**MySQL集群**

**技术文档**

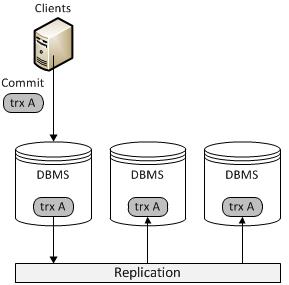
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本编号** | | V0.8 | | |
| **文档状态** | | [√] 草稿；[ ] 审核；[ ] 发布 | | |
| **起草人** | |  | | |
| **审核人** | |  | | |
| **批准人** | |  | | |
| **最后更新时间** | |  | | |
| **关键词** | |  | | |
| **总页数** | |  | | |
| **版本更新摘要** | | | | |
| **版本号** | **日期** | | **审阅人** | **更新摘要** |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |

# 数据复制技术简介

数据库复制是指频繁的将数据从一个节点（服务器上的一个数据库）复制到另一个节点，可以将数据库复制系统看作一个分布式数据库，但是其中所有节点共享相同的信息。这类系统也称为数据库集群。

## Master-Slave/Multi-Master复制

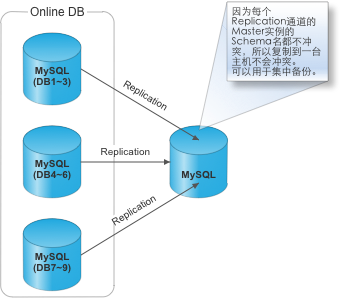
### Master-Slave复制



在Master-Slave系统中，Master节点记录数据修改日志，并将日志通过网络传送到Slave节点。Slave节点从Master节点接收修改操作流，并重放修改。

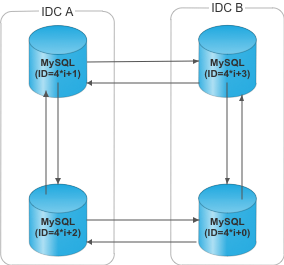
### Multi-Master复制

在Multi-Master复制系统中，能够将修改提交到任意一个数据库节点(主要是环形复制)，这些修改通过网络传送到其他数据库节点，所有节点都作为Master使用。



这是跨机房的HA。为了容灾或者加速，很多公司都采用在不同机房部署数据库的方式，所以就涉及到数据同步。为了保证每个机房产生的数据不冲突，一般来说我们采用的是auto\_increment\_increment，auto\_increment\_offset这两个参数，可以控制步进。例如双MAster，我们会配置主库是奇数序列的ID，备库是偶数序列的ID，这样切换时就算有少量binlog还未应用，也不会导致数据冲突。跨机房以后，例如两个机房都有双Master，两个机房之间数据又需要同步，以前需要借助第三方脚本或者程序，有了多Master，按如下方式搭建，设置步进为4，就可以保证每个机房有双MAster HA，机房之间数据又可以同步。

这是一备多的备份。因为我们采用的分库策略，使我们一个集群会有很多个实例，每个实例里面有几个Schema，但是肯定不会重复。例如第一个实例是1～3号Schema。第二个实例就是4～6号Schema，所以binlog应用到一起并不会冲突数据。



## Asynchronous/Synchronous/并行复制

### Asynchronous复制

异步复制采用延时（lazy）复制策略，Master数据库异步方式传送修改复本到其他节点。在日志记录（或）日志传送后，Master节点提交事务。当事务提交后，至少短暂时间内，部分节点拥有不同数据。

### Asynchronous复制缺点

slave会对master有一个更新延迟，当master宕机，slave被提升为新的master时，必然会发生数据丢失。

### Synchronous复制

同步复制采用即时（eager）复制策略，通过在一个事务中完成所有复本修改保持节点复本同步。当事务提交后，所有节点拥有相同数据。

理论上来讲，同步复制具有如下优点：（1）高可用，节点崩溃时没有数据丢失、数据复本一致、不需要复杂耗时的故障切换；（2）提升性能，可以在所有节点上并行执行事务；（3）跨集群因果关系，例如，事务执行后发送的SELECT query总能够看到事务效果，即使事务是在其他节点上执行的。

传统的即时复制协议一次协调一个操作，并使用两阶段提交或分布式锁。假设系统中n个节点o个操作每秒t个事务吞吐量，每秒消息数是：



这就意味着，每增加节点，都会导致事务响应时间和冲突与死锁频率指数级增长。

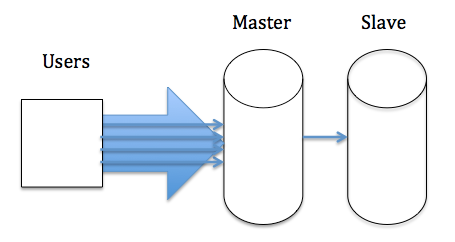
### Synchronous复制缺点

性能低下

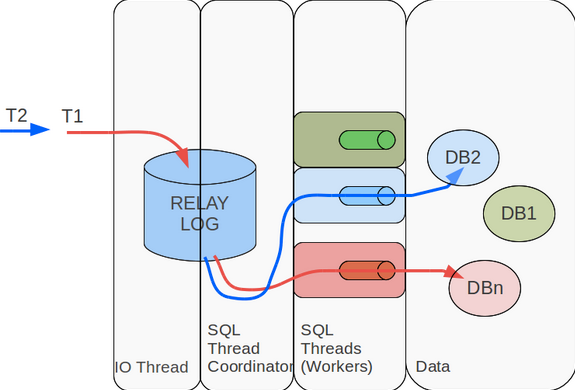
原因：当主库执行完一个事务，所有的从库都执行了该事务才返回给客户端。因为需要等待所有从库执行完该事务才能返回，所以全同步复制的性能必然会收到严重的影响。

### 并行复制

半同步复制解决了Master-Slave的强一致问题，那么性能问题呢？可以看到参与复制的主要有两个线程：IO线程和SQL线程，分别用于拉取和回放binlog。对于Slave而言，所有拉取和解析binlog的动作都是串行的，相对于Master并发处理用户请求，在高负载下， 若Master产生binlog的速度超过Slave消费binlog的速度，导致Slave出现延迟。如图4，可以看到，Users和Master之间的管道远远大于Master和Slave之间的管道。



那么如何并行化，并行IO线程，还是并行SQL线程？其实两方面都可以并行，但是并行SQL线程的收益更大，因为SQL线程做的事情更多(解析，执行)。并行IO线程，可以将从Master拉取和写Relay log分为两个线程；并行SQL线程则可以根据需要做到库级并行，表级并行，事务级并行。库级并行在mysql官方版本5.6已经实现。如图，并行复制框架实际包含了一个协调线程和若干个工作线程，协调线程负责分发和解决冲突，工作线程只负责执行。图中，DB1，DB2和DB3的事务就可以并发执行，提高了复制的性能。有时候库级并发可能不够，需要做表级并发，或更细粒度的事务级并发。



并行复制如何处理冲突？并发的世界是美好的，但不能乱并发，否则数据就乱了。Master上面通过锁机制来保证并发的事务有序进行，那么并行复制呢？Slave必需保证回放的顺序与Master上事务执行顺序一致，因此只要做到顺序读取binlog，将不冲突的事务并发执行即可。对于库级并发而言，协调线程要保证执行同一个库的事务放在一个工作线程串行执行；对于表级并发而言，协调线程要保证同一个表的事务串行执行；对于事务级而言，则是保证操作同一行的事务串行执行。

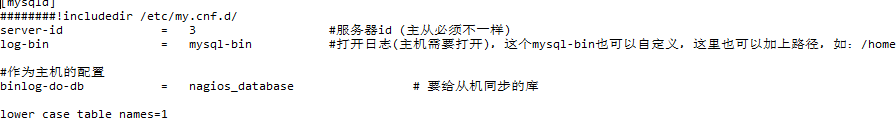
是否粒度越细，性能越好？这个并不是一定的。相对于串行复制而言，并行复制多了一个协调线程。协调线程一个重要作用是解决冲突，粒度越细的并发，可能会有更多的冲突，最终可能也是串行执行的，但消耗了大量的冲突检测代价。

### 同步复制改进方案

## MySQL同步粒度及引擎

### MySQL同步粒度

可通过配置文件my.cnf配置,如下



参数binlog-do-db=要给从节点同步的库，如果不配置此参数，整个数据库都将进行同步，如果之前设置了要同步的库，然后将此参数删除，同步的库包括（两个数据库中库名表名字段名及数据内容相同的库将会进行同步）。

### MySQL同步支持引擎

经过测试mysql主从同步支持引擎有（innodb，myisam，黑洞引擎，isam 等数据库引擎），且数据库引擎还可交叉使用，(如主节点使用innodb，从节点使用myisam，黑洞引擎等)

# MySQL数据库复制方案

## MySQL Replication

MySQL Replication是MSQL Server提供的功能，允许从一个MSQL数据库服务器（称为master）复制到一个或多个MySQL数据库服务器（称为slaves）。MySQL Replication的采用异步复制机制，复制过程不是即时的，不能确保slaves和master内容相同。

MySQL Replication使用灵活，根据配置，可以复制所有数据库、选中的数据库，甚至是一个数据中选中的表。

在MySQL Replication中，相关服务器具有下面两个角色中的一个：（1）Master：Master MySQL服务器将所有修改数据的事务写入到binary log中；（2）Slave：Slave MySQL服务器启动时连接master，从master的binary log中下载事务，然后将本地服务器重放（apply）事务。

需要注意的是，Slaves本身也可以作为masters，Slaves从其master重放的事务可以再写入到其自身的binary log中，能够像其直接生成的事务那样尤其slaves重放。

### Replication优点

****1）扩展：****将负载分布在多个slaves上以提高性能，所有的writes以及事务中的read操作都将有master处理，其他reads将转发给slaves；对于“读写比”较高的应用，replication可以通过增加slaves节点来提高并发能力；因为write只能在master上提交，因此架构扩展对提升write并发能力并不明显，对于writes密集性应用我们应该考虑其他架构。

****2）数据安全：****slave可以中断自己的replication进程，这不会打断master上的数据请求，所以可以在slave上运行backup服务，定期全量backup是保护数据的手段之一。（如果在master上执行backup，需要让master处于readonly状态，这也意味这所有的write请求需要阻塞）。

****3）分析：****数据在master上创建，那么数据分析可以在slave上进行，这将不会影响master的性能。利用mysql做数据分析（或者数据分析平台的源数据），通常都是将某个slave作为数据输入端。

****4）远距数据分布：****如果master的物理位置较远，你可以在临近需求的地方部署slaves，以便就近使用数据，而不需要总是访问远端的master，这在数据分析、数据备份与容灾等方面有很大帮助。

### MySQL Replication架构方式

#### Master-Slave

单个Master和一个或多个Slaves是最简单的架构方式。这种架构方式中，Master和Slave中数据可能不一致，并且不同的Slaves的数据可能略有差异，例如当Slaves通过不同网络连接方式接入到Master时，本地连接的Slave数据可能更新，因为Internet传输延迟会减缓数据复制过程。

Master-Slave架构具有如下优点：（1）不同的MySQL服务器上可以运行稍有差异的结构（如不同的索引），这样可以将少量开销很大的queries在指定的Slave上执行，以免拖慢Master；（2）配置管理简单。

Master-Slave架构具有如下不足：（1）没有自动冗余，在Master故障时，Slaves可能数据与Master不一致，不能提升为Master，这在Slave部署在低配设备上时尤其常见；（2）Write query不能在Slave上执行；（3）复制与RAID1相同，磁盘空间使用率不高；（4）在下载binary log时，每个Slave都增加一些Master负载，因此Slaves个数不能无限制增加。

#### Master-Master

Master-Master架构方式包含两个MySQL Server，两个都配置为Master和Slave，一个服务器执行的事务能够被另一个获得，反之亦然。

这种架构方式具有很致命的问题：（1）相互冲突的query在两个节点上同时执行导致的某些query的不确定性和竞争条件，很容易导致数据的不一致性；（2）一旦存在不一致，还很难发现，直到严重到由于复制query不能执行导致复制断开。

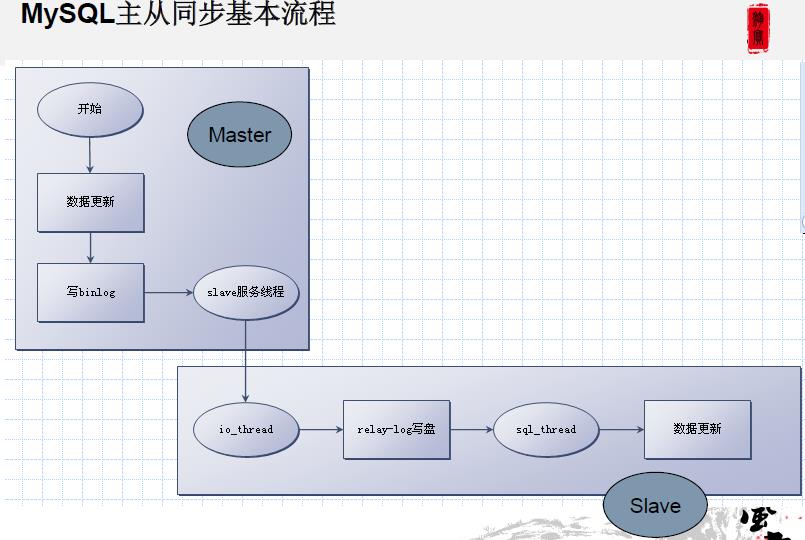
由于存在致命的一致性问题，这种方案不宜使用，如果确实需要多个活体，考虑使用其他高可用解决方案。

#### Active/Passive Master

Active/Passive Master架构方式包含两个MySQL服务器，配置为Master-Master复制，但是在一个Passive节点外设置了某种形式的“write barrier”，以确保在任意时间点只能有一个节点能够执行修改数据的query。

### MySQL Replication工作机制

#### MySQL主从同步流程



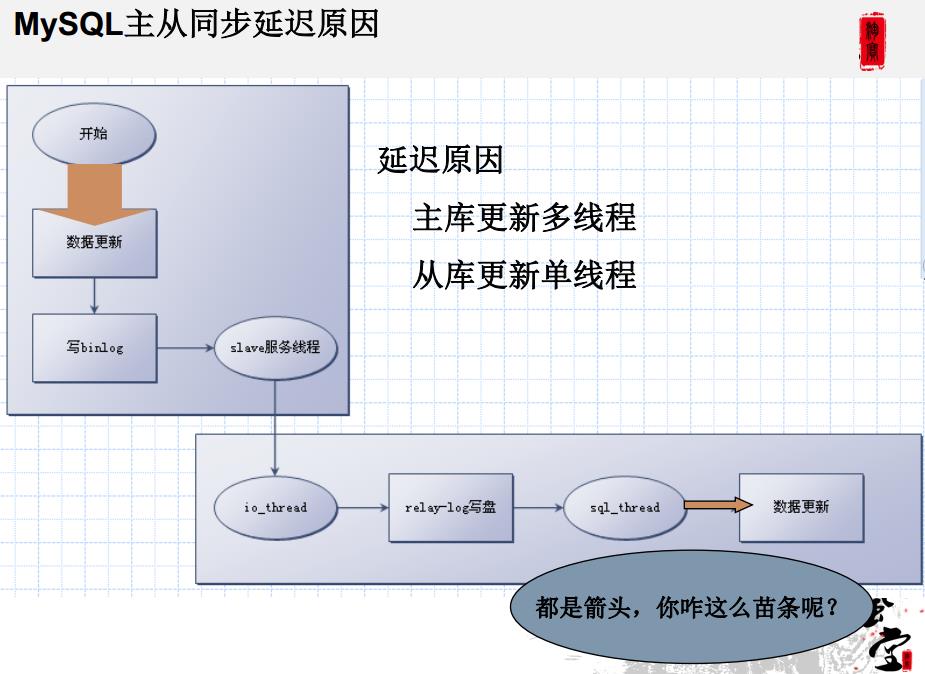
Slave 上面的IO线程连接上 Master，并请求从指定日志文件的指定位置(或者从最开始的日志)之后的日志内容；

Master 接收到来自 Slave 的 IO 线程的请求后，通过复制的 IO 线程根据请求信息读取指定日志指定位置之后的日志信息，返回给 Slave 端的 IO 线程。返回信息中除了日志所包含的信息之外，还包括本次返回的信息在 Master 端的 Binary Log 文件的名称以及在 Binary Log 中的位置；

Slave 的 IO 线程接收到信息后，将接收到的日志内容依次写入到 Slave 端的Relay Log文件(mysql-relay-bin.xxxxxx)的最末端，并将读取到的Master端的bin-log的文件名和位置记录到master- info文件中，以便在下一次读取的时候能够清楚的高速Master“我需要从某个bin-log的哪个位置开始往后的日志内容，请发给我”；

Slave 的 SQL 线程检测到 Relay Log 中新增加了内容后，会马上解析该 Log 文件中的内容成为在 Master 端真实执行时候的那些可执行的 Query 语句，并在自身执行这些 Query。这样，实际上就是在 Master 端和 Slave 端执行了同样的 Query，所以两端的数据是完全一样的。

#### MySQL主从同步延迟原因

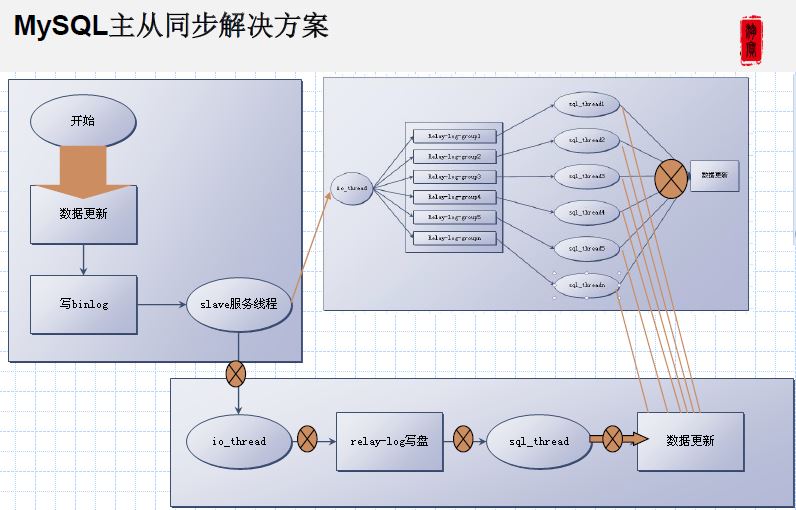


从上面来看，主从同步延迟的主要原因是因为主库采用多线程更新，而从库采用单线程更新。看起来解决问题很简单，只要从库也采用多线程更新即可。

但这样做会带来一个新问题，当从relay\_log读到对同一个条记录的多条操作时（比如先后insert和update 了同一个记录），此时如果这两个操作分配给了2个sql 线程并发去更新从库，那就不能保证它们的执行顺序了。

解决这个问题的方法是，当有多个sql\_thread线程来更新从库的时候，必须让每个sql\_thread对应一张表，如果从库有10张表，那么就使用10个sql\_thread分别更新，这样就解决了并发带引起的乱序的问题；

#### MySQL主从同步解决方案



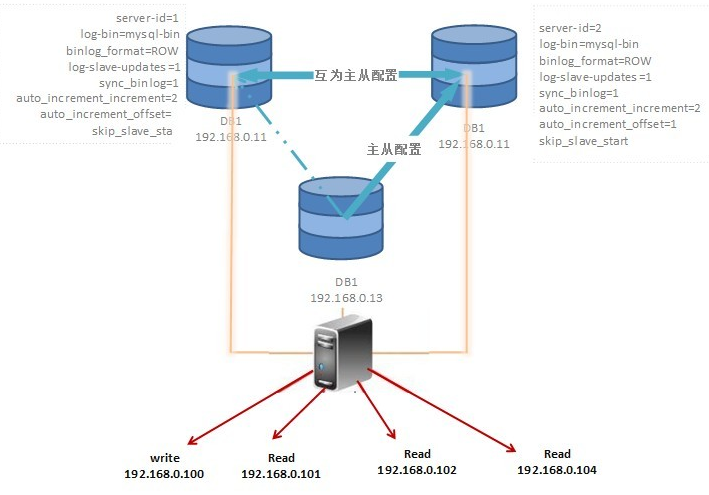
改进的主从同步方案：

1、在io\_thread接收主库日之后，分成N份relay-log存放；  
2、再用N个sql\_thread分别读取日志分发；  
3、确保同一个表的更新语句顺序与主库binlog相同

## Multi Master Replication Manager（MMM）

MMM（Multi Master Replication Manager）是开源的Perl脚本集，用于Active/Passive Master架构方式的自动化创建和管理。

Active/Passive底层的MySQL Replication复制机制是异步的，当Master故障时可能导致部分事务丢失，Multi Master Replication Manager并不能处理这种情况。实际上，如果这种事务丢失不可接受，任何基于异步复制的高可用技术都不适用。

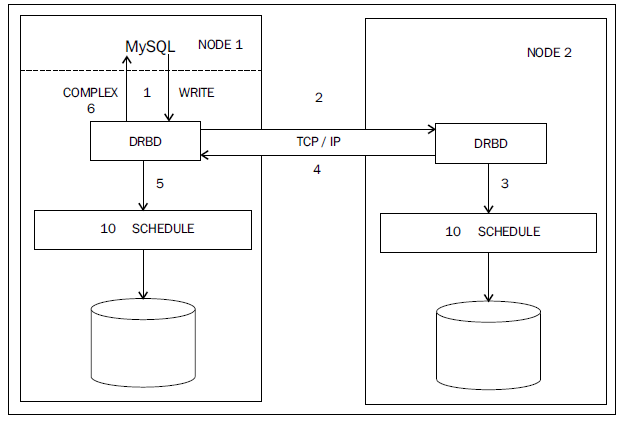


## Distributed Replicated Block Device（DRBD）

DRBD （Distributed Replicated Block Device）是软件实现的监控快设备的（如硬盘、分区、逻辑卷等）无共享服务器间复制存储方案。

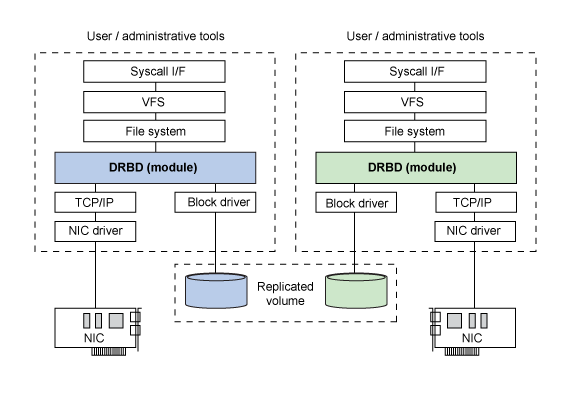
DRBD是领先的Block级复制开源软件。Block级复制通过机器间硬盘驱动（Block）级数据复制实现高可用数据库。换句话说，在两个机器上，每次main Server内核上的Write操作，都被发送到另一个Server上，写入到其硬盘。

DRBD通过在集群相关Linux机器上安装一个内核模块，一旦加载，该内核模块在磁盘驱动调度执行写入前捕获IO Write操作，并通过TCP/IP发送到复本Server上，复本Server将Write发送到其本地磁盘。在此过程中的某个阶段，第一个节点像其磁盘发送Write，并向MySQL报告Write执行完成。这里需要一致性和性能的折中考虑，由一个配置参数可以指定在节点接收到Write的处理过程中的哪个点通知应用Write执行完成。为了最大持久化，通常是在Write在对等节点磁盘上执行之后才通知应用，这种称为同步模式（synchronous mode），执行过程如图所示。



DRBD默认不支持同时写入到多个节点上，如果想要同时写入到多个节点上，需要使用集群感知文件系统（如GFS）。

#### DRBD 架构



### DRBD 主要功能

尽管复制磁盘的思想从概念上来说是很简单的（开发也相对比较容易），但是一个健壮的实现也有很多固有的复杂性。例如，向一个网络驱动器复制块相对比较简单，但是，处理故障和暂时断电（以及随后的驱动器同步）才是真正解决方案的开始。本节将介绍 DRBD 提供的主要功能，包括各种 DRBD 支持的故障模型。

#### 复制模式

本文前面探究了节点之间复制数据的各种方法（尤其是这两种 — 完全同步和完全异步）。DRBD 支持每种方法的变体，这些方法比异步模式提供更多的数据保护，代价是性能略有下降。内存异步模式（或半异步模式） 是介于同步模式和异步模式之间的一个变体。在这种模式下，写操作是在数据存储到本地磁盘并镜像到对等节点内存后被确认的。该模式提供更多保护，因为数据被镜像到另一个节点，这仅仅是针对易失性内存，而不是非易失性磁盘。这仍然可能丢失数据（例如，如果两个节点都发生故障），但是主节点故障不会引起数据丢失，因为数据已经被复制了。

#### 联机设备验证

DRBD 允许对本地和远程对等设备进行联网验证（在输入/输出发生的同时）。验证意味着 DRBD 可以核实本地和远程磁盘是否是相互间的副本，这是一个耗时的操作。但是相比在节点之间移动数据进行验证，DRBD 提供了一个更为高效的方法。为了保护节点间的带宽（可能是一个受限资源），DRBD 不需要在节点间移动数据进行验证，而是移动数据（hash）的加密摘要。这样一来，一个节点可以计算一个块的散列值；将较小的签名转移到对等节点，该节点也可以计算散列值，然后对两者进行比较。如果散列值是相同的，数据块已经被正确的复制了。如果散列值不相同，将过期的数据块标记为不同步，随后的同步确保数据块是正确同步的。

#### 通信完整性

节点之间的通信可能会将错误引入复制数据（由于软件或防火墙漏洞，或者不能被 TCP/IP 的校验码检测出来的其他错误）。为了提供数据的完整性，DRBD 计算消息完整性代码，以随数据一起在节点之间移动。这支持接收节点验证输入数据，并在发现一个错误时请求重发数据。DRBD 使用 Linux 加密应用程序编程接口，因此在完整性算法使用方面是比较灵活的。

#### 自动恢复

DRBD 可以从多种错误中恢复，但是一个最严重的错误就是所谓的 “裂脑（split-brain）”。在这个错误场景中，节点之间的通信链发生故障，每个节点都认为自己是主节点。而对于主节点而言，每个节点支持写操作，而不会将这些操作传播到对等节点。这导致每个节点中的存储不一致。

大多数情况下，脑裂恢复是人工进行的，但是 DRBD 提供几个主动恢复这一状况的操作方法，所用恢复算法具体取决于实际的存储方式。

发生裂脑之后，同步存储最简单的方法是在链接出现故障时其中一个节点没有发现改变。这样，已经发生改变的节点与潜在对等节点进行简单的同步。另一个方法是丢弃变更较少的那个节点中的更改。这使得最大变更集合的节点可以继续工作，但也意味着一个主机上的变更将丢失。

还有两个方法是根据节点当时的状态丢弃更改。一种方法是将最后一次转换成主节点的那个节点中的变更丢弃，而另一种是将最老的主节点（是第一次转换成主节点的节点）中的变更丢弃。您可以在 DRBD 配置文件中操作每个节点，但是它们最终的使用取决于使用存储的应用程序以及数据是否有必要丢弃或进行人工恢复。

#### 优化同步

复制存储设备的一个关键因素是节点之间数据同步的方法是否高效。DRBD 使用的模式其中两个是活动日志和快速同步位图。活动日志存储最近写入的块并确定故障恢复后应同步哪些块。快速同步位图确定连接断开时同步（或不同步）的数据块。节点重新连接之后，可使用这些位图快速同步节点，彼此之间进行精确的复制。时间是很重要的，因为它代表辅助磁盘不一致时的窗口。

## MySQL Fabirc

 Farbic由replication基础特性和一些扩展框架构建而成，用于管理MySQL Servers Farms，与基本的replicaiton相比，它实现了2个核心的特性：HA和sharding。Fabric集群中任何时候只有一个Primary（即master），其他的实例为Secondaries（即slaves）；通过使用replication，将数据在多个节点上备份，HA总是保持集群的数据可用性。如下特性是replication所不具备的：

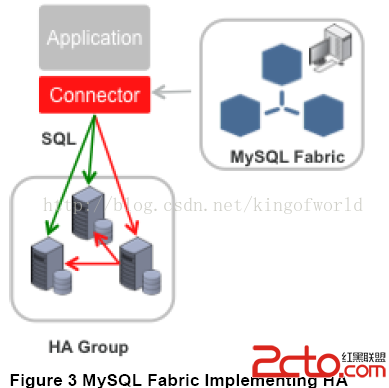
    1）故障检测和角色迁移（Failover）：Fabric进程用于监控集群中的所有节点，如果发现primary失效，稍后他将从Secondaries中选择一个“数据最新”的节点，并提升为primary；此后其他的secondaries将从新的priamry上同步数据变更操作。Connectors（比如Connector/J客户端）发现primary故障时也会通知Fabirc，那么Fabric将通知信息作为决策的一部分来判定priamry的状态。这个特性简称为“自动Failover”，是replication架构中必备的手段之一。

    2）数据库请求路由（Router）：当Fabric提升一个新的primary后，它将会更新state store（存储replication的节点状态等），此后Connectors将会获取新的state数据并在客户端本地cache。因此，application不需要时刻关注（aware）集群中servers拓扑结构的变化，只需要根据state cache中的server状态，将writes发送给相应的primary即可。这种特性有Connectors客户端与与Fabric共同实现，在普通的replication架构中，客户端自动角色路由是无法完成的。如果cache中拓扑不是最新的，application的操作异常将会反馈给Fabirc，参考1）。

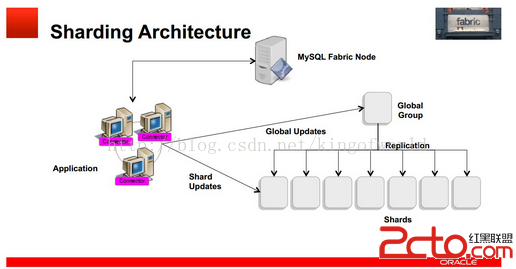
Fabirc支持sharding，对较大规模的数据可以非常便捷的在集群中分布而无需太多人工干预，我们可以简单的认为Fabric是replication模式的完善，支持自动Failover。对于互联网应用，Fabric架构简单而且有效，是首选方案。

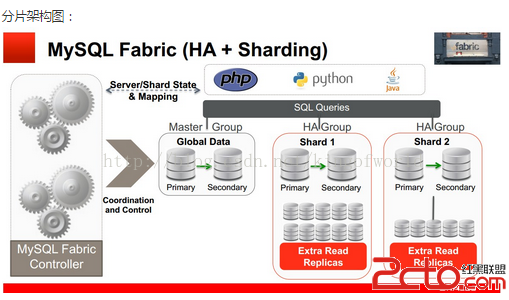
### MySQL Fabirc架构及实现原理

Fabric使用HA组实现高可用性，其中一台是主服务器，其他是备份服务器， 备份服务器通过同步复制实现数据冗余。应用程序使用特定的驱动，连接到Fabric 的Connector组件，当主服务器发生故障后，Connector自动升级其中一个备份服务器为主服务器，应用程序无需修改。



### Fabric支持可扩展性及负载均衡的架构





http://www.2cto.com/database/201504/387166.html

http://www.2cto.com/database/201408/327941.html

## MySQL Cluster

### **MySQL Cluster简**介：

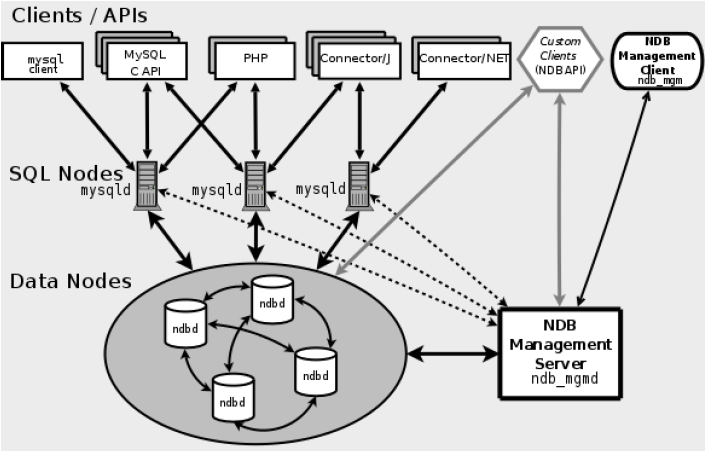
MySQL Cluster 是MySQL 官方集群部署方案，它的历史较久。支持通过自动分片支持读写扩展，通过实时备份冗余数据，是可用性最高的方案，声称可做到99.999%的可用性。

### 架构及实现原理

MySQL Cluster是无共享内存数据库集群，集成标准MySQL服务器与NDB（NetWork DataBase）内存集群存储引擎。采用无共享架构，可以使用低廉的硬件，且对硬件和软件需求具体要求很小。

MySQL Cluster设计方式不存在单点故障。无共享架构方式的系统中每个组件都有独立的内存和磁盘，并且不推荐使用共享存储机制，如网络共享、网络文件系统、SAN等。

MySQL Cluster由成为主机的计算机集组成，每个主机上运行一个或多个进程。这些进程称为节点，包括MySQL Server（用来访问NDB数据）、数据节点（用来存储数据）、一个或多个管理服务器，并且可能还有特殊数据访问程序，组件间关系如图所示：



### MySQL Cluster缺点及限制

对需要进行分片的表需要修改引擎Innodb为NDB，不需要分片的可以不修改。NDB的事务隔离级别只支持Read Committed，即一个事务在提交前，查询不到在事务内所做的修改；而Innodb支持所有的事务隔离级别，默认使用Repeatable Read，不存在这个问题。外键支持：虽然最新的Cluster版本已经支持外键，但性能有问题（因为外键所关联的记录可能在别的分片节点中），所以建议去掉所有外键。Data Node节点数据会被尽量放在内存中，对内存要求大。

数据库系统提供了四种事务隔离级别：  
A.Serializable（串行化）：一个事务在执行过程中完全看不到其他事务对数据库所做的更新（事务执行的时候不允许别的事务并发执行。事务串行化执行，事务只能一个接着一个地执行，而不能并发执行。）。  
B.Repeatable Read（可重复读）：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，但是不能看到其他其他事务对已有记录的更新。  
C.Read Commited（读已提交数据）：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，而且能看到其他事务已经提交的对已有记录的更新。  
D.Read Uncommitted（读未提交数据）：一个事务在执行过程中可以看到其他事务没有提交的新插入的记录，而且能看到其他事务没有提交的对已有记录的更新。

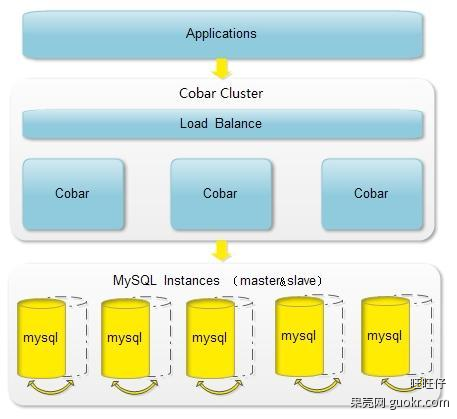
* **管理节点（****ManagementNode）**
* **数据节点（Data Node）**
* **SQL节点（SQL Node）**
* **标准MySQL客户端（Standard MySQL Clients）**
* **NDB客户端程序（NDB Client Programs）**

**管理客户端（Management Clients）**

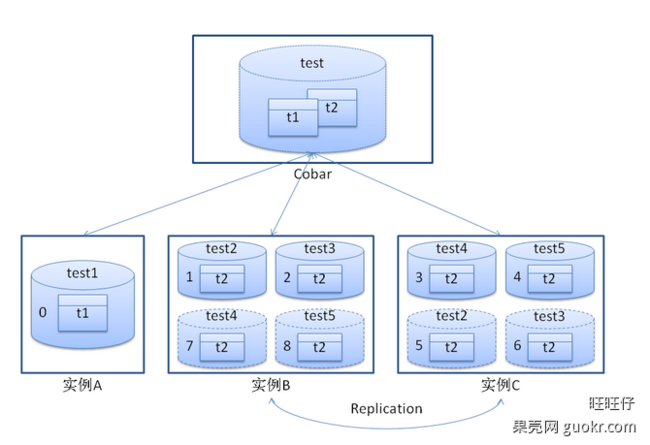
## Mysql  [cobar](http://www.bubuko.com/so/1/cobar" \o "cobar)

### **[cobar](http://www.bubuko.com/so/1/cobar" \o "cobar)** 简介

Cobar是阿里巴巴（B2B）部门开发的一种关系型数据的分布式处理系统，它可以在分布式的环境下看上去像传统数据库一样为您提供海量数据服务。那么具体说说我们为什么要用它，或说cobar--能干什么？以下是我们业务运行中会存在的一些问题：  
 1.随着业务的进行数据库的数据量和访问量的剧增，需要对数据进行水平拆分来降低单库的压力，而且需要高效且相对透明的来屏蔽掉水平拆分的细节。  
 2.为提高访问的可用性，数据源需要备份。  
 3.数据源可用性的检测和failover。  
 4.前台的高并发造成后台数据库连接数过多，降低了性能，怎么解决。   
针对以上问题就有了cobar施展自己的空间了，cobar中间件以proxy的形式位于前台应用和实际数据库之间，对前台的开放的接口是mysql通信协议。将前台SQL语句变更并按照数据分布规则转发到合适的后台数据分库，再合并返回结果，模拟单库下的数据库行为。



### [cobar](http://www.bubuko.com/so/1/cobar" \o "cobar)应用架构



### [cobar](http://www.bubuko.com/so/1/cobar" \o "cobar)应用介绍

通过Cobar提供一个名为test的数据库，其中包含t1,t2两张表。后台有3个MySQL实例(ip:port)为其提供服务，分别为：A,B,C。  
 期望t1表的数据放置在实例A中，t2表的数据水平拆成四份并在实例B和C中各自放两份。t2表的数据要具备HA功能，即B或者C实例其中一个出现故障，不影响使用且可提供完整的数据服务。  
cabar优点总结：  
数据和访问从集中式改变为分布：  
 （1）Cobar支持将一张表水平拆分成多份分别放入不同的库来实现表的水平拆分  
 （2）Cobar也支持将不同的表放入不同的库  
 （3） 多数情况下，用户会将以上两种方式混合使用  
注意！：Cobar不支持将一张表，例如test表拆分成test\_1,test\_2, test\_3.....放在同一个库中，必须将拆分后的表分别放入不同的库来实现分布式。  
解决连接数过大的问题。  
对业务代码侵入性少。  
提供数据节点的failover,HA：  
 (1)Cobar的主备切换有两种触发方式，一种是用户手动触发，一种是Cobar的心跳语句检测到异常后自动触发。那么，当心跳检测到主机异常，切换到备机，如果主机恢复了，需要用户手动切回主机工作，Cobar不会在主机恢复时自动切换回主机，除非备机的心跳也返回异常。  
 (2)Cobar只检查MySQL主备异常，不关心主备之间的数据同步，因此用户需要在使用Cobar之前在MySQL主备上配置双向同步。  
cobar缺点：  
开源版本中数据库只支持mysql，并且不支持读写分离。

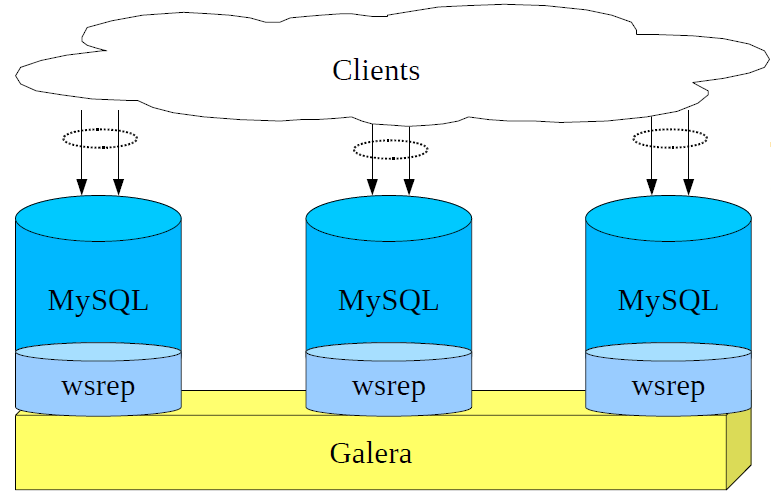
### [mysql](http://www.bubuko.com/so/1/mysql" \o "mysql)   [cluster](http://www.bubuko.com/so/1/cluster" \o "cluster)   [fabric](http://www.bubuko.com/so/1/fabric" \o "fabric)   [cobar](http://www.bubuko.com/so/1/cobar" \o "cobar)   [对比](http://www.bubuko.com/so/1/%e5%af%b9%e6%af%94" \o "对比)







## Galera Cluster



### Galera注意事项

1. 使用Galera必须要给MySQL-Server打wsrep补丁。可以直接使用官方提供的已经打好补丁的MySQL安装包，如果服务器上已经安装了标准版MYSQL，需要先卸载再重新安装。卸载前注意备份数据。
2. MySQL/Galera集群只支持InnoDB存储引擎。如果你的数据表使用的MyISAM，需要转换为InnoDB，否则记录不会在多台复制。可以在备份老数据时，为mysqldump命令添加–skip-create-options参数，这样会去掉表结构的声明信息，再导入集群时自动使用InnoDB引擎。不过这样会将AUTO\_INCREMENT一并去掉，已有AUTO\_INCREMENT列的表，必须在导入后重新定义。
3. MySQL 5.5及以下的InnoDB引擎不支持全文索引（FULLTEXT indexes），如果之前使用MyISAM并建了全文索引字段的话，只能安装MySQL 5.6 with wsrep patch。
4. 所有数据表必须要有主键（PRIMARY）,如果没有主键可以建一条AUTO\_INCREMENT列。
5. MySQL/Galera集群不支持下面的查询：LOCK/UNLOCK TABLES，不支持下面的系统变量：character\_set\_server、utf16、utf32及ucs2。
6. 数据库日志不支持保存到表，只能输出到文件（log\_output = FILE），不能设置binlog-do-db、binlog-ignore-db。
7. 跟其他集群一样，为了避免节点出现脑裂而破坏数据，建议Galera集群最低添加3个节点。
8. 在高并发的情况下，多主同时写入时可能会发生事务冲突，此时只有一个事务请求会成功，其他的全部失败。可以在写入/更新失败时，自动重试一次，再返回结果。
9. 节点中每个节点的地位是平等的，没有主次，向任何一个节点读写效果都是一样的。实际可以配合VIP/LVS或HA使用，实现高可用性。
10. 如果集群中的机器全部重启，如机房断电，第一台启动的服务器必须以空地址启动：mysqld\_safe–wsrep\_cluster\_address=gcomm:// >/dev/null &

### Replication API

#### wsrep API

**wsrep API**是通用的数据库复制插件接口，定义了实现事务型数据库和类似应用的同步式writeset复制所需的应用回调和复制插件调用集。wsrep API旨在将复制实现从应用细节中抽象和隔离出来。wsrep API的主要目标是基于认证的（certification-based）multi-master复制，但同样适用于异步和同步式master/slave复制。

**wsrep API**使用认为数据库服务器有状态的复制模型，其中状态是指数据库中的内容。当使用数据库时，用户修改数据库的内容，因而改变其状态。**wsrep API**将数据库状体表示为一系列原子改变（atomic changes），或者是事务。

在数据库集群中，所有节点总是保持相同的状态，通过复制和按相同序列顺序回放状态修改保持彼此同步。

#### Global Transaction ID（GTID）

为了保持集群中状态是同一的，wsrep API使用Golbal Transaction ID（GTID），GTID能够标识状态改变，并能够辨别当前状态与上一个状态改变的相对关系。GTID由两部分组成：（1）State UUID，状态及其经历的改变序列的唯一标识符；（2）Ordinal Sequence Number，seqno，表示改变在序列中位置的64位无符号整数。

#### Glaera Replication Plugin

#### Group Comunication Plugins

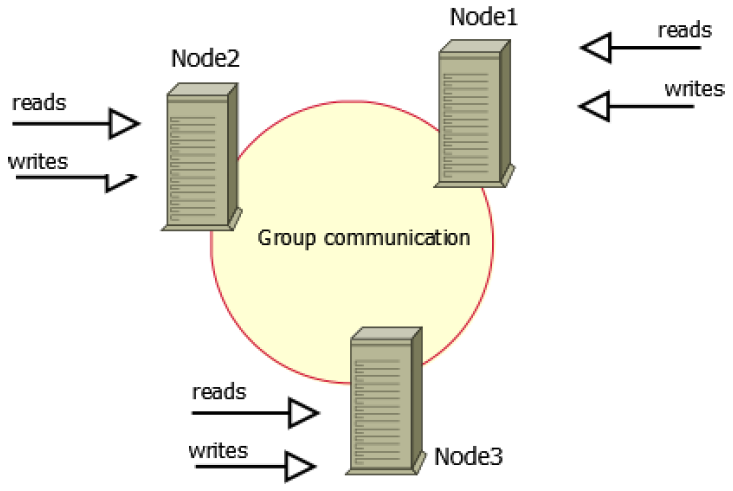
#### 参考资料

1. <https://launchpad.net/wsrep>
2. http://galeracluster.com/documentation-webpages/architecture.html

## Percona XtraDB Cluster(PXC)

### Percona XtraDB Cluster简介

Percona XtraDB Cluster（PXC）是免费、开源的MyDSQL高可用软件，具有如下**特征**：（1）PXC由Node组成，建议集群至少包含三个Node，但是两个Node也能运行；（2）每个Node是普通的MySQL/Percona Server实例，可以将已有MySQL/Percona Server转换为PXC重Node，也可以将Node从PXC中分离为普通Server；（3）每个Node包含所有数据。



PXC主要**优点**：（1）query在Node本地执行，所有数据在本地可见，不需要远程访问；（2）没有中央管控，任何Node任何时间失效都不影响Cluster的持续运行；（3）好的read负载扩展解决方案，可以在任何Node上执行read query。

PXC主要缺点：（1）增加新Node开销大，新增Node需要从一个已有Node拷贝所有数据；（2）不能作为有效的write扩展方案，将write流量分离到多个Node上，能够有些性能提升，但是不能期望太多，因为最终write将在所有Node上执行;（3）有多个数据复本，复本数与Node数相同。

PXC基于Percona Server with XtraDB，并且需要Write Set Replication补丁，使用Galera Library实现Multi-Master同步复制。

# 负载均衡调度

## HAProxy

### Haproxy简介

HAProxy提供高可用性、[负载均衡](http://baike.baidu.com/view/51184.htm" \t "http://baike.baidu.com/_blank)以及基于TCP和HTTP应用的代理，支持[虚拟主机](http://baike.baidu.com/view/7383.htm" \t "http://baike.baidu.com/_blank)，它是免费、快速并且可靠的一种解决方案。HAProxy特别适用于那些负载特大的web站点，这些站点通常又需要会话保持或七层处理。HAProxy运行在当前的硬件上，完全可以支持数以万计的[并发连接](http://baike.baidu.com/view/922692.htm" \t "http://baike.baidu.com/_blank)。并且它的运行模式使得它可以很简单安全的整合进您当前的架构中， 同时可以保护你的web服务器不被暴露到网络上。

HAProxy实现了一种[事件驱动](http://baike.baidu.com/view/536048.htm" \t "http://baike.baidu.com/_blank), 单一进程模型，此模型支持非常大的并发连接数。[多进程](http://baike.baidu.com/view/549647.htm" \t "http://baike.baidu.com/_blank)或多线程模型受内存限制 、系统调度器限制以及无处不在的锁限制，很少能处理数千并发连接。[事件驱动模型](http://baike.baidu.com/view/2877739.htm" \t "http://baike.baidu.com/_blank)因为在有更好的资源和时间管理的用户空间(User-Space) 实现所有这些任务，所以没有这些问题。此模型的弊端是，在多核系统上，这些程序通常扩展性较差。这就是为什么他们必须进行优化以 使每个CPU时间片(Cycle)做更多的工作

### 配置

配置HAProxy Session亲缘性的三种方式

haproxy负载均衡保持客户端和服务器Session亲缘性的三种方式:

1 用户IP 识别aproxy 将用户IP经过hash计算后 指定到固定的真实服务器上（类似于nginx 的IP hash 指令）配置指令 balance source

2 cookie 识别：haproxy 将WEB服务端发送给客户端的cookie中插入(或添加前缀)haproxy定义的后端的服务器COOKIE ID。配置指令例举 cookie SESSION\_COOKIE insert indirect nocache用firebug可以观察到用户的请求头的cookie里 有类似" Cookie jsessionid=0bc588656ca05ecf7588c65f9be214f5; SESSION\_COOKIE=app1" SESSION\_COOKIE=app1就是haproxy添加的内容

3 session 识别：haproxy 将后端服务器产生的session和后端服务器标识存在haproxy中的一张表里。客户端请求时先查询这张表。配置指令例举 appsession JSESSIONID len 64 timeout 5h request-learn

<http://severalnines.com/tutorials/mysql-load-balancing-haproxy-tutorial>

# MySQL数据同步错误及处理

## mysql主从同步失败原因

1. 网络的延迟
2. 主从两台机器的器的负载不一致
3. max\_allowed\_packet设置不一致
4. key自增键开始的键值跟自增步长设置不一致引起的主从不一致
5. mysql异常宕机情况下，如果未设置sync\_binlog=1或者 innodb\_flush\_log\_at\_trx\_commit=1很有可能出现binlog或者relaylog文件出现损坏，导致主从不一致
6. mysql本身的bug引起的主从不同步
7. 版本不一致，特别是高版本是主，低版本为从的情况下，主数据库上面支持的功能，从数据库上面不支持该功能

## 如何判断mysql主从同步是否成功

执行show slave status 命令

查看Slave\_IO\_Running: Slave\_SQL\_Running: 这两个参数的状态，正常是YES，如果是NO，那么主从复制肯定是有问题的

Slave\_IO\_Running: 如果等于Connecting 表示Master 没有启动MySQL服务。Last\_IO\_Error: 和 Last\_SQL\_Error:是否为null,为null则无错，有值则有错

## 如何判断mysql主从同步是否同步完成

简单来讲就是从库先通过io线程读取主库的二进制文件Master\_Log\_File和位置Read\_Master\_Log\_Pos然后缓存到本地（从库服务器）的中继文件Relay\_Log\_File中并记录已经读取到的位置Relay\_Log\_Pos，再通过从库的sql线程去读取中继文件Relay\_Log\_File，这个sql线程执行会记录已经执行到了哪个文件Relay\_Master\_Log\_File和哪个位置Exec\_Master\_Log\_Pos，如果Exec\_Master\_Log\_Pos和Read\_Master\_Log\_Pos相等就说明同步完成，否则同步末完成，正在同步。

## [MySQL同步异常处理](http://blog.itpub.net/29254281/viewspace-1181865/)

### **Slave\_IO＿Running ：NO 错误处理**

1）常见错误：

1.1 网络问题；

1.2 My.cnf配置文件有误

1.3 授权问题 ， replication slave和file权限是必须要有的；

### **Slave\_SQL＿Running ：NO 错误处理**

1)数据没有同步；

2)从节点上进行写操作；

处理办法：

**跳过当前错误**：但是这样数据已经不一致。 set GLOBAL SQL\_SLAVE\_SKIP\_COUNTER=1;

**从做主从：**这个方法不适用是数据量大的时候，因为从做主从会将主节点的bin-log文件传入到从节点中，进行重复制，数据量大工作量就会大，时间也会更多，所以不可取，

**从错误点开始重做主从：**找到错误点，从这个点开始做复制，这样这要及时的获取到错误，数据量不在，数据也能及时同步，数据也能保证一致。语句为

change master to   
> master\_host='master\_ip',  
> master\_user='用户名',   
> master\_password='密码',   
> master\_port=端口,   
> master\_log\_file=File',   
> master\_log\_pos=120 ;

本语句就能保证你从错误开始做主从；

参见

<http://blog.itpub.net/29254281/viewspace-1181865/>

<http://www.cnblogs.com/xiazh/archive/2011/04/22/1971182.html>

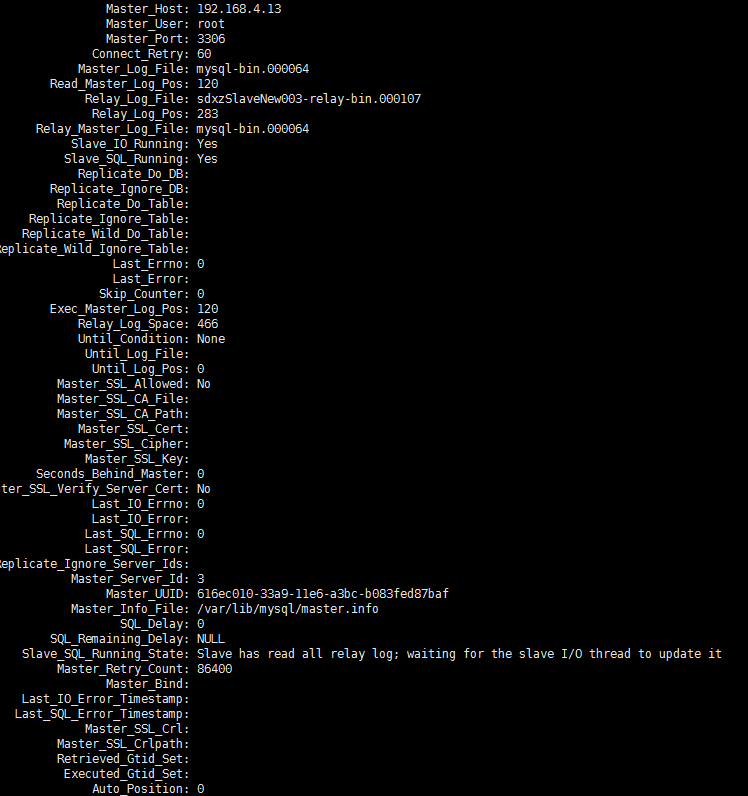
<http://blog.csdn.net/zbfaaadjl/article/details/18557975>

<http://www.cnblogs.com/luckcs/articles/2543607.html>

<http://www.2cto.com/database/201305/215708.html>

## [命令详解](http://blog.csdn.net/shiqidide/article/details/7263652)

### [SHOW SLAVE STATUS 详解](http://blog.csdn.net/shiqidide/article/details/7263652)



SHOW SLAVE STATUS会返回以下字段：

**Slave\_IO\_State：**SHOW PROCESSLIST输出的State字段的拷贝。SHOW PROCESSLIST用于从属I/O线程。如果线程正在试图连接到主服务器，正在等待来自主服务器的时间或正在连接到主服务器等，本语句会通知您

**Master**\_**User**：被用于连接主服务器的当前用户。

**Master\_Port：**当前的主服务器接口。

**Connect\_Retry：**--master-connect-retry选项的当前值

**Master\_Log\_File：**I/O线程当前正在读取的主服务器二进制日志文件的名称。

**Read\_Master\_Log\_Pos**：在当前的主服务器二进制日志中，I/O线程已经读取的位置。

**Relay\_Log\_File：**SQL线程当前正在读取和执行的中继日志文件的名称。

**Relay\_Log\_Pos**：在当前的中继日志中，SQL线程已读取和执行的位置。

**Relay\_Master\_Log\_File：**由SQL线程执行的包含多数近期事件的主服务器二进制日志文件的名称。

**Slave\_IO\_Running：**I/O线程是否被启动并成功地连接到主服务器上。

**Slave\_SQL\_Running：**SQL线程是否被启动。

**Replicate\_Do\_DB,Replicate\_Ignore\_DB**：使用--replicate-do-db和--replicate-ignore-db选项指定的[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/shiqidide/article/details/_blank)清单。

Replicate\_Do\_Table,Replicate\_Ignore\_Table,Replicate\_Wild\_Do\_Table,Replicate\_Wild\_Ignore\_Table

使用

--replicate-do-table,--replicate-ignore-table,--replicate-wild-do-table

和--replicate-wild-ignore\_table选项指定的表清单。

**Last\_Errno,Last\_Error：**被多数最近被执行的查询返回的错误数量和错误消息。错误数量为0并且消息为空字符串意味着“没有错误”。如果Last\_Error值不是空值，它也会在从属服务器的错误日志中作为消息显示。

举例说明：

**Last\_Errno:** 1051 Last\_Error: error 'Unknown table 'z'' on query 'drop table z' 该消息指示，表z曾经存在于在主服务器中并已被取消了，但是它没有在从属服务器中存在过，因此对于从属服务器，DROP TABLE失败。（举例说明，在设置复制时，如果您忘记了把此表拷贝到从属服务器中，则这有可能发生。）

**Skip\_Counter：**最近被使用的用于SQL\_SLAVE\_SKIP\_COUNTER的值。

**Exec\_Master\_Log\_Pos：**来自主服务器的二进制日志的由SQL线程执行的上一个时间的位置（Relay\_Master\_Log\_File）。

在主服务器的二进制日志中的(Relay\_Master\_Log\_File,Exec\_Master\_Log\_Pos)对应于在中继日志中的(Relay\_Log\_File,Relay\_Log\_Pos)。

**Relay\_Log\_Space：**所有原有的中继日志结合起来的总大小。

**Until\_Condition,Until\_Log\_File,Until\_Log\_Pos：**在START SLAVE语句的UNTIL子句中指定的值。

Until\_Condition具有以下值：如果没有指定UNTIL子句，则没有值

如果从属服务器正在读取，直到达到主服务器的二进制日志的给定位置为止，则值为Master

如果从属服务器正在读取，直到达到其中继日志的给定位置为止，则值为Relay

Until\_Log\_File和Until\_Log\_Pos用于指示日志文件名和位置值。日志文件名和位置值定义了SQL线程在哪个点中止执行。

Master\_SSL\_Allowed,Master\_SSL\_CA\_File,Master\_SSL\_CA\_Path,Master\_SSL\_Cert,Master\_SSL\_Cipher,Master\_SSL\_Key

这些字段显示了被从属服务器使用的参数。这些参数用于连接主服务器。Master\_SSL\_Allowed具有以下值：如果允许对主服务器进行SSL连接，则值为Yes，如果不允许对主服务器进行SSL连接，则值为No，如果允许SSL连接，但是从属服务器没有让SSL支持被启用，则值为Ignored。

与SSL有关的字段的值对应于--master-ca,--master-capath,--master-cert,--master-cipher和--master-key选项的值。

**Seconds\_Behind\_Master：**本字段是从属服务器“落后”多少的一个指示。当从属SQL线程正在运行时（处理更新），本字段为在主服务器上由此线程执行的最近的一个事件的时间标记开始，已经过的秒数。当此线程被从属服务器I/O线程赶上，并进入闲置状态，等待来自I/O线程的更多的事件时，本字段为零。总之，本字段测量从属服务器SQL线程和从属服务器I/O线程之间的时间差距，单位以秒计。

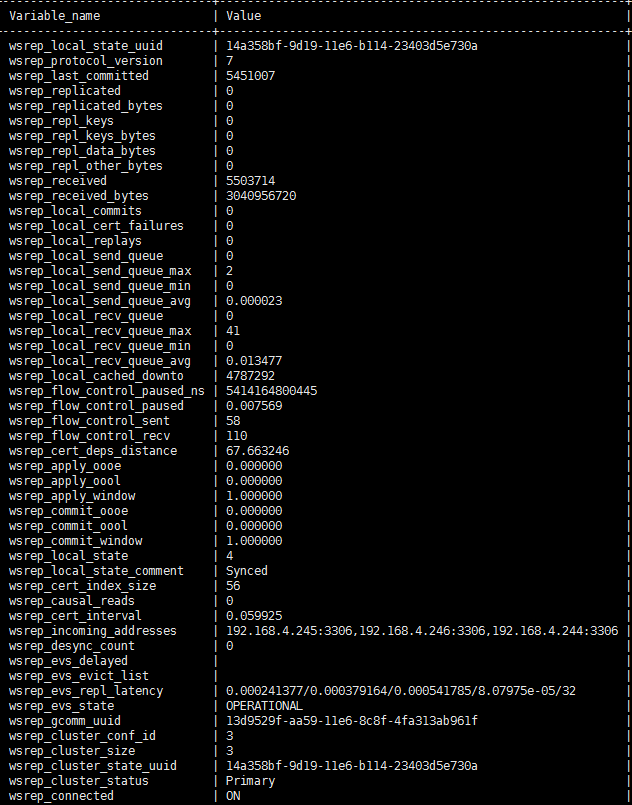
如果主服务器和从属服务器之间的网络连接较快，则从属服务器I/O线程会非常接近主服务器，所以本字段能够十分近似地指示，从属服务器SQL线程比主服务器落后多少。如果网络较慢，则这种指示不准确；从属SQL线程经常会赶上读取速度较慢地从属服务器I/O线程，因此，Seconds\_Behind\_Master经常显示值为0。即使I/O线程落后于主服务器时，也是如此。换句话说，本列只对速度快的网络有用。

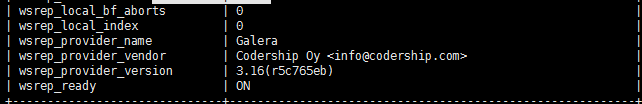
即使主服务器和从属服务器不具有相同的时钟，时间差计算也会起作用（当从属服务器I/O线程启动时，计算时间差。并假定从此时以后，时间差保持不变）。如果从属SQL线程不运行，或者如果从属服务器I/O线程不运行或未与主服务器连接，则Seconds\_Behind\_Master为NULL（意义为“未知”）。举例说明，如果在重新连接之前，从属服务器I/O线程休眠了master-connect-retry秒，则显示NULL，因为从属服务器不知道主服务器正在做什么，也不能有把握地说落后多少。

参见

<http://blog.csdn.net/shiqidide/article/details/7263652>

### show status like 'wsrep%'详解





集群内每个节点的wsrep\_cluster\_state\_uuid的value都应该是一样的，否则说明该节点不在集群中了

**wsrep\_cluster\_conf\_id：**显示了整个集群的变化次数。所有节点都应相同，否则说明某个节点与集群断开了  
 **wsrep\_cluster\_size：**显示了集群中节点的个数  
 **wsrep\_cluster\_status：**显示集群里节点的主状态。标准返回primary。如返回non-Primary或其他值说明是多个节点改变导致的节点丢失或者脑裂。如果所有节点都返回不是Primary，

则要重设quorum。具体参见

<http://galeracluster.com/documentation-webpages/quorumreset.html如果返回都正常，说明复制机制在每个节点都能正常工作，下一步该检查每个节点的状态确保他们都能收到write-set>

1. **检查节点状态**

节点状态显示了集群中的节点接受和更新write-set状态，以及可能阻止复制的一些问题

**1.wsrep\_ready**显示了节点是否可以接受queries。ON表示正常，如果是OFF几乎所有的query都会报错，报错信息提示“ERROR 1047 (08501) Unknown Command”

**2. SHOW GLOBAL STATUS LIKE 'wsrep\_connected**’显示该节点是否与其他节点有网络连接。（实验得知，当把某节点的网卡down掉之后，该值仍为on。说明网络还在）丢失连接的问题可能在于配置wsrep\_cluster\_address或wsrep\_cluster\_name的错误

**3.wsrep\_local\_state\_comment** 以人能读懂的方式显示节点的状态，正常的返回值是Joining, Waiting on SST, Joined, Synced or Donor，返回Initialized说明已不在正常工作状态

**三、查看复制的健康状态**

通过Flow Control的反馈机制来管理复制进程。当本地收到的write-set超过某一阀值时，该节点会启动flow control来暂停复制直到它赶上进度。监控本地收到的请求和flow control，有如下几个参数：

**wsrep\_local\_recv\_queue\_avg**——平均请求队列长度。当返回值大于0时，说明apply write-sets比收write-set慢，有等待。堆积太多可能导致启动flow control

**wsrep\_local\_recv\_queue\_max** 和 wsrep\_local\_recv\_queue\_min可以看队列设置的最大最小值。  
 **wsrep\_flow\_control\_paused** 显示了自从上次查询之后，节点由于flow control而暂停的时间占整个查询间隔时间比。总体反映节点落后集群的状况。如果返回值为1，说明自上次查询之后，节点一直在暂停状态。如果发现某节点频繁落后集群，则应该调整wsrep\_slave\_threads或者把节点剔除  
 **wsrep\_cert\_deps\_distance**显示了平行apply的最低和最高排序编号或者sql编号之间的平均距离值。这代表了节点潜在的并行程度，和线程相关  
**四、检测网络慢的问题**  
通过检查发送队列来看传出的连接状况  
 **wsrep\_local\_send\_queue\_avg**显示自上次查询之后的平均发送队列长度。比如网络瓶颈和flow control都可能是原因

**wsrep\_local\_send\_queue\_max** 和 wsrep\_local\_send\_queue\_min可以看队列设置的最大值和最小值  
**五、日志监控**  
在my.cnf中做如下配置  
# wsrep Log Options  
wsrep\_log\_conflicts=ON   #会将冲突信息写入错误日志中，例如两个节点同时写同一行数据  
wsrep\_provider\_options="cert.log\_conflicts=ON"    #复制过程中的错误信息写在日志中  
wsrep\_debug=ON    #显示debug 信息在日志中，其中也包括鉴权信息，例如账号密码。因此在生产环境中不开启

参见

<http://www.linuxeye.com/database/2862.html>

## MySQL主从同步操作注意事项

主从同步状态下关数据库顺序 先主后从

主从同步状态下开数据库顺序 先从后主 开启之后一定要查看IO和SQL线程是否启动