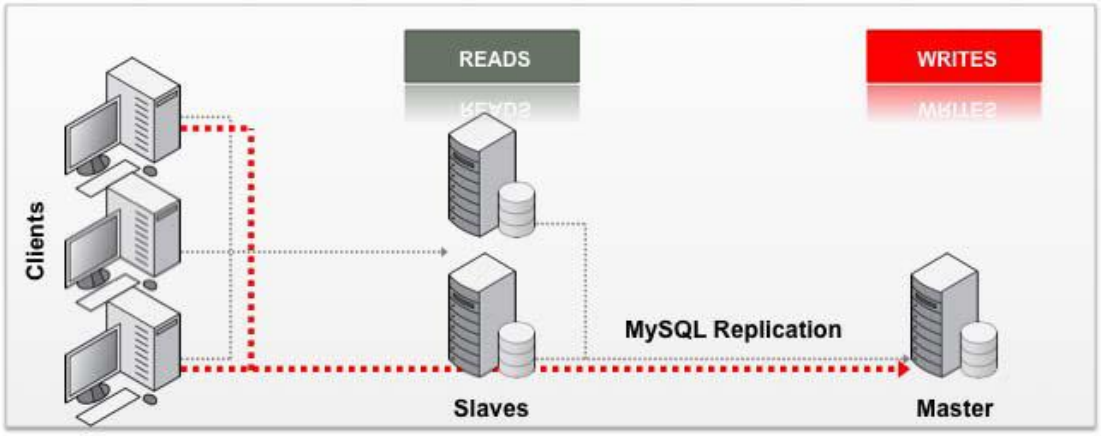


图 3 用MySQL 复制来水平扩展

特别的是，所有的写操作（UPDATE，INSERT，DELETE）都被送到主机执行而读操作（SELECT）被送到从机执行。这使得原先要在主机上执行的读操作负载可以用从机上的资源来替代完成。这样更有效地利用了资源，因为负载有效地分散到了多台服务器。

读写分离可以在系统的不同层处理，例如在应用层（它保持所有到服务器的连接，并决定何时使用到主机的连接而不是到从机的连接）内部或者在数据库连接器内部。

例如，在用MySQL JDBC连接器（Connector/J）的时候，在连接字符串中你可以提供多个服务器，以主机开始，然后是从机。如果我们用图 10 的配置（“black”是主机，“blue”和“green”是从机），那么应用程序可以通过使用连接字符串“jdbc:mysql:replication://black,blue,green/clusterdb”来连接到“clusterdb”数据库。一旦用这种方式连接，Connector/J会根据连接的属性”ReadOnly”的值（通过connection.getReadOnly()来查询这个属性，并用connection.setReadOnly(bool)来写这个属性）来把事务路由到合适的服务器（主/从）。

由于复制是异步的，如果应用程序需要一个只读操作来使用数据库中最新的数据，并绝对确定会把指令传给主机而不是从机。在这种情况下用Connector/J就意味着运行connection.setReadOnly(false)。

高可用

在这种场景下，想法是从主机复制变化到从机，目的是当主机故障下线时自动切换到从机而不是报错、崩溃或者维护。

和水平扩展一样，不同的方式下应用不同的服务器选择。例如你在图 9 中显示的配置方式下使用Connector/J，那么应用程序可以使用“jdbc:mysql://black,blue/clusterdb”作为连接字符串；Connector/J会在“black”正常工作的时候把所有操作送到“black“而在”black“故障的时候把所有操作送到”blue“。

异地复制

异地复制的目的是在两个在地理上分散的地点之间复制数据，通常是间隔很远

的距离。由于网络延时的影响，异步复制在这种场景下更加适用。例如把数据从纽约的中心办公室复制到旧金山的地区办公司，就可以在本地的数据库里执行查询了。显然这种方式对于某个地点发生灾难时（例如停电或者自然灾害）提供灾难恢复也非常适用。

图 4 地理冗余的复制

备份数据库

为避免任何因主机备份数据库而造成的性能下降或锁死，你可以选择在从机上备份数据库。注意，在使用MySQL Cluster的时候主数据库可以在线备份。

分析

许多商业智能或者分析查询占用大量资源并且花很长的时间来执行。在这种场景下，可以创建多台从机来执行这些分析查询。按照这种配置，主机不会因为执行这些查询而影响性能。

这对于MySQL Cluster尤其有用，因为MySQL Cluster主要是基于主键的读写应用的理想选择，但在大数据集中执行非常复杂的查询却非常缓慢。简单地把MySQL Cluster的数据复制到另一台数据引擎（通常是MyISAM或InnoDB）并在那里生成你的报表。这可以在为了地理冗余而复制到远程的MySQL Cluster时的同时完成 - 如下面的图 5 所示。

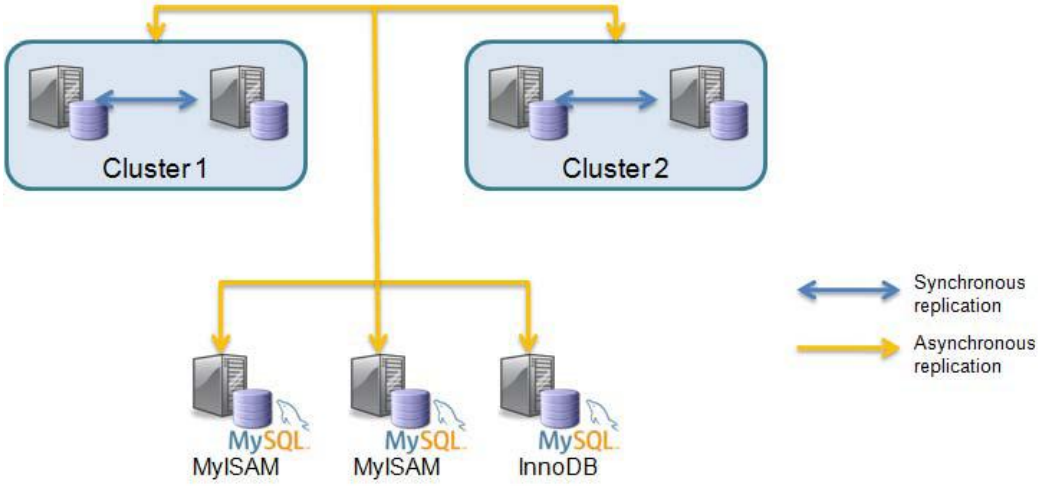


图 5 组合复制用例

5. 复制拓扑

MySQL支持多种不同的复制拓扑。下面我们讨论其中的一些拓扑以及一些有所保留的支持的拓扑。

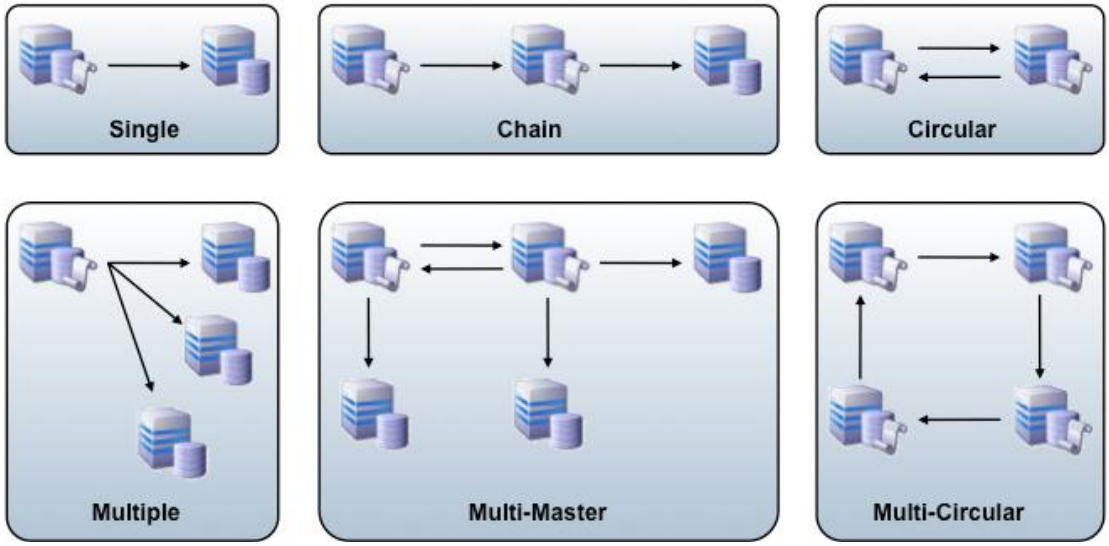


图 6 常见的MySQL复制拓扑

主从

这是最常见的也是最易于配置和管理的拓扑形式。在这种拓扑下我们有两台服务器，一台做主机，一台做从机。写操作都在主机上执行而读操作可以分散到主机和从机之间。

一主多从

这种场景下多台从机连到主机。这可以大幅度水平扩展系统，但代价是增加了管理成本。

主从从

这种配置是主从或者一主多从配置的扩展。在这种配置下，可以给已经附着到主机的从机添加从机的从机。这样配置的效果是，处在中间位置的从机既是主机又是从机。在这种配置下，所有的写操作都在第一级主机上执行。

多主

在主/主配置下，两台服务器组合成一个链，互为主从。虽然，这种配置带来了能够在任一个系统中写数据的好处，但确实明显增加了设置、配置和管理的复杂度。另外，除非你用MySQL Cluster，否则不会有冲突检测机制以至于应用程序必须确定不去更新还没有被复制的行。

也可以用几台MySQL服务器搭成环状，得到更高水平的扩展性和性能（假设应用程序能避免向不同服务器的相同的行发送冲突的更新数据）。MySQL5.5 引入了新的过滤特性，能够更好地处理当一台服务器正在更新复制到环中其他服务器时发生故障的情况。

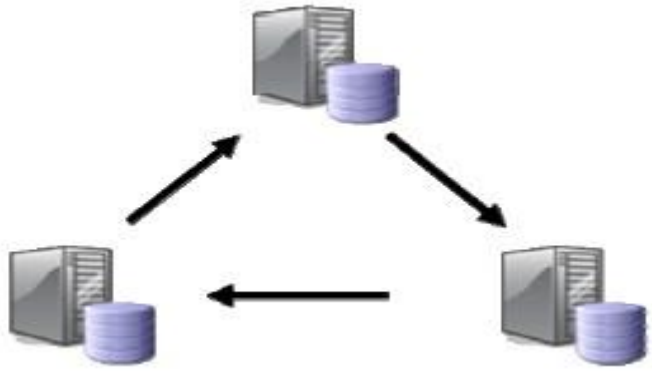


图 7 多主环

当在使用多主复制的同时用了自动增长的列，你应该在每台服务器上使用auto_increment_offset和auto_increment_increment参数来确保不会重复分配值。例如一个 2 台服务器的情况如下面的表 1 所示。

Server	auto_increment_increment	auto_increment_offset	Values
black	2	1	1,3,5...
blue	2	2	2,4,6...

表 1 避免自动增长冲突的参数值

多主一从（多源）

MySQL目前还不支持这种复制拓扑。在多主的配置下，从机基本上“为两台主机服务”，意味着从机从超过一台的主机复制变化。

由于MySQL Cluster允许多台MySQL服务器写到同一个Cluster，因此在一些合适的应用场景，如果需要的话配置这种功能是可能的（每台MySQL服务器仅仅是一台主机的从机）。

6. 复制的内部工作流

使用MySQL复制，主机把更新的内容写到二进制日志文件并维护这些文件的索引以跟踪日志流转。二进制日志文件作为更新的记录被送到从机上。当从机连到主机时，会决定从上次它成功更新的点开始读取记录。然后从机就开始接收从那个时间以后的更新记录。接着从机截断且等待主机通知它有新的更新。这些概念的基本描述见下面的图 8。

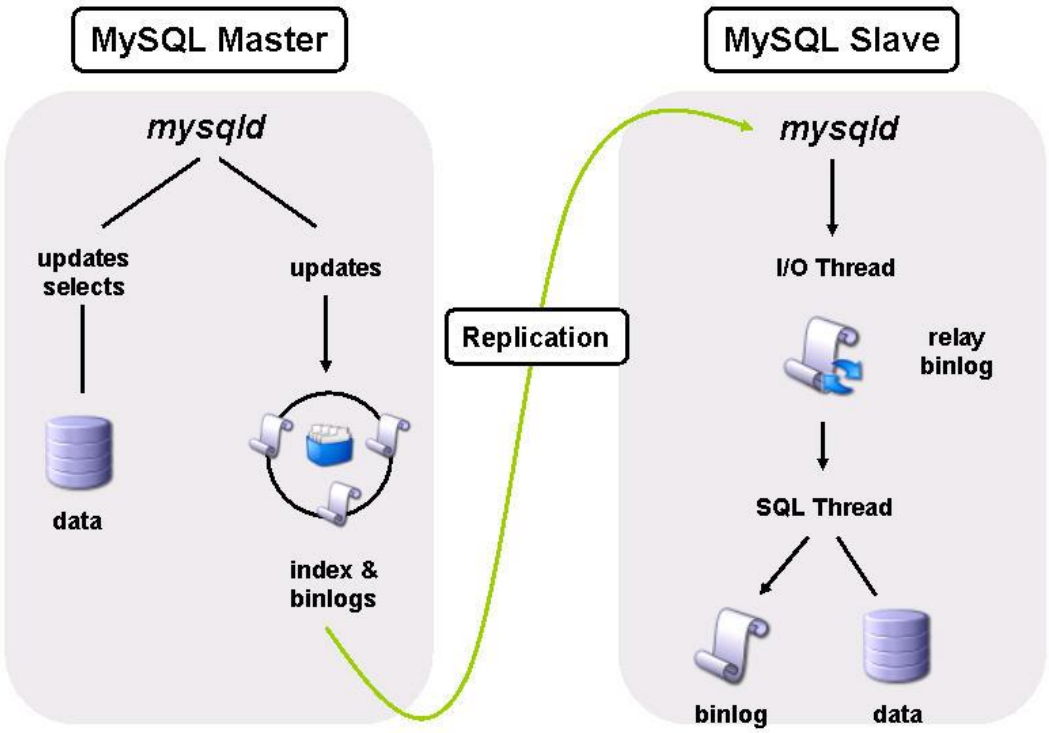


图 8 MySQL复制时如何应用的

复制线程

许多线程被用来实现从主机到从机的复制；每个线程都在这里有描述。如果使用MySQL Cluster，那么还包含额外的线程 – 详见第 11 章节。

Binlog Dump 线程

主机创建这个线程来发送二进制日志内容给从机。Binlog dump线程在读取每个被送到从机的事件时需要锁住主机的二进制日志。一旦读取了事件，就解锁了，即使事件还没发送到从机。带了多台从机的主机为当前每个连接的从机创建一个binlog dump线程，每个从机有自己的I/O和SQL线程。

从机 I/O 线程

当在从机上执行START SLAVE指令时，I/O线程就被创建了，这个线程连接到主机并要求主机用二进制日志发送更新。从机I/O线程读取主机binlog dump线程送来的更新消息，然后在从机本地数据目录里面的中继日志中复制更新消息。

从机 SQL 线程

从机创建这个线程来读取从机I/O线程写的中继日志并执行包含在中继日志中的更新消息。

复制日志文件

在复制过程中，MySQL服务器创建许多用于保持从主机来的中继二进制日志的文件并在中继日志中记录关于当前状态和位置的信息。

从机用了三种类型的文件：

Relay log

从机上的中继日志(relay log)包含了从主机的二进制日志读取的事件。二进制日志里的事件最终是在从机上由从机的SQL线程执行的。

Master.info

从机的状态和当前的配置信息保存在master.info文件里面。这个文件包含了从机的复制连接信息，包括主机名，使用的登录信息和从机在主机二进制日志中的当前位置。

Relay-log.info

关于从机中继日志的执行点的状态信息可以在relay-log.info文件中找到。

在主机上，二进制日志和关联的索引文件来跟踪所有被复制的更新消息。

7. 配置 MySQL 复制

这一章节我们描述一种设置MySQL 复制的方法。虽然，这个过程是根据设置单个从机来写的，但也可以重复步骤来设置多台从机。在本指南中，我们假设你已经成功下载并安装了至少两台MySQL服务器。

关于这个例子的注意点：

我们配置了两台MySQL5.5 服务器，这两台服务器工作的角色如下：

Master
Host Name: black
IP: 192.168.0.31

Slave
Host Name: blue
IP:192.168.0.34

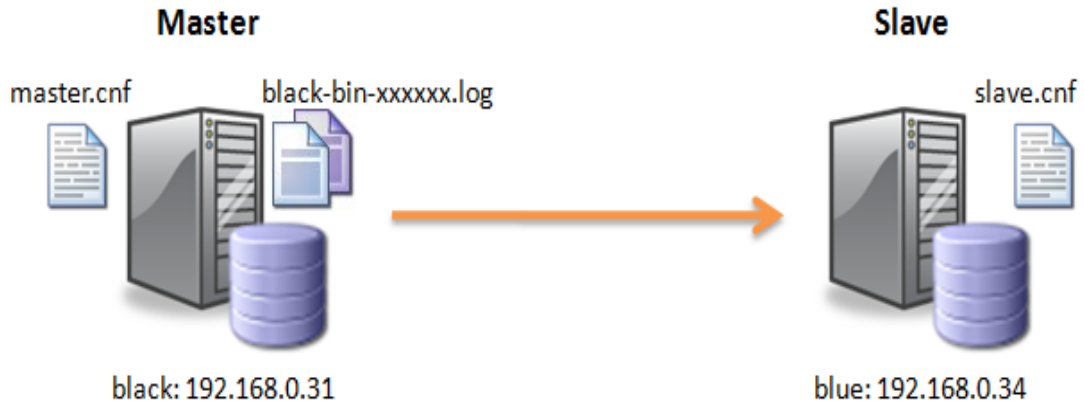


图 9 例子使用的配置

假定你用InnoDB存储引擎。参考第 11 章来理解MySQL Cluster(NDB)存储引擎有什么不同。

为了涵盖更复杂的使用案例，这个例子假定用作主机的MySQL服务器已经在使用了并且已经包含了需要复制给新的从机的数据。如果从空的数据开始，那么步骤 3 和 4 可以跳过。

兼容性

如果你想在两台已经安装了MySQL的服务器上设置复制，请确认主机和从机上的MySQL版本是兼容的。兼容的版本列表参见：
http://dev.mysql.com/doc/refman/5.5/en/replicationcompatibility.html

步骤 1：配置主机和从机的 cnf 文件

设置复制的第一步是编辑主机和从机的“my.cnf“文件。MySQL在安装的时候就包含了这个默认文件了，但为防止已经有在其他服务器上运行的MySQL服务器，我们提供本地配置文件”master.cnf“和”slave.cnf“，当启动MySQL服务器的时候，这些配置文件就会生效。

在master.cnf文件的【mysqld】部分，我们至少增加两个选项：

- log-bin：在这个例子中我们选black-bin.log
- server- id：在这个例子中我们选 1。只有二进制日志打开的情况下服务器才会作为复制的主机工作。server-id参数必须是 1 到 2 的 32 次方之间的一个正整数。

master.cnf:

```
[mysqld]
server-id=1
log-bin=black-bin.log
datadir=/home/billy/mysql/master/data
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

注意：为了使用事务的InnoDB在复制中最大的持久性和一致性，你应该指定 innodb_flush_log_at_trx_commit=1,sync_binlog=1 选项。

接下来，你需要在从机的slave.cnf文件的【mysqld】部分增加server-id选项。server-id的值类似主机，必须是 1 到 2 的 32 次方之间的一个正整数，而且必须和主机的ID不一样。如果你设置多台从机，那么每台必须有区别于主机和其他从机的唯一的server-id值。可以把server-id值认为是类似IP地址的东西：这些ID在复制服务器通信的时候标识了每台唯一的服务器实例。

通过设置relay-log-index和relay-log，你也可以定义从机使用的文件名。

slave.cnf:

```
[mysqld]
server-id=2
relay-log-index=slave-relay-bin.index
relay-log=slave-relay-bin
datadir=/home/billy/mysql/slave/data
```

现在用服务管理器或者直接从命令行重启MySQL服务器：

```
[billy@black ~]$ mysqladmin -u root shutdown # only needed if MySQL already running
[billy@black ~]$ mysqld --defaults-file=/home/billy/mysql/master/master.cnf &
[billy@blue ~]$ mysqladmin -u root shutdown # only needed if MySQL already running
[billy@blue ~]$ mysqld --defaults-file=/home/billy/mysql/slave/slave.cnf&
```

注意：如果从机之前就在复制，就用-skip-slave-start选项来启动从机，这样从机就不会立刻尝试连接主机。你也可以用 -log-warnings选项来启动从机，这样你就能在错误日志里得到更多的问题消息（例如，网络或者连接问题）。这选项默认是打开的，但除非这个选项的值大于 1，否则中止连接的消息不会记录到错误日志里。

步骤 2：创建一个有复制权限的用户

设置复制的下一步是在主机上创建一个专门用于复制的账号。为了安全起见，我们强烈建议创建一个专门用于复制的用户，这样你就不需要授予这个账号复制以外的权限。在从机能够连接上的主机上创建一个账号。像前文提到的那样，这账号必须被赋予REPLICATION SLAVE权限。你可以在MySQL客户端或者用你爱用的MySQL管理工具来执行GRANT（授权）：

```
[billy@black ~]$ mysql -u root --prompt='master> '
master> CREATE USER repl_user@192.168.0.34;
master> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.34 IDENTIFIED
BY 'billy';
```

步骤 3：锁住主机，注意二进制日志的位置并备份主机数据库

在主机上执行FLUSH TABLES WITH READ LOCK语句来刷新所有的表并阻断写操作：

```
master> FLUSH TABLES WITH READ LOCK;
```

当FLUSH TABLES WITH READ LOCK中的读操作锁生效时，用以下命令在主机上读取当前的二进制日志名称和偏差的值：

```
master> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| black-bin.000001 |   1790   |              |                  |
+------------------+----------+--------------+------------------+
```

“File“列显示了日志的名称，”Position“列显示了日志在文件中的偏差量。在这个例子中，二进制日志文件名是”black-bin.000001“，Position是”1790”。你最好写下这些值，因为后面你设置从机的时候你会用到它们。

它们代表了从机什么时候开始处理新的更新的坐标。

注意：如果之前主机没有打开二进制日志就运行了，用SHOW MASTER STATUS命令显示的日志名称和position值会是空的。如果是这种情况，那么你后面要在从机日志里面指定的名称和position的值应该是空字符串（“”）和 4。

接下来，继续开着你刚才用过的MySQL客户端窗口来执行FLUSH TABLES WITH READ LOCK，你需要把想要复制到从机上的主机数据库里面的数据转存出来。在我们的这个例子中我们把“clusterdb”数据库的内容转存出来。

注意：如果从机有一套和主机上不同的用户账号，你也许不想复制“mysql”系统数据库。这种情况下，你应该在转存的过程中排除“mysql”系统数据库。

你可以在MySQL客户端或者用MySQL Workbench图形化的方式来执行mysqldump。

```
[billy@black ~]$ mysqldump -u root clusterdb >
/home/billy/mysql/master/clusterdb.sql
```

你可以用以下命令在主机上重新打开写操作功能：

```
master> UNLOCK TABLES;
```

步骤 4：在从机上加载 Dump（转存）文件

接下来，你要把主机上的mysqldump文件加载到从机上（当然，这需要从“black”拷贝clusterdb.sql文件到“blue”）。这可以在命令行或者MySQL Workbench的图形化方式来执行。由于在从机上还没有“clusterdb”数据库，因此在装载表和数据之前必须先创建这个数据库：

```
[billy@blue ~]$ mysql -u root  -e  ‘create database clusterdb;’
[billy@blue ~]$ mysql -u root clusterdb <
/home/billy/mysql/slave/clusterdb.sql
```

步骤 5：初始化复制

现在我们准备好在从机上初始化复制了。在从机上执行以下命令：

```
slave> SLAVE STOP;
```

接着，你要输入一条CHANGE MASTER命令：

```
slave> CHANGE MASTER TO MASTER_HOST=’192.168.0.31’,
```

```
-> MASTER_USER='repl_user',
-> MASTER_PASSWORD='billy',
-> MASTER_LOG_FILE='black-bin.000001',
-> MASTER_LOG_POS=1790;
```

这些参数代表：

- **MASTER_HOST**：主机的IP或者名称，这个例子里面填blue或者 192.168.0.31
- **MASTER_USER**：这是我们在步骤 2 中授予REPLICATION SLAVE权限的用户，在这个例子中就是“repl_user”
- **MASTER_PASSWORD**：这是我们在步骤 2 中分配给”repl_user”的密码
- **MASTER_LOG_FILE**：这是我们在步骤 3 中确定的文件名
- **MASTER_LOG_POS**：这是我们在步骤 3 中确定的位置（position）

最后，在从机上开始复制：

```
slave> start slave;
```

步骤 6：基础检查

现在我们准备做一个基础检查来确保复制确实在工作。在这个例子中我们在主机上的“simples”表中插入一行数据，然后验证在从机上有这条新的行数据。

```
master> insert into clusterdb.simples values (999);
slave> select * from clusterdb.simples;
+-----+
| id  |
+-----+
|   1 |
|   2 |
|   3 |
| 999 |
+-----+
```

8 迁移到半同步复制

这个章节假设你已经异步复制成功并运行了，但接着你想迁移到半同步复制 - 换句话说，本演练建立在第 7 章节的基础上。当然如果你没有使用异步复制，也可以直接使用半同步复制。

注意：半同步复制只在MySQL5.5 或更新的版本中可用。

步骤 1：在主机和从机上安装插件

给每台MySQL服务器安装正确的插件：

```
master> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
slave> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

步骤 2：激活半同步复制

要让半同步复制工作，必须在主机和至少一台从机上激活（在我们的例子里只有一台从机）。这可以在master.cnf中设置rpl_semi_sync_master_enabled为“ON”和在slave.cnf中设置rpl_semi_sync_slave_enabled为“ON”来实现；或者可以在MySQL命令行输入指令来实现：

```
master> SET GLOBAL rpl_semi_sync_master_enabled = on;
slave> SET GLOBAL rpl_semi_sync_slave_enabled = on;
slave> STOP SLAVE IO_THREAD; START SLAVE IO_THREAD;
```

如果复制还没有在运行，那么不需要停止并重启从机的IO线程，你只需要输入START SLAVE。

从主机的透视图来检查半同步复制正在工作，并检查在半同步模式下至少连了一台从机：

```
master> SHOW STATUS LIKE 'Rpl_semi_sync_master_status';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| Rpl_semi_sync_master_status |   ON  |
+-----------------------------+-------+

master> SHOW STATUS LIKE 'Rpl_semi_sync_master_clients';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Rpl_semi_sync_master_clients |   1   |
+------------------------------+-------+
```

步骤 3：确认复制是在半同步模式下运行

我们在主机上增加一条新的行然后检查状态参数来确保它被半同步复制了（换句话说就是从机在主机超时前确认收到了变化并且确认异步插入客户端）。

```
master> insert into clusterdb.simples values (100);
master> SHOW STATUS LIKE 'Rpl_semi_sync_master_yes_tx';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Rpl_semi_sync_master_yes_tx |   1   |
+-------------------------------+-------+
slave> select * from clusterdb.simples;
+-----+
| id  |
+-----+
|   1 |
|   2 |
|   3 |
| 100 |
| 999 |
+-----+
```

9 复制管理和故障排除

这一章我们将覆盖如何在MySQL复制上做一些基础的管理和故障排除的任务。

检查复制状态

在管理一个复制进程的时候最常见的任务就是确保复制正在进行并且主机和从机之间没有任何错误。

用来完成这个任务的主要的命令式SHOW SLAVE STATUS，这命令必须在从机上执行。例如：

```
slave> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
Slave_IO_State:
Master_Host: 192.168.0.31
Master_User: repl_user
Master_Port: 3306
Connect_Retry: 60
```

```
Master_Log_File: black-bin.000003
Read_Master_Log_Pos: 790
Relay_Log_File: slave-relay-bin.000013
Relay_Log_Pos: 936
Relay_Master_Log_File: black-bin.000003
Slave_IO_Running: No
Slave_SQL_Running: No
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: 790
Relay_Log_Space: 1238
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: NULL
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
```

下面是一些关于如何解读输出结果的指导：

- Slave_IO_State - 表明从机当前的状态
- Slave_IO_Running - 显示读取主机二进制日志的IO线程是否在运行
- Slave_SQL_Running - 显示执行中继日志中事件的SQL线程是否在运行
- Last_Error - 显示在处理中继日志的时候最后一条记录的错误。理想情况下这个值应该是空的，表示没有错误

- Seconds_Behind_Master – 显示从机SQL线程在处理主机二进制日志时候的秒数。数字大（或者增长快）表明从机不能处理来自于主机的大量的语句。这个值是 0 的话通常可以认为从机能够跟上主机，但严格来说有些情况下也不能这么认为。例如，如果主机和从机间的网络连接丢失了但从机的IO线程没有察觉到 – 也就是slave_net_timeout还没到（但连接已经断了）。

在主机上，你可以通过检查服务器上运行的进程列表来检查从机的状态。

```
master> SHOW PROCESSLIST \G
```

由于是从机在驱动复制进程的内核，因此主机进程列表中获得的信息非常少。

暂停复制

你可以在从机上用STOP SLAVE语句来停止复制，用START SLAVE来开始复制。

用STOP SLAVE来停止从机执行二进制日志：

```
slave> STOP SLAVE;
```

当停止执行时，从机不再通过IO线程从主机读取二进制日志并且不再通过SQL线程处理中继日志中还没执行的事件。你可以指定线程的类型来独立地停止IO或者SQL线程。例如：

```
slave> STOP SLAVE IO_THREAD;
```

如果你想在从机上执行备份或者其他任务，仅仅只处理来自主机的事件，停止SQL线程会是有效的。IO线程会继续从主机读取变化，但这些变化不会马上被应用，这样当你再次开始从机操作的时候从机就能轻易地赶上进度。

停止IO线程会让中继日志里的语句执行到中继日志停止接收新事件的那个点为止。当你想要让从机赶上从主机来的事件时，当你想在从机上做管理但要确保你已经在指定的点有最新的更新时，可用停止IO线程的选项。这种方法同样也能用来停止从机上的事件执行，同时你在主机上做管理确保复制再次启动的时候不会有大量积压的事件要执行。

要再次开始执行复制，用START SLAVE语句：

```
slave> START SLAVE;
```

如果必要，你可以分别独立启动IO线程和SQL线程。

查看二进制日志

如前文所提到的，服务器的二进制日志由包含描述修改数据库内容的“事件”组成。服务器用二进制格式来写这些文件。要用文本格式来显示内容的话，就要用mysqlbinlog功能。你也可以用mysqlbinlog来显示从机上的中继日志文件，因为中继日志的格式和二进制日志的格式相同。

要了解更多关于mysqlbinlog功能的信息，请参见：

http://dev.mysql.com/doc/refman/5.1/en/mysqlbinlog.html

10 故障转移和恢复

有时候主机会故障或者需要关机来维护；这章节描述需要做的步骤。

步骤 1：预备条件

在第 7 和第 8 章节我们用主从配置的形式为两台MySQL服务器配置了复制。在这一章节，我们用两种方式来扩展它：

1. 允许改变关系 - 从机可以变为主机（例如，当我们需要关闭原来的主机来维护的时候），原来的主机随后就变为从机。
2. 我们用图 10 中显示的更复杂的配置 - 一台主机(black)和两台从机(blue和green）。

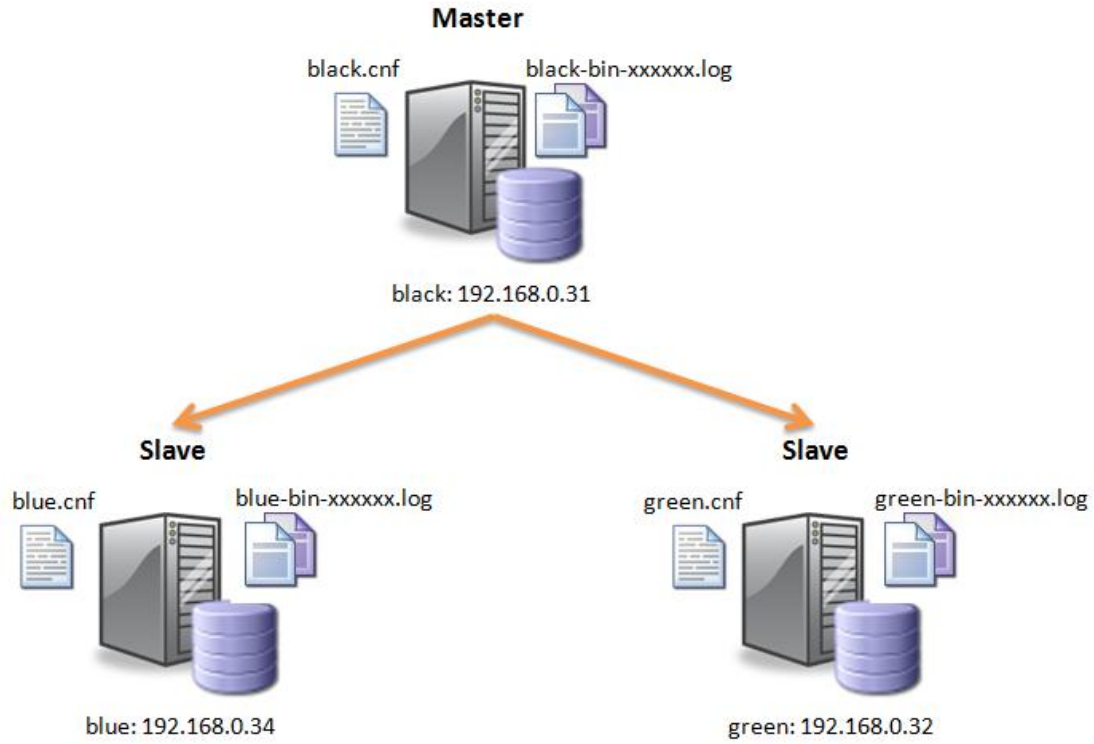


图 10 一主双从

由于这些服务器中的任一台都可以做主机，因此即便是从机也应该记录二进制日志。同样，black也可能变成从机，因此应该有合适的配置数据。配置文件可以参考以下的内容：

```
black.cnf:
  [mysqld]
  datadir=/home/billy/mysql/master/data
  server-id=1
  # Replication Master
  log-bin=black-bin.log
  innodb_flush_log_at_trx_commit=1
  sync_binlog=1
  # Replication Slave
  relay-log-index=slave-relay-bin.index
  relay-log=slave-relay-bin
blue.cnf
  [mysqld]
  datadir=/home/billy/mysql/slave/data
  server-id=2
  # Replication Master
  log-bin=blue-bin.log
  innodb_flush_log_at_trx_commit=1
```

```
  sync_binlog=1
  # Replication Slave
  relay-log-index=slave-relay-bin.index
  relay-log=slave-relay-bin
green.cnf
  [mysqld]
  datadir=/home/billy/mysql/slave/data
  server-id=3

  # Replication Master
  log-bin=green-bin.log
  innodb_flush_log_at_trx_commit=1
  sync_binlog=1
  # Replication Slave
  relay-log-index=slave-relay-bin.index
  relay-log=slave-relay-bin
```

由于任一台服务器都可以作为主机而另外两台作为从机，因此必须在任一台服务器上创建具有连接其他两台服务器权限的复制用户：

```
black> CREATE USER repl_user@192.168.0.34;
black> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.34 IDENTIFIED
BY 'billy';
black> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.32 IDENTIFIED
BY 'billy';
blue> CREATE USER repl_user@192.168.0.31;
blue> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.31 IDENTIFIED
BY 'billy';
blue> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.32 IDENTIFIED
BY 'billy';
green> CREATE USER repl_user@192.168.0.34;
green> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.31 IDENTIFIED
BY 'billy';
green> GRANT REPLICATION SLAVE ON *.* TO repl_user@192.168.0.34
IDENTIFIED BY 'billy';
```

最后，如果正在用半同步复制，那么主机和从机的插件都应该在所有的 3 台服务器上安装，但是主/从的功能应该在正确的服务器上激活：

```
black> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
black> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
black> SET GLOBAL rpl_semi_sync_master_enabled = on;
blue> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
blue> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

```
blue> SET GLOBAL rpl_semi_sync_slave_enabled = on;
blue> STOP SLAVE IO_THREAD; START SLAVE;
green> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
green> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
green> SET GLOBAL rpl_semi_sync_slave_enabled = on;
green> STOP SLAVE IO_THREAD; START SLAVE;
```

步骤 2：检测主机是否故障

很多情况下，主机需要故意下线，比如在执行硬件维护时，这类情况不适用于步骤 2。还有其他的情况，故障非常明显，例如硬件崩溃了，也不适用于步骤 2。然而，也会有许多更细微的‘故障’是你想要检测到的，并且触发故障转移。

一个简单的方法是监控从一台或更多的从机获取的

“Seconds_Behind_Master”值：

```
blue> show slave status \G
*************************** 1. row ***************************
Slave_IO_State: Waiting for master to send event
Master_Host: 192.168.0.31
Master_User: repl_user
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: black-bin.000003
Read_Master_Log_Pos: 599
Relay_Log_File: slave-relay-bin.000013
Relay_Log_Pos: 745

Relay_Master_Log_File: black-bin.000003
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: 599
Relay_Log_Space: 1047
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:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 1
```

如果Seconds_Behind_Master值增加了并且送到主机的更新等级没有明显改变那么这可能是一个信号表明主机由问题（特别是在多台从机都观察到同样的情况）。

如图 15 所示，MySQL企业版监控器能够用来监控复制的状态因此可以用来报警，这对于故障转移是必须的。

步骤 3：停止写到主机

如果主机的MySQL服务器进程死了或者下层的硬件故障了，那么这一步可能是没用的，但如果对于计划中的主机维护或者细微的故障，你会想要在这段转变的时间内防止客户端应用程序因此发生改变。

有许多方法来停止更新 - 通过应用程序或负载均衡或连接器。另一个方法可以在所有表上加一个锁：

```
black> FLUSH TABLES WITH READ LOCK;
```

步骤 4：把从机提升为主机

在我们的例子中有两台从机，因此需要哪台变为主机；这个选择的依据应该根

据哪台从机上的数据更加新 - 换句话说就是哪台从机处理更多的来自原来主机的复制事件；这可以通过在每台从机上运行“show slave status \G”来决定。

一旦选择好，被选为新主机的服务器（在这个例子中选择了“blue”）的IO线程（见图 8）应该停止：

```
blue> STOP SLAVE IO_THREAD;
```

执行完这条语句，从机的SQL线程会继续运行并执行中继日志中剩下的更新。一旦没有足够的时间处理所有中继日志的改变（等到“SHOW SLAVE STATUS \G”输出“Exec_Master_Log_Pos”=“Read_Master_Log_Pos”），在即将成为新主机的服务器上应该停止复制：

```
blue> STOP SLAVE;
```

如果使用半同步复制，那么在这个点激活新的主机上的主机侧的插件：

```
blue> SET GLOBAL rpl_semi_sync_master_enabled = on;
```

在从新的主机（blue）开始复制到剩下的从机（green）之前，你需要决定在新主机二进制日志中当前的position：

```
blue> show master status \G
*************************** 1. row ***************************
File: blue-bin.000001
Position: 807
Binlog_Do_DB:
Binlog_Ignore_DB:
```

在剩下的从机（“green”）上，给它重新指定新的主机：

```
green> stop slave;
green> CHANGE MASTER TO MASTER_HOST='192.168.0.34',
-> MASTER_USER='repl_user',
-> MASTER_PASSWORD='billy',
-> MASTER_LOG_FILE='blue-bin.000001',
-> MASTER_LOG_POS=807;
green> start slave;
```

如果用半同步复制，那么通过在新主机上执行以下查询（结果应该是 1）来确保剩下的从机是以异步复制方式注册的：

```
blue> SHOW STATUS LIKE 'Rpl_semi_sync_master_clients';
```

```
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Rpl_semi_sync_master_clients  |   1   |
+-------------------------------+-------+
```

复制现在又开始运行了，如图 11 所示。

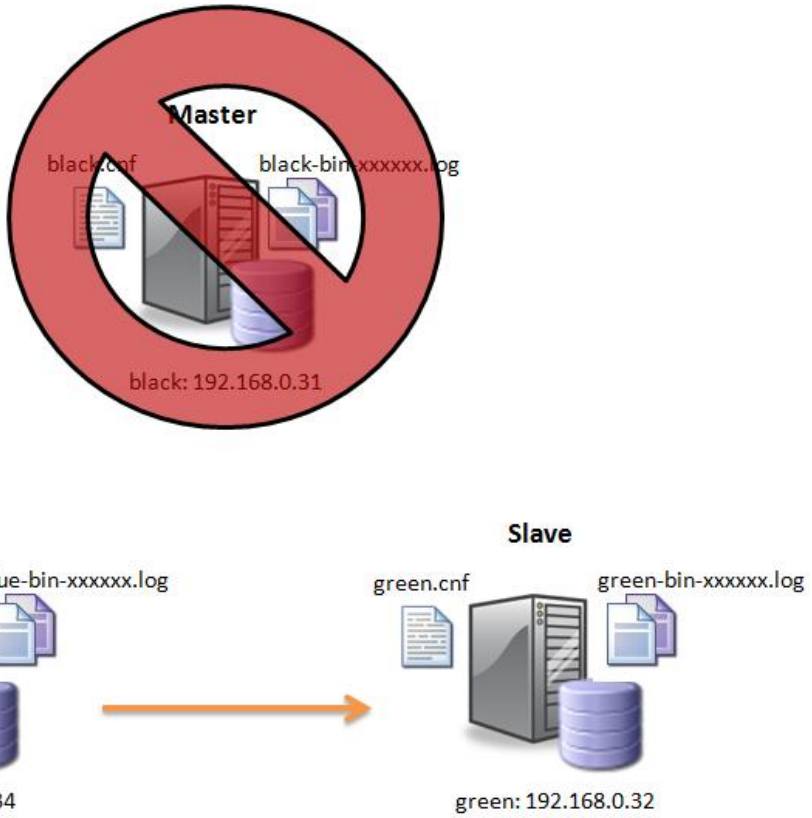


图 11 用新的主机重新启动复制

步骤 5：应用了中继日志后把写操作重定向到新的主机

应用程序现在可以开始把写操作送到新的主机上（读操作在新的主机和剩下的从机上执行）。这可以通过使用连接器、负载均衡或者在应用程序内部实现。

步骤 6：用新的主机同步故障的主机

在某些时候，原来的主机（black）可以配置成为新主机的一台从机。有 2 种选择来达到这个目的：

1. 把black完全作为一台新的从机并按照第 7 章节中的步骤 3 到 6 来做。这是

个很好的选择，如果在新的主机刚引入之前非常大量的更新应用在新的主机上，或者如果有一些原主机上的改变还没有复制到新的主机上（这可能是由于没有使用半同步复制引起的原主机上的故障）。

2. 把原主机作为新主机的一台从机引入原系统并让它赶上进度。这一章剩下的部分展现了这种方法。

用这两种方法之中的任意一种，都要在原主机上先释放读操作的锁：

```
black> UNLOCK TABLES;
```

如果用半同步复制那么在原主机（新从机）上激活插件：

```
black> SET GLOBAL rpl_semi_sync_slave_enabled = on;
```

从步骤 4 中记录的点来启动新的从机：

```
black> CHANGE MASTER TO MASTER_HOST='192.168.0.34',
    -> MASTER_USER='repl_user',
    -> MASTER_PASSWORD='billy',
    -> MASTER_LOG_FILE='blue-bin.000001',
    -> MASTER_LOG_POS=807;
black> start slave;
```

在这个点，我们又回到 1 主 2 从的运行状态，如图 12 所示。

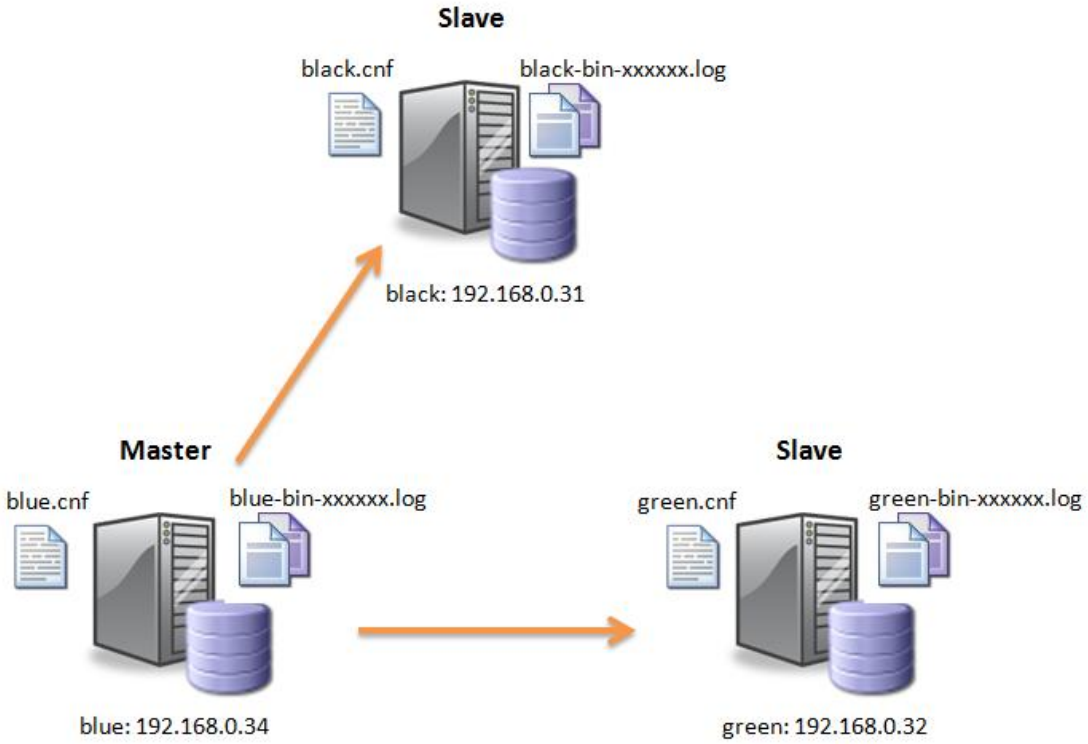


图 12 回到 1 主 2 从

最后一步是可做可不做的，就是通过再做一遍这个流程把原先的主机（black）再变为当前的主机。

11 使用 MySQL CLuster 复制时的差别

MySQL Cluster是可扩展的，高性能的，集群化的数据库，最初是为电信行业的一些世界上最苛刻的应用而开发的。这些电信的应用常常需要数据库的可用性超过 99.999%。

MySQL Cluster可以做为MySQL的一个插入式的存储引擎，但架构完全不同于其他的MySQL存储引擎（例如InnoDB和MyISAM），并且这会影响到复制如何工作和应用。这种架构在下面的图 13 中有展现。对复制的影响而言，关键的架构特点是数据不是存在一个MySQL服务器实例中的；而是分布在许多数据节点。集群内的数据节点之间有同步复制来提供高可用性 - 注意这个同步复制和本白皮书描述的复制没有关系。数据可以直接从数据节点被访问（例如使用本身的C++ API）或者可以用一台或者更多的MySQL服务器来提供SQL访问。每台MySQL服务器都可以读或写任何表并且改变可以立即被其他MySQL服务器观察到。

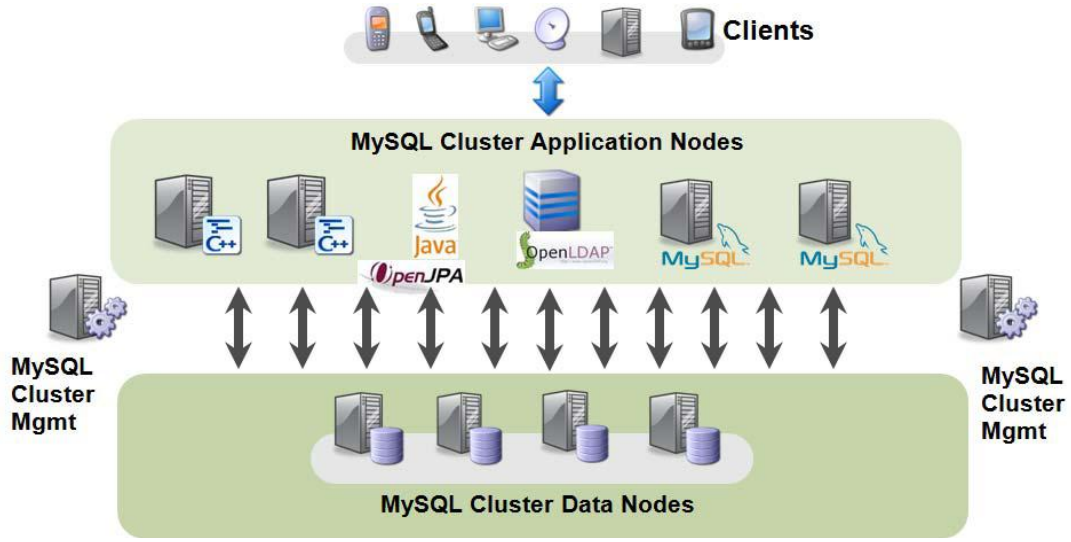


图 13 MySQL Cluster架构

通常用MySQL Cluster复制来提供地理上的冗余 - 内部的（同步的）复制在数据中心协同工作的数据节点之间提供高可用性，然后MySQL（异步的）复制到远程站点防止灾难性的站点故障。

这是如何影响MySQL复制的呢？改变可以来自于任何MySQL服务器或者甚至直接来自于数据节点，因此应用了一个机制，让所有变化集中到一台或更多的MySQL服务器的二进制日志中然后这些服务器作为MySQL复制主机运行。这是由NDB Binlog Injector线程执行的。这个线程确保所有的变化（不管它们在集群中的那个地方应用）以正确的顺序写到二进制日志中。

如图 14 所示，这提供了更健壮的复制系统架构的机会。图示的两台MySQL服务器，每台服务器的二进制日志都包含了相同的变化，其中任一台服务器都可以用作主机把变化复制到MySQL Cluster部署的一台或更多从机上或者复制到使用InnoDB/MyISAM存储引擎的MySQL服务器上。虽然每个主二进制日志应该包含相同的变化，但它们是相互独立的；以方便故障时从主机转移到另一个，集群范围内的’Epoch’用来表示在一个时间点集群的状态。这些高级的故障转移技术超出了本白皮书的范围，更多的资讯可以在MySQL Cluster参考指南中找到：http://dev.mysql.com/doc/mysql-clusterexcerpt/5.1/en/mysql-cluster-replication-failover.html

MySQL Cluster支持正在运行的MySQL以多主激活—激活的配置复制或复制环方式复制。此外，MySQL Cluster可以提供冲突检测/解决来涵盖改变冲突的情况（在这些不同的主机上写同样的行）。要让这个冲突检测/解决功能工作，需要做一些额外的事情 - 这超出了本白皮书的范围，你可以在MySQL Cluster参考指南中找到详细的内容：

http://dev.mysql.com/doc/mysql-cluster-excerpt/5.1/en/mysql-cluster-replication-multimaster.html

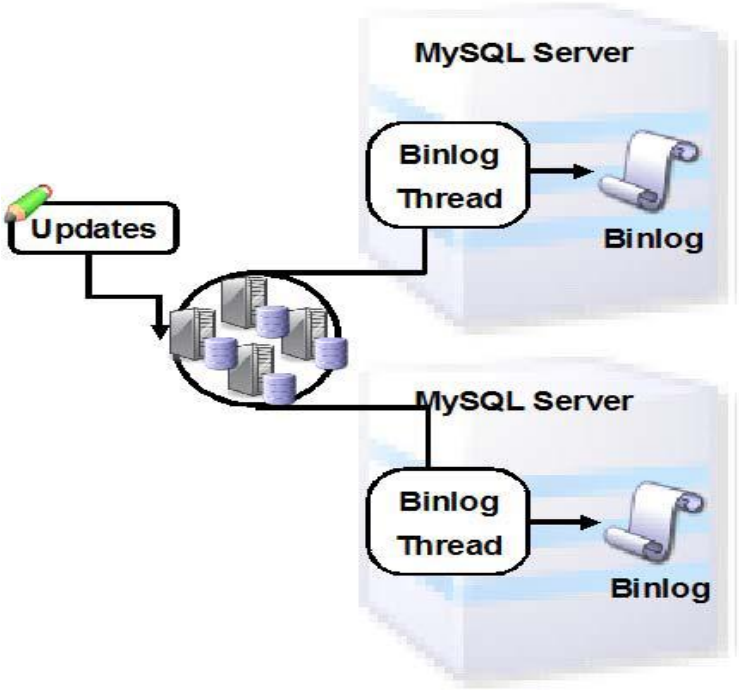


图 14 集群中的多主

MySQL Cluster的版本和主要的MySQL版本发布时间表不同，所以最新的MySQL复制功能不总是能在MySQL Cluster版本中找到。例如，在写最新的MySQL Cluster7.1 的GA版本的时候用的是修改过的MySQL5.1 版本的数据库，所以在MySQL5.5 中引入的提升并没有体现。一个例外是在MySQL Cluster7.1.3 中引入的属性升级和降级。

当从MySQL Cluster复制的时候，通常使用基于行的复制（而不是基于语句的复制）。

12. 用 MySQL 企业版监控器来监控复制

带查询分析器的MySQL企业版监控器是一个分布式的web应用，你可以在你公司的防火墙内安全地部署。监控器持续监控你所有的MySQL服务器并提前警告你潜在的问题及调优方法，避免潜在的问题造成重大的损失。它也给你提供它发现的问题的MySQL专家建议，这样你就知道如何花时间来优化你的MySQL系统了。

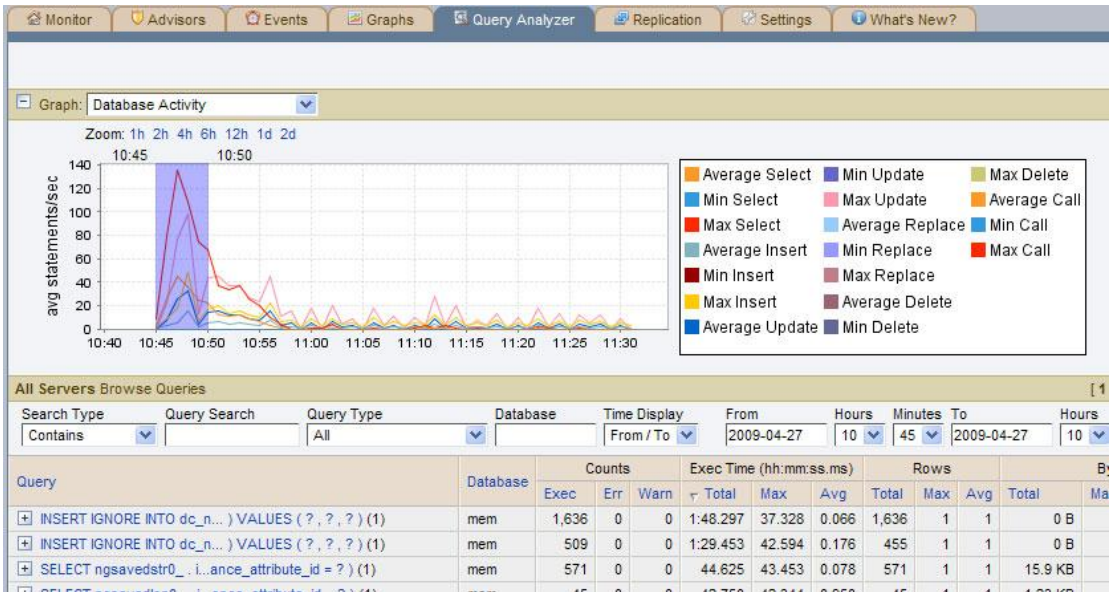


图 15 MySQL企业版监控器

企业版监控器提供MySQL复制特定功能，包括：

自动检测复制关系，自动分组并维护，不需要DBA设置或配置

一个综合的，实时的视图看到所有主/从数据库的性能和状态 - 如下图 16 所示。

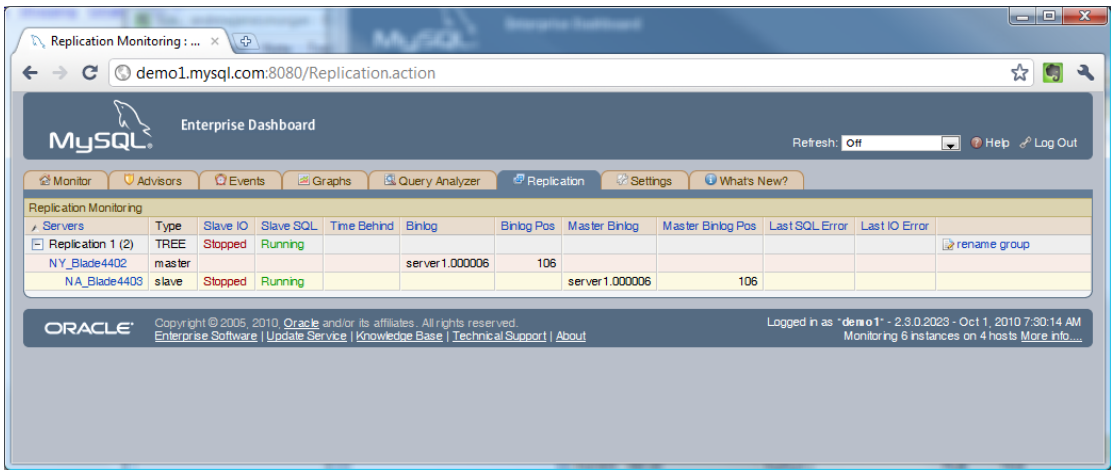


图 16 MySQL企业版监控器中的复制统计

MySQL企业版监控器对MySQL客户来说可以作为一个商业的选择来获得。

13. 结论

MySQL复制已经被证明是一种有效的解决方案，用在一些对环境要求最苛刻的

web网站和企业应用中，以极致的数据库扩展能力来驱动这些应用。

本白皮书讨论了部署MySQL复制的商务和技术上的好处，并提供了实践的一步步的指导如何开始复制。

正如已经被MySQL5.5 版本所证明的那样，复制是一个活跃的发展领域，不断地在数据完整性、性能和部署灵活性等各领域有改进。

以下资源补充本白皮书中所提到的材料，并可以加速你评估和部署MySQL复制。

14. 资源

MySQL 5.5 下载: http://dev.mysql.com/downloads/mysql/5.5.html

MySQL复制用户指南:
http://dev.mysql.com/doc/refman/5.5/en/replication.html

MySQL5.5 复制的提升提供可扩展性和高可用性 - 网络研讨会重播:
http://www.mysql.com/news-and-events/on-demand-webinars/display-od-572.html

MySQL Cluster 地理复制网络研讨会重播:
http://www.mysql.com/news-and-events/ondemand-webinars/display-od-415.html

MySQL Cluster 复制文档:
http://dev.mysql.com/doc/mysql-clusterexcerpt/5.1/en/mysql-cluster-replication.html

MySQL 在Ticketmaster的应用 (MySQL复制的重要用户):

http://www.mysql.com/customers/view/?id=684

请直接联系甲骨文（免费电话）

中国：800 810 0161
香港：800 901 039
台湾：0800 672 253