Mycat生产部署经验分享

——Mycat官方分享

Mycat作为一个强大灵活高性能的数据库中间件，在面临大数据库的时候，可以让DBA和架构师有更多的选择，而不是受限于MySQL本身或者应用本身。在实际部署中，为了达到系统最佳状态，我们需要对后端的MySQL做一些优化。

## 一：通用准则

 Mysql尽量用比较新的稳定版，当前来说5.6和5.7都是比较靠谱的一个选择，因为Mysq这两个版本做了大量优化。另外Mysql的各种变种版本都可以考虑。以下是一些通用准则：

* 对于非严格苛刻交易型的数据表，建议用MariaDB，这个版本目前在开源界很盛行，评价很高，percona版本也值得推荐，percona有很多辅助的运维工具。
* 对于交易型的数据表，可以考虑Mysql官方稳定版，若交易型的数据表要求可靠性非常高，比如是替代Oracle，也可以选择Galera Cluster这种高可用的方案，他以一定的写入性能损失带来了数据的高可用和高并发访问。
* 根据数据的可靠性要求，可以采用各种数据同步方案，比如1主多从，读写分离提升数据表的读的并发能力。
* 部分表可以用NoSQL方式存储，而前端访问方式不变，Mycat支持后端MongoDB和很多NoSQL系统，以提升查询能力
* 部分表可以采用MySQL内存表，来提升查询和写入速度，替代部分复杂缓存方案。

下面是一个可能的Mycat部署方案，不同的表用不同的存储方式，让不同的表根据其访问模式，都达到最佳状态。

Mycat

Table4

Table3

Table2

Table1

## 二：高可靠性

Mycat本身是无状态的，可以用HAProxy或四层交换机等设备组成Mycat的高可用集群，后端MySQL则配置为主从同步，此时整个系统就是高可用的。

HAproxy

MySQL Master

MySQL Slaver

MySQL Slaver

注意，当我们配置MySQL主从的时候，除了Galera Cluster是多主模式，其他标准是一主一从，此时，Mycat里面要将MySQL Slave配置为WriteHost，而不是readHost，这样当第一个WriteHost宕机，Mycat就会自动切断到第二个WriteHost，即MySQL Slave

上，完成自动的故障切换。

<dataHost name="localhost1" maxCon="2000" minCon="10" balance="0"

 writeType="0" dbType="mysql" dbDriver="native">

 <heartbeat>select user()</heartbeat>

 <!-- can have multi write hosts -->

 <writeHost host="hostM1" url="localhost:3306" user="root"

 password="123456"/>

 <writeHost host="hostS1" url="localhost2:3306" user="root"

 password="123456"/>

另外，对于Java应用来说，也可以使用Mycat的子项目 HA-DataSource ，<https://github.com/MyCATApache/Mycat-Tools/tree/master/HA-DataSource>

这个连接池具备HAProxy的特性，又省去了一个中间环节。

## 三：MySQL通用调优

**首先MySQL要绝对避免使用Swap内存，网上有多种办法，可以参考。**

这里是MySQL5.6及以上的调优参数，主要是提升多个database/table的写入和查询性能：

[mysqld]

当Order By 或者Group By等需要用到结果集时，参数中设置的临时表的大小小于结果集的大小时，就会将该表放在磁盘上，这个时候在硬盘上的IO要比内销差很多。所耗费的时间也多很多，Mysql 会取min(tmp\_table\_size, max\_heap\_table\_size)的值，因此两个设置为一样大小，除非是大量使用内存表的情况，此时max\_heap\_table\_size要设置很大。

max\_heap\_table\_size=200M

tmp\_table\_size=200M

下面这部分是Select查询结果集的缓存控制，query\_cache\_limit表示缓存的Select结果集的最大字节数，这个可以限制哪些结果集缓存，query\_cache\_min\_res\_unit表示结果集缓存的内存单元大小，若需要缓存的SQL结果集很小，比如返回几条记录的，则query\_cache\_min\_res\_unit越小，内存利用率越高，query\_cache\_size表示总共用多少内存缓存Select结果集，query\_cache\_type则是控制是否开启结果集缓存，默认0不开启，1开启，2为程序控制方式缓存，比如SELECT SQL\_CACHE …这个语句表明此查询SQL才会被缓存，对于执行频率比较高的一些查询SQL，进行指定方式的缓存，效果会最好。

FLUSH QUERY CACH命令则清理缓存，以更好的利用它的内存，但不会移除缓存，RESET QUERY CACHE 使命从查询缓存中移除所有的查询结果。

#query\_cache\_type =1

#query\_cache\_limit=102400

#query\_cache\_size = 2147483648

#query\_cache\_min\_res\_unit=1024

MySQL最大连接数，这个通常在1000-3000之间比较合适，根据系统硬件能力，需要对Linux打开的最大文件数做修改

max\_connections =2100

下面这个参数是InnoDB最重要的参数，是缓存innodb表的索引，数据，插入数据时的缓冲，尽可能的使用内存缓存，对于MySQL专用服务器，通常设置操作系统内存的70%-80%最佳，但需要注意几个问题，不能导致system的swap空间被占用，要考滤你的系统使用多少内存，其它应用使用的内在，还有你的DB有没有myisa引擎，最后减去这些才是合理的值。

innodb\_buffer\_pool\_size=4G

innodb\_additional\_mem\_pool\_size除了缓存表数据和索引外,可以为操作所需的其他内部项分配缓存来提升InnoDB的性能。这些内存就可以通过此参数来分配。推荐此参数至少设置为2MB，实际上，是需要根据项目的InnoDB表的数目相应地增加

innodb\_additional\_mem\_pool\_size=16M

innodb\_max\_dirty\_pages\_pct值的争议，如果值过大，内存也很大或者服务器压力很大，那么效率很降低，如果设置的值过小，那么硬盘的压力会增加.

innodb\_max\_dirty\_pages\_pct=90

MyISAM表引擎的数据库会分别创建三个文件：表结构、表索引、表数据空间。我们可以将某个数据库目录直接迁移到其他数据库也可以正常工作。然而当你使用InnoDB的时候，一切都变了。InnoDB 默认会将所有的数据库InnoDB引擎的表数据存储在一个共享空间中：ibdata1，这样就感觉不爽，增删数据库的时候，ibdata1文件不会自动收缩，单个数据库的备份也将成为问题。通常只能将数据使用mysqldump 导出，然后再导入解决这个问题。innodb\_file\_per\_table=1可以修改InnoDB为独立表空间模式，每个数据库的每个表都会生成一个数据空间。

独立表空间

优点：

1.每个表都有自已独立的表空间。

2.每个表的数据和索引都会存在自已的表空间中。

3.可以实现单表在不同的数据库中移动。

4.空间可以回收（drop/truncate table方式操作表空间不能自动回收）

5.对于使用独立表空间的表，不管怎么删除，表空间的碎片不会太严重的影响性能，而且还有机会处理。

缺点：

单表增加比共享空间方式更大。

结论：

共享表空间在Insert操作上有一些优势，但在其它都没独立表空间表现好。

**实际测试，当一个MySQL服务器作为Mycat分片表存储服务器使用的情况下，单独表空间的访问性能要大大好友共享表空间，因此强烈建议使用独立表空间。**

当启用独立表空间时，由于打开文件数也随之增大，需要合理调整一下 innodb\_open\_files 、table\_open\_cache等参数。

innodb\_file\_per\_table=1

innodb\_open\_files=1024

table\_open\_cache=1024

Undo Log 是为了实现事务的原子性，在MySQL数据库InnoDB存储引擎中，还用Undo Log来实现多版本并发控制(简称：MVCC)。Undo Log的原理很简单，为了满足事务的原子性，在操作任何数据之前，首先将数据备份到Undo Log，然后进行数据的修改。如果出现了错误或者用户执行了 ROLLBACK语句，系统可以利用Undo Log中的备份将数据恢复到事务开始之前的状态。因此Undo Log的IO性能对于数据插入或更新也是很重要的一个因素。于是，从MySQL 5.6.3开始，这里出现了重大优化机会：

As of MySQL 5.6.3**, you can store InnoDB undo logs in one or more separate undo tablespaces outside of the system tablespace**. This layout is different from the default configuration where the undo log is part of the system tablespace. The I/O patterns for the undo log make these tablespaces **good candidates to move to SSD storage**, while keeping the system tablespace on hard disk storage. [**innodb\_rollback\_segments**](http://dev.mysql.com/doc/refman/5.6/en/innodb-parameters.html#sysvar_innodb_rollback_segments)参数在此被重命名为innodb\_undo\_logs

因此总共有3个控制参数：innodb\_undo\_tablespaces表明总共多少个undo表空间文件，innodb\_undo\_logs定义在一个事务中innodb使用的系统表空间中回滚段的个数。如果观察到同回滚日志有关的互斥争用，可以调整这个参数以优化性能，默认是128最大值，官方建议先设小，若发现竞争，再调大

**注意这里的参数是要安装MySQL时候初始化InnoDB引擎设置的，innodb\_undo\_tablespaces参数无法后期设定。**

innodb\_undo\_tablespaces=128

innodb\_undo\_directory= SSD硬盘或者另外一块硬盘，跟数据分开

innodb\_undo\_logs=64

下面是InnoDB的日志相关的优化选项

innodb\_log\_buffer\_size这是 InnoDB 存储引擎的事务日志所使用的缓冲区。类似于 Binlog Buffer，InnoDB 在写事务日志的时候，为了提高性能，也是先将信息写入 Innofb Log Buffer 中，当满足 innodb\_flush\_log\_trx\_commit 参数所设置的相应条件(或者日志缓冲区写满)之后，才会将日志写到文件(或者同步到磁盘)中。innodb\_log\_buffer\_size 不用太大，因为很快就会写入磁盘。innodb\_flush\_log\_trx\_commit的值有0：log buffer中的数据将以每秒一次的频率写入到log file中，且同时会进行文件系统到磁盘的同步操作1：在每次事务提交的时候将log buffer 中的数据都会写入到log file，同时也会触发文件系统到磁盘的同步; 2：事务提交会触发log buffer 到log file的刷新，但并不会触发磁盘文件系统到磁盘的同步。此外，每秒会有一次文件系统到磁盘同步操作。对于非关键交易型数据，采用2即可以满足高性能的日志操作，若要非常可靠的数据写入保证，则需要设置为1，此时每个commit都导致一次磁盘同步，性能下降。

innodb\_log\_file\_size此参数确定数据日志文件的大小，以M为单位，更大的设置可以提高性能，但也会增加恢复故障数据库所需的时间。innodb\_log\_files\_in\_group分割多个日志文件，提升并行性。innodb\_autoextend\_increment对于大批量插入数据也是比较重要的优化参数（单位是M）

innodb\_log\_buffer\_size=16M

innodb\_log\_file\_size =256M

innodb\_log\_files\_in\_group=8

innodb\_autoextend\_increment=128

innodb\_flush\_log\_at\_trx\_commit=2

#建议用GTID的并行复制，以下是需要主从复制的情况下，相关的设置参数。

#gtid\_mode = ON

#binlog\_format = mixed

#enforce-gtid-consistency=true

#log-bin=binlog

#log-slave-updates=true

## 四：Mycat调优

使用最新JDK，比如JDK8

* system.xml中的processor参数设置为CPU核心数的2×4倍
* Linux主机的网络性能优化
* mycat所在服务器多网卡绑定，bond技术，增加网络吞吐量
* 合适的大内存，conf/wrapper.conf中修改

wrapper.java.additional.5=-XX:MaxDirectMemorySize=2G

wrapper.java.initmemory=2048

wrapper.java.maxmemory=2048

## 五：开发和设计方面

* 数据库索引非常重要，特别是超过百万行的表的情况下，没有索引，数据库性能很低，差别可以达到100倍，因此合适的索引很重要。
* 避免无意义的Select \* from xxx，因为这种语句会消耗很多IO和网络带宽
* 合适的分片方式，避免很多SQL是跨越多个分片执行
* 开发设计一开始就需要认证考虑大规模数据下的性能问题，精确严格的设计数据库