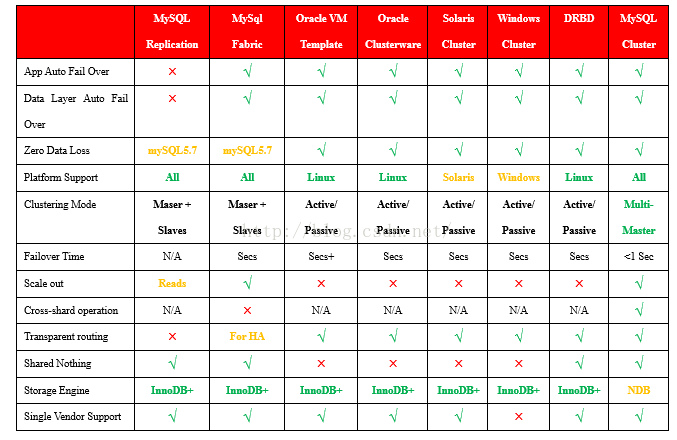
# **mySQL集群(cluster)**

这一章，我根本不打算写，因为mySQL 的 官方Cluster方案基本上都是bullshit，尤其是它的官方集群方案，竟然都无人维护了，而且mySQL集群完全可以用眼下另一种方案去做替换，根本无需做成cluster。

因此，在这一章为了不浪费读者的宝贵时间，我只会列出mySQL集群的几种比较方案，目前有一些第三方提供的mySQL集群方案还是不错的选择。  
MySQL的cluster方案有很多官方和第三方的选择，选择多就是一种烦恼，因此，我们考虑MySQL数据库满足下三点需求并来考察市面上可行的解决方案：

* 高可用性：主服务器故障后可自动切换到后备服务器
* 可伸缩性：可方便通过脚本增加DB
* 服务器负载均衡：支持手动把某公司的数据请求切换到另外的服务器，可配置哪些公司的数据服务访问哪个服务器

这是我列出的时下市面上比较流行的几种mySQL集群方案中一些核心功能的比较，供参考：



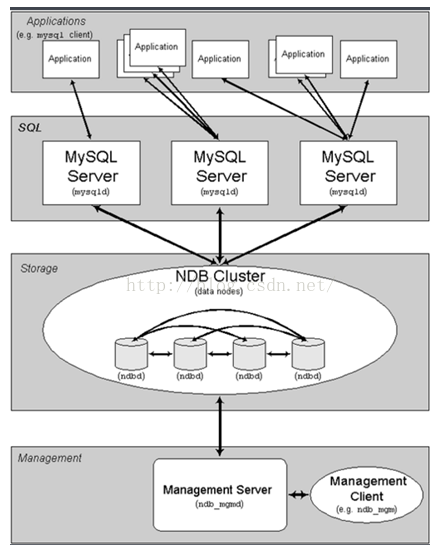
## **推荐第三方mySQL集群方案**

综合比较下来，笔者推荐采用MySQL Fabric和MySQL Cluster方案，以及另外一种较成熟的集群方案Galera Cluster。

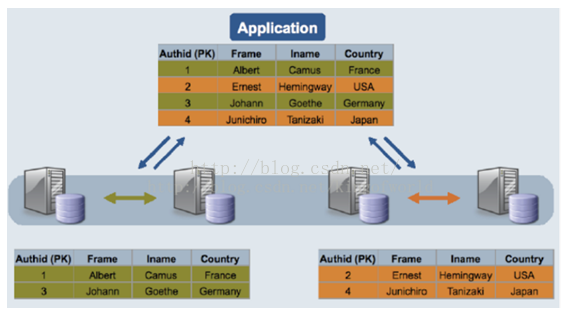
## **几种mySQL集群方案的比较**

### **MySQLCluster**

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主要由三种类型的服务组成：  
NDB Management Server：管理服务器主要用于管理cluster中的其他类型节点（Data Node和SQL Node），通过它可以配置Node信息，启动和停止Node。 SQL Node：在MySQL Cluster中，一个SQL Node就是一个使用NDB引擎的mysql server进程，用于供外部应用提供集群数据的访问入口。Data Node：用于存储集群数据；系统会尽量将数据放在内存中。



缺点及限制：

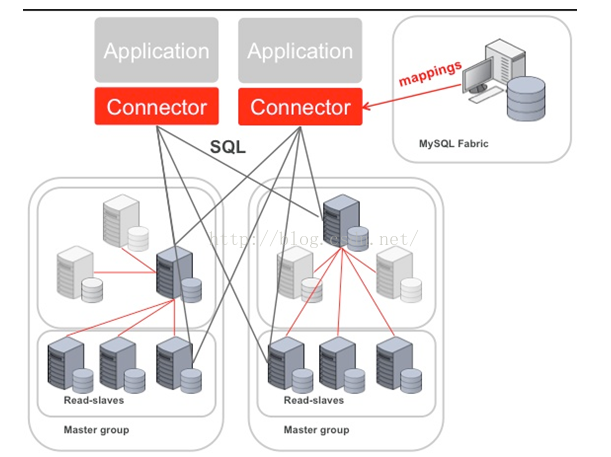
* 对需要进行分片的表需要修改引擎Innodb为NDB，不需要分片的可以不修改。
* NDB的事务隔离级别只支持Read Committed，即一个事务在提交前，查询不到在事务内所做的修改；而Innodb支持所有的事务隔离级别，默认使用Repeatable Read，不存在这个问题。
* 外键支持：虽然最新的Cluster版本已经支持外键，但性能有问题（因为外键所关联的记录可能在别的分片节点中），所以建议去掉所有外键。
* Data Node节点数据会被尽量放在内存中，对内存要求大。

****数据库系统提供了四种事务隔离级别：****

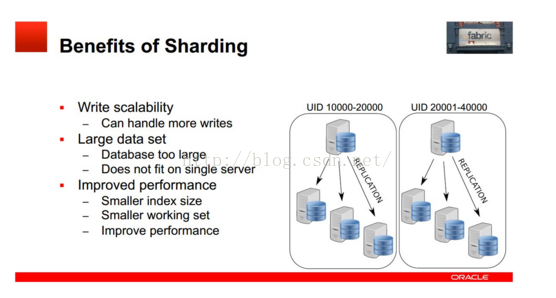
* Serializable（串行化）：一个事务在执行过程中完全看不到其他事务对数据库所做的更新（事务执行的时候不允许别的事务并发执行。事务串行化执行，事务只能一个接着一个地执行，而不能并发执行。）。
* Repeatable Read（可重复读）：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，但是不能看到其他其他事务对已有记录的更新。
* Read Commited（读已提交数据）：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，而且能看到其他事务已经提交的对已有记录的更新。
* Read Uncommitted（读未提交数据）：一个事务在执行过程中可以看到其他事务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

### **MySQL Fabric**

为了实现和方便管理MySQL 分片以及实现高可用部署，Oracle在2014年5月推出了一套为各方寄予厚望的MySQL产品 -- MySQL Fabric, 用来管理MySQL 服务，提供扩展性和容易使用的系统，Fabric当前实现了两个特性：高可用和使用数据分片实现可扩展性和负载均衡，这两个特性能单独使用或结合使用。  
MySQL Fabric 使用了一系列的python脚本实现。  
应用案例：由于该方案在去年才推出，目前在网上暂时没搜索到有大公司的应用案例。  
架构及实现原理：  
Fabric支持实现高可用性的架构图如下



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如下：



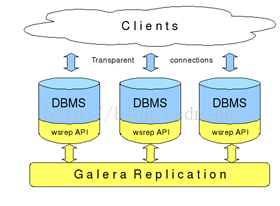
使用多个HA 组实现分片，每个组之间分担不同的分片数据（组内的数据是冗余的，这个在高可用性中已经提到）  
应用程序只需向connector发送query和insert等语句，Connector通过MasterGroup自动分配这些数据到各个组，或从各个组中组合符合条件的数据，返回给应用程序。  
缺点及限制：  
影响比较大的两个限制是：  
自增长键不能作为分片的键；事务及查询只支持在同一个分片内，事务中更新的数据不能跨分片，查询语句返回的数据也不能跨分片。  
分片：如何支持可扩展性和负载均衡  
当一台机器或一个组承受不了服务压力后，可以添加服务器分摊读写压力，通过Fabirc的分片功能可以将某些表中数据分散存储到不同服务器。我们可以设定分配数据存储的规则，通过在表中设置分片key设置分配的规则。另外，有些表的数据可能并不需要分片存储，需要将整张表存储在同一个服务器中，可以将设置一个全局组（Global Group）用于存储这些数据，存储到全局组的数据会自动拷贝到其他所有的分片组中。

### **Galera Cluster**

Galera Cluster号称是世界上最先进的开源数据库集群方案。



****主要优点及特性：****  
真正的多主服务模式：多个服务能同时被读写，不像Fabric那样某些服务只能作备份用同步复制：无延迟复制，不会产生数据丢失热备用：当某台服务器当机后，备用服务器会自动接管，不会产生任何当机时间自动扩展节点：新增服务器时，不需手工复制数据库到新的节点支持InnoDB引擎对应用程序透明：****应用程序不需作修改****。  
架构及实现原理：  
首先，我们看看传统的基于mysql Replication（复制）的架构图：



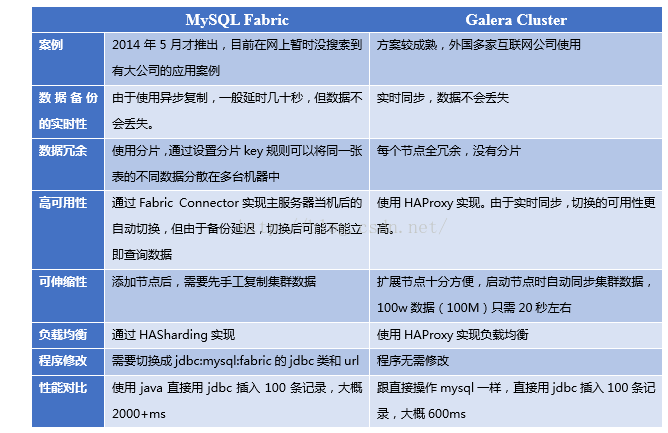
Replication方式是通过启动复制线程从主服务器上拷贝更新日志，让后传送到备份服务器上执行，这种方式存在事务丢失及同步不及时的风险。Fabric以及传统的主从复制都是使用这种实现方式。  
而Galera则采用以下架构保证事务在所有机器的一致性。  
客户端通过Galera Load Balancer访问数据库，提交的每个事务都会通过wsrep API 在所有服务器中执行，要不所有服务器都执行成功，要不就所有都回滚，保证所有服务的数据一致性，而且所有服务器同步实时更新。

****缺点及限制：****  
由于同一个事务需要在集群的多台机器上执行，因此网络传输及并发执行会导致性能上有一定的消耗。所有机器上都存储着相同的数据，全冗余。若一台机器既作为主服务器，又作为备份服务器，出现乐观锁导致rollback的概率会增大，编写程序时要小心。不支持的SQL：LOCK / UNLOCK TABLES / GET\_LOCK(), RELEASE\_LOCK()…不支持XA Transaction  
目前基于Galera Cluster的实现方案有三种：Galera Cluster for MySQL、Percona XtraDB Cluster、MariaDB Galera Cluster。  
我们采用较成熟、应用案例较多的Percona XtraDB Cluster。

****应用案例：****  
超过2000多家外国企业使用



## **Fabric对比Galera**



# **mySQL连接数优化**

我们如果经常遇见MySQL：ERROR1040：Too many connections的情况，一种情况是访问量确实很高，MySQL服务器扛不住了，这个时候就要考虑增加从服务器分散读压力，从架构层面。另外一种情况是MySQL配置文件中max\_connections的值过小。来看一个例子。  
mysql> show variables like 'max\_connections';  
+-----------------+-------+  
|  
Variable\_name | Value |  
+-----------------+-------+  
|  
max\_connections | 800 |  
+-----------------+-------+  
#### 这台服务器最大连接数是256，然后查询一下该服务器响应的最大连接数；  
mysql> show global status like 'Max\_used\_connections';  
+----------------------+-------+  
|  
Variable\_name | Value |  
+----------------------+-------+  
|  
Max\_used\_connections | 245 |  
+----------------------+-------+  
#### MySQL服务器过去的最大连接数是245，没有达到服务器连接数的上线800，不会出现1040错误。  
#### Max\_used\_connections /max\_connections \* 100% = 85%  
#### 最大连接数占上限连接数的85%左右,如果发现比例在10%以下，则说明MySQL服务器连接数的上限设置得过高了。

# **key\_buffer\_size**

key\_buffer\_size是设置MyISAM表索引缓存空间的大小，此参数对MyISAM表性能影响最大。下面是一台MyISAM为主要存储引擎服务器的配置：  
  
  
mysql> show variables like 'key\_buffer\_size';  
+-----------------+-----------+  
|  
Variable\_name | Value |  
+-----------------+-----------+  
|  
key\_buffer\_size | 536870912 |  
+-----------------+-----------+  
#### 从上面可以看出，分配了512MB内存给key\_buffer\_size。再来看key\_buffer\_size的使用情况：  
mysql> show global status like 'key\_read%';  
+-------------------+--------------+  
|  
Variable\_name | Value |  
+-------------------+-------+  
|  
Key\_read\_requests | 27813678766 |  
|  
Key\_reads | 6798830|  
+-------------------+--------------+  
 一共有27813678766个索引读取请求，有6798830个请求在内存中没有找到，直接从硬盘读取索引。  
key\_cache\_miss\_rate = key\_reads / key\_read\_requests \* 100%  
比如上面的数据，key\_cache\_miss\_rate为0.0244%，4000%个索引读取请求才有一个直接读硬盘，效果已经很好了，key\_cache\_miss\_rate在0.1%以下都很好，如果key\_cache\_miss\_rate在0.01%以下的话，则说明key\_buffer\_size分配得过多，可以适当减少。

# **mySQL的临时表**

当执行语句时，关于已经被创建了隐含临时表的数量，我们可以用如下命令查询其具体情况：  
  
  
mysql> show global status like 'created\_tmp%';  
+-------------------------+----------+  
|  
Variable\_name | Value |  
+-------------------------+----------+  
|  
Created\_tmp\_disk\_tables | 21119 |  
|  
Created\_tmp\_files | 6 |  
|  
Created\_tmp\_tables | 17715532 |  
+-------------------------+----------+  
#### MySQL服务器对临时表的配置：  
mysql> show variables where Variable\_name in ('tmp\_table\_size','max\_heap\_table\_size');  
+---------------------+---------+  
|  
Variable\_name | Value |  
+---------------------+---------+  
|  
max\_heap\_table\_size | 2097152 |  
|  
tmp\_table\_size | 2097152 |  
+---------------------+---------+  
每次创建临时表时，Created\_tmp\_table都会增加，如果磁盘上创建临时表，Created\_tmp\_disk\_tables也会增加。Created\_tmp\_files表示MySQL服务创建的临时文件数，比较理想的配置是：  
Created\_tmp\_disk\_tables / Created\_tmp\_files \*100% <= 25%  
比如上面的服务器：  
Created\_tmp\_disk\_tables / Created\_tmp\_files \*100% =1.20%，这个值就很棒了。

# **mySQL打开表的情况**

Open\_tables表示打开表的数量，Opened\_tables表示打开过的表数量，我们可以用如下命令查看其具体情况：  
   
mysql> show global status like 'open%tables%';  
+---------------+-------+  
|  
Variable\_name | Value |  
+---------------+-------+  
|  
Open\_tables | 351 |  
|  
Opened\_tables | 1455 |  
#### 查询下服务器table\_open\_cache;  
mysql> show variables like 'table\_open\_cache';  
+------------------+-------+  
|  
Variable\_name | Value |  
+------------------+-------+  
|  
table\_open\_cache | 2048 |  
+------------------+-------+  
如果Opened\_tables数量过大，说明配置中table\_open\_cache的值可能太小。  
比较合适的值为：  
open\_tables / opened\_tables\* 100% > = 85%  
open\_tables / table\_open\_cache\* 100% < = 95%

# **mySQL的进程使用情况**

如果我们在MySQL服务器的配置文件中设置了thread\_cache\_size，当客户端断开时，服务器处理此客户请求的线程将会缓存起来以响应一下客户而不是销毁(前提是缓存数未达上线)Thread\_created表示创建过的线程数，我们可以用如下命令查看：  
   
mysql> show global status like 'thread%';  
+-------------------+-------+  
|  
Variable\_name | Value |  
+-------------------+-------+  
|  
Threads\_cached | 40|  
|  
Threads\_connected | 1 |  
|  
Threads\_created | 330 |  
|  
Threads\_running | 1 |  
+-------------------+-------+  
#### 查询服务器thread\_cache\_size配置如下：  
mysql> show variables like 'thread\_cache\_size';  
+-------------------+-------+  
|  
Variable\_name | Value |  
+-------------------+-------+  
|  
thread\_cache\_size | 100 |  
+-------------------+-------+  
如果发现Threads\_created的值过大的话，表明MySQL服务器一直在创建线程，这也是比较耗费资源的，可以适当增大配置文件中thread\_cache\_size的值。

# **查询缓存(query cache)**

它主要涉及两个参数，query\_cache\_size是设置MySQL的Query Cache大小，query\_cache\_type是设置使用查询缓存的类型，我们可以用如下命令查看其具体情况：  
mysql> show global status like 'qcache%';  
+-------------------------+-----------+  
|  
Variable\_name | Value |  
+-------------------------+-----------+  
|  
Qcache\_free\_blocks | 22756 |  
|  
Qcache\_free\_memory | 76764704 |  
|  
Qcache\_hits | 213028692 |  
|  
Qcache\_inserts | 208894227 |  
|  
Qcache\_lowmem\_prunes | 4010916 |  
|  
Qcache\_not\_cached | 13385031 |  
|  
Qcache\_queries\_in\_cache | 43560 |  
|  
Qcache\_total\_blocks | 111212 |  
+-------------------------+-----------+  
****MySQL查询缓存变量的相关解释如下：****

* Qcache\_free\_blocks： 缓存中相领内存快的个数。数目大说明可能有碎片。flush query cache会对缓存中的碎片进行整理，从而得到一个空间块。
* Qcache\_free\_memory：缓存中的空闲空间。
* Qcache\_hits：多少次命中。通过这个参数可以查看到Query Cache的基本效果。
* Qcache\_inserts：插入次数，没插入一次查询时就增加1。命中次数除以插入次数就是命中比率。
* Qcache\_lowmem\_prunes：多少条Query因为内存不足而被清楚出Query Cache。通过Qcache\_lowmem\_prunes和Query\_free\_memory相互结合，能                          够更清楚地了解到系统中Query Cache的内存大小是否真的足够，是否非常频繁地出现因为内存不足而有Query被换出的情况。
* Qcache\_not\_cached：不适合进行缓存的查询数量，通常是由于这些查询不是select语句或用了now()之类的函数。
* Qcache\_queries\_in\_cache：当前缓存的查询和响应数量。
* Qcache\_total\_blocks：缓存中块的数量。

****query\_cache的配置命令：****

mysql> show variables like 'query\_cache%';  
+------------------------------+---------+  
|  
Variable\_name | Value |  
+------------------------------+---------+  
|  
query\_cache\_limit | 1048576 |  
|  
query\_cache\_min\_res\_unit | 2048 |  
|  
query\_cache\_size | 2097152 |  
|  
query\_cache\_type | ON |  
|  
query\_cache\_wlock\_invalidate | OFF |  
+------------------------------+---------+

****字段解释如下：****

* query\_cache\_limit：超过此大小的查询将不缓存。
* query\_cache\_min\_res\_unit：缓存块的最小值。
* query\_cache\_size：查询缓存大小。
* query\_cache\_type：缓存类型，决定缓存什么样的查询，示例中表示不缓存select sql\_no\_cache查询。
* query\_cache\_wlock\_invalidat：表示当有其他客户端正在对MyISAM表进行写操作，读请求是要等WRITE LOCK释放资源后再查询还是允许直接从Query Cache中读取结果，默认为OFF（可以直接从Query Cache中取得结果。）
* query\_cache\_min\_res\_unit的配置是一柄双刃剑，默认是4KB，设置值大对大数据查询有好处，但如果你的查询都是小数据查询，就容易造成内存碎片和浪费。
* 查询缓存碎片率 = Qcache\_free\_blocks /Qcache\_total\_blocks \* 100%
* 如果查询碎片率超过20%，可以用 flush query cache 整理缓存碎片，或者试试减少query\_cache\_min\_res\_unit，如果你查询都是小数据库的话。
* 查询缓存利用率 = (Qcache\_free\_size –  Qcache\_free\_memory)/query\_cache\_size \* 100% 。查询缓存利用率在25%一下的话说明query\_cache\_size设置得过大，可适当减少;查询缓存利用率在80%以上而且Qcache\_lowmem\_prunes > 50的话则说明query\_cache\_size可能有点小，不然就是碎片太多。
* 查询命中率 = (Qcache\_hits - Qcache\_insert)/Qcache)hits \* 100%，比如説：服务器中的查询缓存碎片率等于20%左右，查询缓存利用率在50%，查询命中率在2%，说明命中率很差，可能写操作比较频繁，而且可能有些碎片。

# **mySQL排序使用情况**

它表示系统中对数据进行排序时所用的Buffer，我们可以用如下命令查看：  
   
mysql> show global status like 'sort%';  
+-------------------+----------+  
|  
Variable\_name | Value |  
+-------------------+----------+  
|  
Sort\_merge\_passes | 10 |  
|  
Sort\_range | 37431240 |  
|  
Sort\_rows | 6738691532 |  
|  
Sort\_scan | 1823485 |  
+-------------------+----------+  
****Sort\_merge\_passes包括如下步骤：****MySQL首先会尝试在内存中做排序，使用的内存大小由系统变量sort\_buffer\_size来决定，如果它不够大则把所有的记录都读在内存中，而MySQL则会把每次在内存中排序的结果存到临时文件中，等MySQL找到所有记录之后，再把临时文件中的记录做一次排序。这次再排序就会增加sort\_merge\_passes。实际上，MySQL会用另外一个临时文件来存储再次排序的结果，所以我们通常会看sort\_merge\_passes增加的数值是建临时文件数的两倍。因为用到了临时文件，所以速度可能会比较慢，增大sort\_buffer\_size会减少sort\_merge\_passes和创建临时文件的次数，但盲目地增大sort\_buffer\_size并不一定能提高速度。