# Mysql介绍

## 0.innoDB引擎

从mysql5.5.x开始，默认的存储引擎更改为InnoDB Plugin引擎，提供了具有提交、回滚、和崩溃恢复能力的事务安全存储引擎；InnoDB锁定在行级，并且也在SELECT语句提供一个与Oracle风格一致的非锁定读，这些特性增加了多用户部署和性能。InnoDB存储引擎被完全整合到了mysql服务器中，不过，为了能在主内存中缓存数据和索引，InnoDB存储引擎会维持它自己的缓冲池。InnoDB会在一个表空间中存储它的表和索引。

那么如何采用InnoDB Plugin引擎呢？

在5.5.x版本的mysql中，只需修改my.cnf配置文件，添加如下内容：

innodb\_file\_per\_table=1

innodb\_file\_format=barracuda

innodb\_strict\_mode=1

barracuda格式支持表压缩功能，TRUNCATE TABLE的速度比以前要快

## 1.充分利用cpu多核到处理能力

在mysql5.1.X中，innodb\_file\_io\_threads参数默认是4，该参数在linux系统上是不可更改的,在windows系统上可以调整。

\*不支持动态改变，需要写入my.cnf中

这个参数的作用：InnoDB使用后台线程处理数据页上读写I/O（输入输出）请求的数量

在mysql5.5.x中，用两个新参数代替：

innodb\_read\_io\_threads

innodb\_write\_io\_threads

如果cpu是2颗8核到，那么可以如下设置：

innodb\_read\_io\_threads=8

innodb\_write\_io\_threads=8

如果数据库到读操作比较多，那么可以设置：

innodb\_read\_io\_threads=10

innodb\_write\_io\_threads=6

## 2.提高刷新脏页数量和合并插入数量，改善磁盘I/O处理能力

在5.1.x版本中，由于代码写死，因此最多只会刷新100个脏页到磁盘、合并20个插入缓冲，即使磁盘有能力处理更多的请求，也会处理这么多，这样在更新量较大（大批量insert）的时候，脏页刷新可能就会跟不上，导致性能下降。

innodb\_io\_capacity

参数可以动态调整刷新脏页的数量，避免了大批量insert到时候脏页刷新跟不上，导致性能下降

该参数默认是200

可以动态调整：SET GLOBAL innodb\_io\_capacity=2000;

参数值标准：

单盘 SAS/SATA 200

SAS\*12 RAID10 2000

SSD 5000

FUSION-IO 50000

## 3.让InnoDB\_Buffer\_Pool缓冲池中到热数据存活更久

InnoDB\_Buffer\_Pool缓冲区有两个区域，一个是sublist of new blocks 区域（经常被访问到区域——热数据），一个是sublist of old blocks区域（不经常访问到区域）,当用户访问数据的时候，如果缓存区里有相应的数据则直接返回，否则会从磁盘读入缓存区的sublist of old blocks区域，然后再移动到sublist of new blocks区域，并通过LRU最近最少使用算法来剔除就数据。

这里会存在一个问题，当有些sql语句做统计用全表扫描，这时就会进入sublist of new blocks区域，把一些真正的热数据“踢走”，这样就会造成缓冲区的数据进进出出，导致磁盘I/O频繁。所以做了如下限制：

控制进入缓冲区sublist of old blocks 区域到数量：

innodb\_old\_blocks\_pct

该参数默认是37，经常做全表扫描时，该参数可以设置低点，以防热数据被踢出缓冲区。

innodb\_old\_blocks\_time

在访问sublist of old blocks区域到数据块时，并不是马上就移动到sublist of new blocks，而是会在sublist of old blocks停留innodb\_old\_blocks\_time（微秒），再移动到sublist of new blocks，防止sublist of new blocks中到数据马上被踢出

## 4.InnoDB同时支持多个BufferPool实例

InnoDB用来缓存它的数据和索引的内存缓冲区的大小。把innodb\_buffer\_pool\_size参数值设置得越高，访问表中的数据需要的磁盘I/O就越少。在一个专用的数据服务器上，最高可以将这个参数设置为机器物理内存大小的80%。建议不要设置太高，因为对物理内存的竞争可能在操作系统上导致内存调度。

innodb\_buffer\_pool\_size 是InnoDB性能的决定性因素，如果数据库大小小于innodb\_buffer\_pool\_size设置的缓冲池大小，此时数据库的性能是最好的，因为客户端访问的数据都在内存里。

问题：

InnoDB\_Buffer\_Pool缓冲池复制管理着free list（初始化空闲页，为每一个page指定一个block头结构，并初始化各种mutex与rw-lock，将page加入Buffer\_Pool的free list链表，等待分配）、flush list（缓冲池产生的脏页（数据库被修改，但未写入磁盘），当innodb\_max\_dirty\_pages\_pct超过设置的值时，会把修改时间较早的page刷入磁盘）、LRU（在内存中但最近又不用的数据块，按照最近最少使用算法，MYSQL会根据哪些数据属于LRU而将其移除内存，从而腾出空间来加载另外的数据）等。

当Innodb\_Buffer\_Pool缓冲池达到好几十G时，如果某个线程正在更新缓冲池，将会造成其他线程必须等待的瓶颈。

方案：

可以通过innodb\_buffer\_pool\_instances参数来增加InnoDB\_Buffer\_Pool实例的个数，并使用哈希函数将读取缓存的数据页随机分配到一个缓冲池里面，这样每个缓冲区实例就可以分别管理着自己的free list 、flush lis、LRU，也就可以解决问题。

可以通过参数innodb\_buffer\_pool\_size必须大于1GB时，生成InnoDB\_Buffer\_Pool多实例才有效，最多支持64个InnoDB\_Buffer\_Pool实例

参数：

innodb\_buffer\_pool\_instances=3

调整完后可以用命令 show engine innodb status\G;来查看

## 5.可选择使用内存分配程序

从5.5.x开始，用户可以控制InnoDB是使用自带的内存分配程序，还是使用当前操作系统中现有的更高效的内存分配程序。

参数：

innodb\_use\_sys\_malloc

默认设置为1，表示InnoDB使用操作系统到内存分配程序

TCMalloc是Google开发到开源工具，与标准glibc库中到malloc相比，TCMalloc在内存到分配效率和速度上要高到多，可以在很大程度上提高mysql服务器在高并发情况下到性能，降低了系统负载。

在安装TCMalloc之前，先要安装关联到libunwind（32位操作系统忽略此步骤）

安装命令：

#wget http://download.savannah.gnu.org/releases/libunwind/libunwind-0.99.tar.gz

#tar zxvf libunwind-0.99.tar.gz

#cd libunwind-0.99

#CFLAGS=-fPIC ./configure –enable-shared

#make CFLAGS=-fPIC

#make CFLAGS=-fPIC install

安装google-perftools：

#wget <http://lnamp-web-server.googlecode.com/files/google-perftools-1.7.tar.gz>

#tar zvxf google-perftools-1.7.tar.gz

#cd google-perftools-1.7

#./configure

#make && make install

#vi /usr/local/MySQL/bin/MySQLd\_safe

然后在#executing MySQLd\_safe的下一行加入以下内容：

export LD\_PRELOAD=/usr/local/lib/libtcmalloc.so

最后重启MySQL服务：

#service mysql restart

查看是否生效：

#lsof -n |grep tcmalloc

## 6.提高默认innodb线程并发数

InnoDB使用操作系统线程来处理用户事务请求，是这样工作的：当InnoDB收到一个用户的请求时，如果已经超过innodb\_thread\_concurrency预先设置的并发线程数量，那么就会按照innodb\_thread\_sleep\_delay预先设定的值休眠N秒，之后再次尝试连接，重试两次的机制是为了减少CPU上下文切换的次数，以降低CPU消耗。如果请求被接受，则会获得一个innodb\_concurrency\_tickets默认500次的通行证，在这些次数用完之前，该线程重新请求时无须再次进行前面所说的innodb\_thread\_concurrency的检查。如果还没有接受，那么就会进入队列中，知道最终被处理掉。

从5.5.x版本开始，innodb\_thread\_concurrency被默认设置为0，表示不限制并发数。

nnodb\_thread\_concurrency=0时，innodb\_thread\_sleep\_delay参数就无效了；同样，innodb\_concurrency\_tickets 也没有意义。这里推荐设置innodb\_thread\_concurrency为0，这样就可以更好的发挥CPU多核处理能力，提高并发量。

从mysql5.6.1开始，这个选项就被废了。

参数：

innodb\_thread\_concurrency 预先设定到并发线程数量

innodb\_thread\_sleep\_delay 预先设定的值休眠N秒（当线程数量超过预先设定到并发线程数量时，新来到线程就会休眠指定N秒后，再次尝试连接，重试两次的机制是为了减少cpu上下文切换到次数，以降低CPU消耗）

innodb\_concurrency\_tickets 默认500次到通行证（请求被接受后）

## 7.预读算法的变化

两种预读算法提高I/O性能，一种是线性预读，一种是随机预读；

线性预读：当顺序读取extent块（包含64个page）innodb\_read\_ahead\_threshold设置的page页数量时，触发 一个异步读取请求，将下一个页提前读取到buffer pool中。在MySQL5.1.X版本中，顺序读取extent块最后一个页时，InnoDB决定是否将下一个页提前读取到 InnoDB\_Buffer\_Pool缓冲池中。

随机预读：如果在InnoDB\_Buffer\_Pool缓冲池中发现同一个extent块内有若干个页，那么会触发一个异步读取请求，把剩余的页读取进来，随机预读增加了不必要的复杂性，常常导致性能下降，因此，在MySQL5.5.X版本中已经将其删除了。

参数：

innodb\_read\_ahead\_threshold 默认是56.可动态修改

show engine innodb status\G; 可查看：

其中:

Pages read ahead :表示每秒预读了多少页

evicted without access :表示预读到页没有被访问，每秒被踢出了多少页

如果发现有很多evicted without access，说明该参数到值太小，应该增大。

## 8.在linux上实现了异步I/O

同步I/O：线程启动一个I/O操作后会立即进入等待状态，直到I/O操作完成才醒来继续执行；

异步I/O：线程启动一个I/O请求到内核后，然后继续处理其他到事情，内核完成I/O请求后，将会通知线程I/O完成。

5.5.x开始实现了异步I/O功能，也就是linux native aio，要想使用linux native aio，可以利用libaio库，libaio对linux native aio的系统调用进行了简单的封装，当然，也可以直接通过系统调用来使用linux native aio。

在使用libaio之前，需要现安装 libaio rpm包：

yum install libaio -y

rpm -qa | grep libaio

参数：

可以通过innodb\_use\_native \_aio参数来选择是否启用异步I/O，默认是ON，即处于开启状态，此参数不支持动态修改

可通过cat /proc/slabinfo|grep kio命令，来查看异步I/O是否能正常工作

如果kiocb那项不为0，代表异步I/O已工作。

## 9.恢复组工作

背景：

一个事务提交时，采取的是先写日志后刷入磁盘的方式，假如此时有多个用户同时提交，那么按照顺序把写入的事务日志刷入磁盘上，就会导致磁盘做多次I/O操作，从而降低IOPS吞吐率。

方案：

从MySQL5.5.X版本开始，会采用组提交的方式来将事务刷入磁盘中，也就是说，如果有多个用户同时提交事务，那么就合并在一起一次性来刷入磁盘，大大提高了吞吐量。

举个搬饮料放入库房的例子：以前，搬运工每次搬一箱饮料放入库房，这样进入库房的频率就很高，他来来回回的也很累，后来他索性每次搬5箱饮料，这样一次搬的东西多了，进出库房的频率也就变低了。

注意：

组提交工作模式只支持在sync\_binlog = 0的情况下, 同样，innodb\_support\_xa也必须等于0。其目的是保证InnoDB存储引擎的redo log事务日志与binlog日志的顺序一致。

参数：

sync\_binlog=0

innodb\_support\_xa=0

保证innodb存储引擎的redo log事务日志与binlog日志顺序一致。

## 10.改善清除程序进度

InnoDB中清除操作是一类定期回收无用数据的操作，在之前的版本中，清除操作是主线程的一部分，这意味着它在运行时可能会堵塞其他的数据操作，比如删除一张大表。

从5.5.x开始，该操作运行于独立的线程中，并支持更多的并发数。用户可以通过设置innodb\_purge\_threads配置参数来选择清除操作是否使用单独的线程，默认情况下设置为0（不使用单独线程），设置为1时表示使用单独的清除线程。

## 11.添加删除缓冲和清除缓冲

在对一个表进行增删改查操作时，里面到索引（聚集索引和非聚集索引）也会更新，其中，主键（聚集索引）是按照顺序进行插入的，而非聚集索引则会分散插入。

顺序读写的速度要比随机读写到速度快，表越大越明显，而插入到性能就会降低。

当一个表做insert操作来更新非聚集索引时：

如果该非聚集索引页被读入Innodb\_Buffer\_Pool缓冲池里，那么就直接更新非聚集索引，并使用正常的写脏数据块方法将其闪存到磁盘中；

如果没有读入缓冲池里，则使用插入缓冲区来缓存非聚集索引页的变化，直到该页被读入Innodb\_Buffer\_Pool缓冲池里，执行插入缓存合并操作，并使用正常的写脏数据块方法将其闪存到磁盘中，从而提高了插入性能。

然而，由于插入缓冲区占用部分Innodb\_Buffer\_Pool缓冲池，因此使得缓存数据页的可用内存减少了。如果数据和索引全部读入Innodb\_Buffer\_Pool缓冲池，并且表中有相对较少的非聚集索引，那么就可以关闭 InnoDB的插入缓冲功能。

如果该非聚集索引页被读入Innodb\_Buffer\_Pool缓冲池里，那么就会直接更新非聚集索引，并使用正常的写脏数据块方法将其闪存到磁盘中，这样一来，插入缓冲区就没有什么作用了，并且还占用一定的内存，这种情况下关闭该功能较好。

5.5.x可用来控制删除缓冲区和插入缓冲区功能，默认是all

参数：

SET GLOBAL innodb\_change\_buffering=all;

## 12.控制自旋锁Spin Lock轮训间隔

何谓自旋锁？

它是为保护共享资源而提出的一种锁机制。其实，自旋锁与互斥锁比较类似，它们都是为了解决对某项资源的互斥使用的。无论是互斥锁，还是 自旋锁，在任何时刻，最多只能有一个保持者，也就是说，在任何时刻最多只能有一个执行单元获得锁。但是两者在调度机制上略有不同。对于互斥锁，如果资源已 经被占用，资源申请者只能进入睡眠状态。但是自旋锁不会引起调用者睡眠，如果自旋锁已经被别的执行单元保持，调用者就一直循环在那里看该自旋锁的保持者是 否已经释放了锁，“自旋”一词就是因此而得名。

为了防止自旋锁循环过快，耗费CPU，在MySQL5.5.X版本里引入了innodb\_spin\_wait\_delay参数，作用是控制轮训间 隔，也就是说在每次轮训的过程中，会休息一会儿然后再轮训。比如，在用一个死循环监控服务状态时，那么每次会睡眠5秒，然后再进行检查，代码如下所示：  
#!/bin/bash  
while true  
do  
 pstree -p MySQL > /dev/null  
 if [ $? -eq 0 ];then  
echo "OK. "

else  
 echo "MySQL is down. " | mail -s "aleat" hechunyang@139.com  
 fi  
 sleep 5  
done

注意

innodb\_spin\_wait\_delay参数的值默认是6，可动态调整。

set global innodb\_spin\_wait\_delay=6;

## 13.快速创建、删除、更改索引

背景：

在5.1.x版本里，创建和删除聚集索引的过程如下：

a.创建一个和原表结构一样的空表，然后创建聚集索引；

b.复制原表的数据到新表，这时会对原表加一个排他锁，其他的绘画dml操作会阻塞，从而保证数据的一致性；

c.复制完毕后删除原表，并把新表改名为原表。

创建和删除非聚集索引的过程如下：

a.创建一个和原表一样的空表，然后创建非聚集索引；

b.复制原表的数据到新表，这时会对原表加一个共享锁，其他的会话不能更新，但可以查询数据，从而保证数据的一致性；

c.复制完毕后删除原表，并把新表改名为原表。

5.5.x版本开始，创建和删除非聚集索引不用复制整个表的内容，只须更新表的索引页；和之前相比，速度会更快，但创建聚集索引(主键)或者是外键时，还是需要复制表的内容，因为聚集索引是把primary key以及row data保存在一起的，而secondary index则是单独存放的，有个指针指向primary key。

## 14.支持创建压缩数据页

从MySQL5.5.X版本开始支持InnoDB数据页压缩，数据页的压缩使数据文件体积变小，减少磁盘I/O，提高吞吐量，小成本地提高了CPU利用率。尤其是对读多写少的应用来说最为有效，同样的内存可以存储更多的数据，充分地“榨干”内存利用率。

工作原理是：当用户获取数据时，如果压缩的页没有在Innodb\_Buffer\_Pool缓冲池里，那么会从磁盘加载进去，并且会在 Innodb\_Buffer\_Pool缓冲池里开辟一个新的未压缩的16 KB的数据页来解压缩，为了减少磁盘I/O以及对页的解压操作，在缓冲池里同时存在着被压缩的和未压缩的页。为了给其他需要的数据页腾出空间，缓冲池里会 把未压缩的数据页踢出去，而保留压缩的页在内存中，如果未压缩的页在一段时间内没有被访问，那么会直接写入磁盘中，因此缓冲池中可能有压缩和未压缩的页， 也可能只有压缩页。

InnoDB采用最近最少使用（LRU）算法，将经常被访问的热数据放入内存里。当访问一个压缩表时，InnoDB会通过自适应的LRU算法来实现内存中 压缩页和未压缩页的平衡，其目的是避免当CPU繁忙时花费太多的时间在解压缩上，也是为了避免在CPU空闲时在解压缩操作上做过多的I/O操作。

当系统处于I/O瓶颈时，这个算法会踢出未压缩的页，而不是未压缩和压缩的页，从而为更多的页注入内存腾出空间；

而当系统处于CPU瓶颈时，这个算法会同时踢出未压缩的页和压缩的页，留出更多的内存来存放热数据，减少解压缩带来的开销。

在以前的版本中，一个数据页是16 KB，现在可以在建表时指定压缩的页是1 KB、2 KB、4 KB还是8 KB，如果设置过小，会导致消耗更多的CPU，通常设置为8 KB。

注意，必须采用Barracuda文件格式且独立表空间，才支持数据页压缩，如下所示：  
innodb\_file\_format = Barracuda  
innodb\_file\_per\_table = 1

要设置数据页为8 KB，在建表的时候加入ROW\_FORMAT=COMPRESSED KEY\_BLOCK\_SIZE=8即可，代码如下：  
CREATE TABLE 'compressed' (  
'id' int(10) unsigned NOT NULL AUTO\_INCREMENT,  
'k' int(10) unsigned NOT NULL DEFAULT '0',  
'c' char(120) NOT NULL DEFAULT '',

'pad' char(60) NOT NULL DEFAULT '',  
PRIMARY KEY ('id'),  
KEY 'k' ('k')  
) ENGINE=InnoDB  
DEFAULT CHARSET=gbk  
ROW\_FORMAT=COMPRESSED  
KEY\_BLOCK\_SIZE=8

## 15.动态关闭innodb更新元数据的统计功能

innodb\_stats\_on\_metadata参数的作用是：每当查询information\_schema元数据库里的表时，InnoDB还 会随机提取其他数据库每个表索引页的部分数据，从而更新information\_schema.STATISTICS表，并返回刚才查询的结果。当你的表 很大，且数量很多时，耗费的时间就会很长，很多经常不访问的数据也会进入Innodb\_Buffer\_Pool缓冲池里，那么就会污染缓冲池，并且 ANALYZE TABLE和SHOW TABLE STATUS语句也会造成InnoDB随机提取数据。

从MySQL5.5.X版本开始，你可以动态关闭innodb\_stats\_on\_metadata，不过默认是开启的。关闭方式如下：

set global innodb\_stats\_on\_metadata=OFF；

# 安全性、稳定性的显著改变

在早期的版本中，mysql主从复制都是异步的，主库发送完binlog日志后，不会查看备库是否接收完毕，如果从库由于各种原因导致落后于主库，此时出现当机，这时备库中的数据就是不完整的；而半同步复制在一定成都上保证提交的事务已经传给了至少一个备库，因此，半同步与异步复制相比，进一步提高了数据完整性。

## 16.中继日志relay-log可自我修复

5.5.x版本开始增加该参数：

relay\_log\_recovery

作用：

当从库当机后，假如relay-log损坏了，导致一部分中继日志没有处理，则自动放弃所有未执行到relay-log,并且重新从master上获取日志，这样就保证了relay-log的完整性。默认该功能是关闭了的，relay\_log\_recovery=1时，开启，建议开启。

## 17.开启InnoDB严格检查模式

特别是采用了页数据压缩功能后，最好是开启该功能。

作用：当对表到操作有错误时，不会有警告信息提示，直接抛出错误

参数：

set global innodb\_strict\_mode=1;

## 18.支持动态更改独立表空间

参数：

innodb\_file\_per\_table=1;

对于innodb的数据结构，首先要解决两个概念性的问题: 共享表空间以及独占表空间。  
什么是共享表空间和独占表空间  
共享表空间以及独占表空间都是针对数据的存储方式而言的。

共享表空间: 某一个数据库的所有的表数据，索引文件全部放在一个文件中，默认这个共享表空间的文件路径在data目录下。 默认的文件名为:ibdata1初始化为10M。

独占表空间:每一个表都将会生成以独立的文件方式来进行存储，每一个表都有一个.frm表描述文件，还有一个.ibd文件。

其中这个文件包括了单独一个表的数据内容以及索引内容，默认情况下它的存储位置也是在表的位置之中。

两者之间的优缺点   
共享表空间：  
优点：  
可以将表空间分成多个文件存放到各个磁盘上（表空间文件大小不受表大小的限制，如一个表可以分布在不同步的文件上）。数据和文件放在一起方便管理。  
缺点：  
所有的数据和索引存放到一个文件中以为着将有一个很常大的文件，虽然可以把一个大文件分成多个小文件，但是多个表及索引在表空间中混合存储，这样对于一个表做了大量删除操作后表空间中将会有大量的空隙，特别是对于统计分析，日值系统这类应用最不适合用共享表空间。

独立表空间：

在配置文件（my.cnf）中设置： innodb\_file\_per\_table =1；

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（除drop table操作处，表空不能自已回收）  
a)Drop table操作自动回收表空间，如果对于统计分析或是日值表，删除大量数据后可以通过:alter table TableName engine=innodb;回收不用的空间。  
b)对于使innodb-plugin的Innodb使用turncate table也会使空间收缩。  
c)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：  
单表增加过大，如超过100个G。  
  
相比较之下，使用独占表空间的效率以及性能会更高一点。  
共享表空间以及独占表空间之间的转化

**19.支持动态更改InnoDB锁超时时间**

默认是50秒

**20.半同步复制安装配置**

半同步复制插件在目录/usr/local/mysql/lib/plugin下，只需(进入该目录)按如下步骤即可安装：

install plugin rpl\_semi\_sync\_master soname 'semisync\_master.so';

set GLOBAL rpl\_semi\_sync\_master\_enabled=ON;

install plugin rpl\_semi\_sync\_slave soname 'semisync\_slave.so';

set GLOBAL rpl\_semi\_sync\_slave\_enabled=ON;

在初次加载该插件后，mysql会将该插件记录到系统表mysql.plugin中，下次启动时，系统会自动加载该插件，无须再次执行上面到命令。

另外，在my.cnf配置文件里加入：

rpl\_semi\_sync\_master\_enabled=1

rpl\_semi\_sync\_slave\_enabled=1

这样以后启动mysql时就会自动开启半同步复制功能。

<http://blog.csdn.net/goustzhu/article/details/9339621>

[MySQL数据的主从复制、半同步复制和主主复制详解 - goustzhu地盘 - 博客频道 – CSDN.NET](http://blog.csdn.net/goustzhu/article/details/9339621)

其他配置参数：

在master主库上有4个相关参数：

rpl\_semi\_sync\_master\_enabled=ON 表示在master上已经开启半同步复制模式

rpl\_semi\_sync\_master\_timeout=10 000 该参数默认为10 000毫秒(可调)，即10秒，用来表示如果主库在某次事务中到等待时间超过10秒，则降级为异步复制模式，不再等待slave从库。如果主库再次探测到slave从库恢复了，则会自动再次回到同步复制模式。

rpl\_semi\_sync\_master\_wait\_no\_slave 表示是否允许master每个事务提交后都要等待slave的接受信号确认，默认为ON，即每个事务都要等待。如果为OFF，则slave追赶上后，也不会开启半同步复制模式，需要手工开启。

rpl\_semi\_sync\_master\_trace\_level=32,指用于开启半同步复制模式时的调试级别，默认是32

在从库上有两个参数：

rpl\_semi\_sync\_slave\_enabled=ON 表示在slave上已经开启半同步复制模式

rpl\_semi\_sync\_slave\_trace\_level=32 调试级别，默认为32.

详细配置请参考：

mysql的数据同步功能配置.dox

# 故障诊断

影响mysql性能的因素：

最主要的因素就是磁盘I/O，其次是内存。

**影响linux服务器性能的因素：**

在操作系统层面影响linux服务器性能的因素主要就是服务器CPU、内存、磁盘I/O、网络I/O、以及linux系统本身的内核。

## 系统性能评估指标

1. cpu性能指标：

从整体上来说，CPU性能指标比较多，因为CPU处理的事物也比较多。如下：

CPU使用率：

如果在持续一段时间里CPU使用率大雨80%，这就可能表明CPU出现了瓶颈；

%us:应用程序（用户空间）:

表示用户应用进程所花费的CPU百分比，包括Nice时间；如果用户时间值很高，表明系统正在执行实际的工作；

%sy:系统（内核空间）：

表示内核操作所花费的CPU百分比，包括中断。系统时间值持续很高表明网络或驱动器堆栈可能存在瓶颈。通常，系统指挥花费很少时间在内核时间上。

%wa：I/O等待：

等待I/O操作所需要的CPU时间总和，系统不应该花费过多的时间等待I/O操作，否则你应该检查一下I/O子系统各方面的性能；

%id:空闲时间

表示CPU空闲的百分比，这个值越大表明系统CPU的负荷越小；

%ni：Nice时间

表示花费在执行re-nicing(改变进程的执行顺序和优先级)进程的CPU百分比

除了上面介绍的这些性能指标以外，还有诸如队列中等待执行的进程数、上下文切换、中断等。

1. 内存性能指标：

影响内存性能指标无非就是内存大小和虚拟空间大小，内存的性能指标相对比较少:

空闲内存：

与其他操作系统相比，在linux系统上不必过分在意空闲内存值，因为linux内核会将大量未使用的内存分配给文件系统来缓存数据，用free –m命令看到内存使用率时会疑惑，就是这个原因。实际空闲内存应为已用内存扣除用于缓冲和缓存的数量而得到；

交换空间使用：

这个值表示已使用的交换空间大小，相当于windows系统中的虚拟内存；

交换空间的使用只能告诉你linux在管理内存上是多么的有效，想要确定内存是否存在瓶颈，需要使用到Swap In/Out，如果Swap In/Out长时间保持在每秒钟超过200~300页，可能表示内存存在瓶颈。

1. 磁盘性能指标：

磁盘I/O等待：

CPU在等待I/O操作时所花费的时间。如果这个值持续很高，很可能表示I/O存在瓶颈；

队列平均长度：

I/O请求的数量，通常硬盘队列值为2~3时最佳；过高可能表示硬盘I/O存在瓶颈；

平均等待时间

I/O请求服务所花费的平均时间，等待时间包括实际I/O操作的时间和在I/O队列中等待的时间，单位为毫秒；

## 19.开源监控和评估工具介绍

dstat:性能监控器。

安装：

#wget <http://pkgs.repoforge.org/dstat/dstat-0.7.2-1.el5.rfx.noarch.rpm>

#rpm -ivh d[stat-0.7.2-1.el5.rfx.noarch.rpm](http://pkgs.repoforge.org/dstat/dstat-0.7.2-1.el5.rfx.noarch.rpm)

语法：

dstat[-afv][options...][delay[count]]

## 20.连接数过多导致程序连接报错到原因

一般情况下：可以通过增大max\_connections值，来解决；

在正常情况下，mysql处理完一条请求后，会根据wait\_timeout值来释放连接（含义：服务器关闭非交互连接之前等待活动到秒数），一般设置为100秒即可。默认为28 800秒

解决办法：

在数据库压力很大的情况下，重启完数据库，通过手工执行下列语句，把热数据加载到InnoDB\_Buffer\_Pool缓冲池里进行预热，从而避免了早高峰连接数升高，程序报错。

select count(\*) from user;select count(\*) from user1;select count(\*) from user3;

在mysql5.6里，为了解决上述问题，提供了一个新特性来快速预热Buffer\_Pool缓存池：

innodb\_buffer\_pool\_dump\_at\_shutdown=1

解释：在关闭时把热数据dump到本地磁盘

innodb\_buffer\_pool\_dump\_now=1

解释：采用手工方式把热数据dump到本地磁盘

innodb\_buffer\_pool\_load\_at\_startup=1

解释：在启动时，把热数据加载到内存

innodb\_buffer\_pool\_load\_now=1

解释：采用手工方式把热数据加载到内存

注意：只有在正常关闭mysql服务，或者pkill mysql时，才会把热数据导出到内存。机器当机或者pkill -9 mysql，是不会导出的。

## 21.磁盘高负荷拖垮mysql

由于插入操作很频繁，导致了磁盘高负荷运转，结果就是CPU Wait I/O升高，负载加大；

该问题的瓶颈是磁盘I/O繁忙导致CPU负载升高。

解决办法：

加内存，把InnoDB\_Buffer\_Pool调整到内存的70%，减少磁盘I/O的压力。

22.记录子查询引起的宕机

案例：服务器的swap分区报警，之后内存不足报警，再后来内存耗尽，导致机器死机；

原因：子查询性能低，特别是在WHERE从句中的IN()子查询；对于该查询，mysql需要先为内层查询语句的查询结果建立一个临时表，然后外层查询语句才能在临时表中查询记录，查询完毕后，mysql还需要测小这些临时表。

方案：

使用连接查询来代替子查询，连接查询不需要建立临时表，其速度比子查询要快。

## 23.诊断事务量突高的原因

诊断步骤：

A．通过binlog来分析数据库在事务量突高时间之前和之后的表；

B.找出写操作频繁的表；

C．接着查看binlog日志，找出些操作频繁的表

二进制日志小知识：二进制日志由配置文件的log-bin选项负载启用，MYSQL服务器将在数据根目录创建两个新文件XXX-bin.001和xxx-bin.index，若配置选项没有给出文件名，mysql将使用主机名称命名这两个文件，其中.index文件包含一份全体日志文件的清单。Mysql会把用户对所有数据库的内容和结构的修改情况记入XXX-bin.n文件中，而不会记录SELECT和没有实际更新的UPDATE语句。

二进制日志有两个作用:

一是恢复数据：通过之前的全量备份，和之后的增量备份来恢复；

二是实现mysql的主从复制。

## 24.事物隔离级别的设置

用下列语句查询全局和会话事物隔离级别：

SELECT @@global.tx\_isolation;

SELECT @@session.tx\_isolation;

SELECT @@tx\_isolation;

用下面的语句该表打个会话或者所有新进链接的隔离级别：

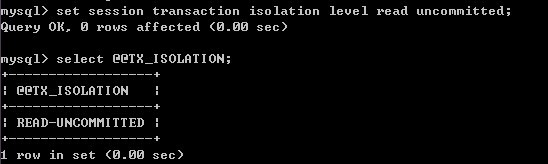
1.全局修改，修改mysql.ini配置文件，在最后加上

1 #可选参数有：READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, SERIALIZABLE.

2 [mysqld]

3 transaction-isolation = REPEATABLE-READ

2.对当前session修改，在登录mysql客户端后，执行命令：



## 25.谨慎设置binlog\_format=MIXED

binlog\_format有三种格式：STATEMENT、ROW和MIXED

STATEMENT在二进制日志里，记录到是实际的SQL语句；

ROW在二进制日志里记录的不是简单的SQL语句，而是实际行到变更；

在二进制日志里，MIXED默认还是采用STATEMENT格式记录的，但在下面这6种情况下会转化为ROW格式:

1.NDB引擎，表的DML操作增、删、改会以ROW格式记录；

2.SQL语句里包含了UUID()函数；

3.自增长字段被更新了；

4.包含了INSERT DELAYED语句；

5.使用了用户定义函数；

6.使用了临时表。

注意：

如果采用的是默认隔离级别REPEATABLE-READ，那么建议设置binlog\_format=ROW。如果是READ-COMMITTED隔离级别，binlog\_format=MIXED和binlog\_format=ROW的效果是一样的，binlog记录的格式都是ROW。

如果binlog\_format 为row格式，那么数据量很大的时候，日志量就会很大，再一个就是写日志所带来的I/O问题？

如果设置为row格式，binlog文件会比STATEMENT格式大很多，而且主从复制都是通过binlog来传输的，binlog的增大也增大了网络开销。

总结：

设置为row格式，则更为安全稳定，却丢失了性能；根据实际需求选择。

在mysql5.6里，针对这个问题进行了优化，可以通过设置binlog\_row\_image=minimal，解决binlog日志过大的问题

当设置为minimal后，binlog日志文件只记录影响后的行记录。

参数：

innodb\_lock\_wait\_timeout=10

## 26．未设置swap分区导致内存耗尽，主机死机

情况：

主机被Hang死，可以ping通，但ssh连接不上，导致cluster无法切换，只有人工去重启那台被Hang死的机器才能正常切换。

原因:

当cp一个大文件，或者在主库上mysqldump数据库时，当物理内存不够用的时候，操作系统就会把mysql所拥有的一部分地址空间映射到swap上去，然而并没有设置swap分区。虽然有设置swappiness为0，但这只能减少使用swap的概率，并不能避免linux使用swap，所以内存耗尽后，主机仍旧会被hang死。

解决方案：

1. 增加2GB的swap分区，避免内存耗尽时机器死机，可以给予些缓冲，重启前端程序释放mysql压力；
2. 增加内存监控，当内存使用率达到90%，通过重启mysql来是否内存，避免机器死机（使用内存Nagios监控脚本）

## 27.mysql故障切换之事件调度器注意事项

基础知识：

　　根据MySQL的官方文档，从MySQL的5.1.6版本开始，MySQL支持了事件调度器，用于处理事件的调度与执行。触发器用于根据DML操作来触发事件，而事件调度器则是定时触发事件，功能类似于Linux的crontab计划任务，但是控制更为精确。在MySQL支持这项功能之前，往往通过Linux的crontab来辅助进行定时任务的触发，而当我们对于任务有更高的定时要求时，或者考虑调用接口处的性能瓶颈时，就需要使用到事件调度器（Event Scheduler）。

　　首先确保数据库选项event\_scheduler处于开启状态。

|  |  |
| --- | --- |
|  | show variables like 'event\_scheduler'; |

　　如果我们看到ON，就表示这个功能处于开启状态，如果为OFF，表示关闭，DISABLED表示禁用。请注意，在数据库实例启动的情况下，我们可以随时在on和off间进行切换，但是，无法从on或off切换到disabled，也无法从disabled切换到on或off。只有在数据库关闭的状态下，这种转换才会生效。（DISABLED状态是在5.1.12版本中引入的）

|  |  |
| --- | --- |
|  | set @@global.event\_scheduler=on; |

　　设置完成后，事件调度器就被开启了，实质上，是在MySQL实例中新建了一个事件调度器线程，通过show processlist\G命令，可以找到一个User为event\_scheduler的线程，就表示事件调度器现在可以使用了。

　　接着我们随便创建一个存储过程，比如往某个表插入一条记录，好让事件调度器进行调度。

|  |  |
| --- | --- |
|  | delimiter //  create procedure `do\_something`()  begin    insert into timelog values ();    end; //  delimiter ; |

　　创建完毕后，我们就开始创建事件，事件最重要的部分有以下几个：

* 事件名称，这个很简单，就是给事件起个名字，加上IF NOT EXISTS的话，如果给定名称的事件已经存在了就不会再创建，并且只抛出一个Warning。
* 调度时间（ON SCHEDULE），可以定义事件被触发的具体时间，也可以指定事件被触发的周期，如果指定周期性任务，还可以指定开始时间和结束时间。
* 事件完成后的行为（ON COMPLETION），默认为NOT PRESERVED，即事件完成以后删除任务本身，注意，对于周期性的任务，不表示事件只做一次，而是指到达结束时间后，再删除这个事件。可以将它指定为PRESERVED，那么当事件完成以后，不会被删除。
* 事件体，就是事件具体需要做什么，定义方式和存储过程类似。

|  |  |
| --- | --- |
|  | CREATE EVENT IF NOT EXISTS `log\_every\_minute`  ON SCHEDULE EVERY 1 MINUTE  ON COMPLETION PRESERVE  DO  CALL do\_something(); |

　　在event\_scheduler为on的情况下，事件一经创建就会生效。

　　这样，我们就成功创建了一个事件，每分钟会在timelog表中插入一条记录，当然，我们可以根据自己的需要，让MySQL去做一些更为复杂的任务。

注意：事件只能在master上触发，在slave上不能触发，如果slave上触发了，同步复制就会坏掉。当主从故障切换后，VIP漂移到了以前的slave上，此时slave就成了新的master。但这时，时间的状态还是维持着slaveSIDE\_DISALE,并不是也改成了ENABLE，这样就会造成切换后，事件无法执行。所以需要人工重新开启事件状态：

Mysql>alter event ‘事件名’enable；

## 28.人工误删innoDB数据文件，如何恢复

注意：当删除数据文件后，此时你会发现数据库还可以正常工作，数据照样可以写入，切记，这个时候千万别把mysql进程杀掉，否则没法挽救。

使用rm -f ib\*删除数据文件和重做日志文件，做测试；

步骤：

1.找到mysqld的进程pid：

#netstat -ntlp | grep mysqld

2.输入以下命令：

# ll /proc/步骤1中的pid/fd | egrep 'ib\_|ibdata'

其中，时间后面的数字就是我们要恢复到文件；

3.关闭前端业务，或者执行如下：

FLUSH TABLES WITH READ LOCK；

这一步的作用是让数据库没有写入操作，以便完成后面的恢复工作；如何验证没有写入操作？

先输入以下命令，让脏页尽快刷入到磁盘里。

set global innodb\_max\_dirty\_pages\_pct=0;

然后查看binlog日志写入情况，确保File和Position的值没有再变化：

show master status；

最后再查看innodb状态信息，确保脏数据页已经刷入磁盘：

show engine innodb status\G;

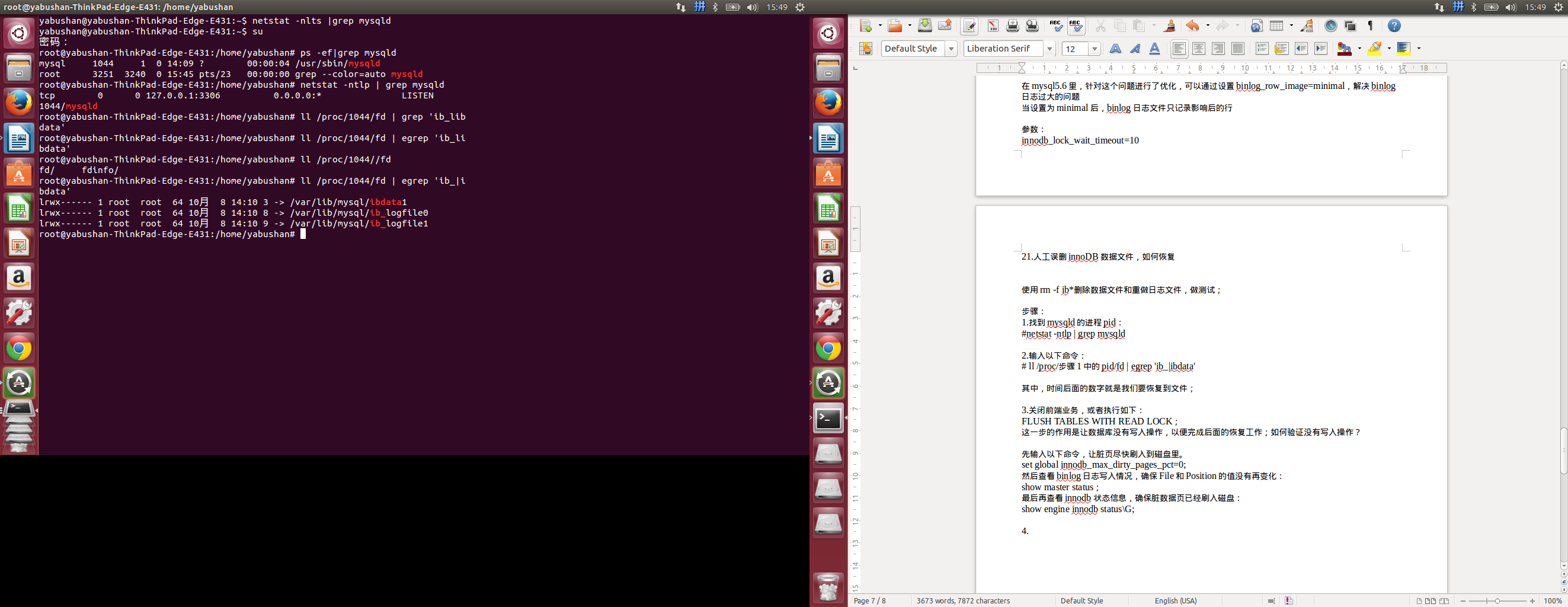
4.将步骤2列出的这些文件复制到原来的目录下：

#cd /proc/1044/fd

#cp 3 /var/lib/mysql/ibdata1

#cp 8 /var/lib/mysql/ib\_logfile0

#cp 9 /var/lib/mysql/ib\_logfile1



5.修改用户属性：

#cd /var/lib/mysql//

#chown mysql:mysql ib\*

6.重启mysql即可：

## 29.update忘记加where条件误操作恢复

参考文档：

<http://www.cnblogs.com/gomysql/p/3582058.html>

# 同步复制报错故障处理

## 30.异步复制和半同步复制的区别

异步复制：

Master把binlog发送过去，不管slave是否接收完，也不管是否执行完，这一动作就结束；

半同步复制：

Master把binlog发送过去，slave确认接收完，但不管它是否执行完，给master一个信号我这边收到了，这一动作就结束了（半同步复制patch谷歌写的代码，mysql5.5上正式应用）。

异步的劣势：

当master上写操作繁忙时，当前POS点，例如，是10，而slave上IO\_THREAD线程接收过来的是3，此时master宕机，会造成7个点未传送到slave上而数据丢失。

下面的这3中故障是在HA集群切换时产生的，由于是异步复制，且sync\_binlog=0,会造成一小部分binlog没接收完，从而导致同步报错。

1. 在master上删除一条记录时出现的故障

案例：在master上删除一条记录后，slave上因找不到该记录而报错，出现这个情况的原因可能是因为主机上已将其删除。

方案：可采取丛集直接跳过的方式解决

Slave上执行如下命令：

Stop slave ;set global sql\_slave\_skip\_counter=1;start slave;

1. 主键重复

案例：主从数据不一致时，slave上已经有该条数据，但又在master上插入了同一条数据，此时就会报错。

方案：

1. 在slave上使用命令：desc 表名

来查看表结构，得到主键的字段名；

1. 删除重复的主键:delete from tablename where id=’id’;
2. 开启同步复制功能：start slave;
3. 在master和slave上确认是否成功删除重复的主键：select \* from tablename;
4. 在master上更新一条记录，而slave上却找不到

案例：主从数据不一致时，master上已经有该条记录，但slave上没有这条记录，之后若在master上又更新了这条记录，此时就会报错。

方案：

1. 在master上，用myqlbinlog分析一下出错的binlog在干什么？

# mysqlbinlog --no-defaults -v -v --base64-output=DECODE-ROWS mysql-bin.000024 | grep –A ‘10’ end\_pos

注：end\_pos 是指出错的位置，在报错信息中会出现

1. 将丢失的数据填补到slave上；
2. 跳过报错：stop slave ;set\_global sql\_slave\_skip\_counter=1;start slave;

## 31.slave的中继日志relay-log损坏

原因：当slave意外宕机时，有可能会损坏中继日志relay-log，再次开启同步复制时，报错。

方案：找到同步的binlog日志和POS点，然后重新进行同步；

①Mysql>show slave status\G;

这里有两个比较重要的参数：

Relay\_master\_Log\_File:MySQL-bin.000010

Exec\_master\_Log\_Pos:821

②重置主从复制：

Mysql>stop slave;

Mysql>CHANGE master TO master\_LOG\_FILE=’MySQL-bn.000010’,master\_LOG\_POS=821;

Mysql>start slave;

注：在5.5中有考虑到该情况，只需要在slave的my.cnf中增加一个参数relay\_log\_recovery=1就可以了。可参考本文章的16

## 32.避免在master上执行大事务

案例：当删除一些大数据时，该删除操作就会变成一个事物，一下子就可能把slave给卡死；

方案：

把该删除操作写成一个存储过程，每删除指定的数据就提交一次，循环操作直至删除完成。

## 33.自动处理同步复制错误

参数：slave\_exec\_mode

动态设置该参数：set global slave\_exec\_mode=’IDEMPOTENT’;

设置该参数后并不能立即生效，需要重启复制进程，第一步：关闭同步复制，第二步：开启同步复制，这样才可以。设置完毕后，当出现错误时，就会自动跳过该错误，并且记录到日志里。（slave\_skip\_errors参数效果一样，该参数不能动态设定，只能在my.cnf中设置）

## 34.如何验证主从数据库一致

Maatkit是一个开源的工具包，为mysql日常管理提供了帮助，目前，已被Percona公司收购并维护。其中mk-table-checksum是用来检测master和slave上的表结构是否一致，而mk-table-sync则是在主从数据不一致时，用来修复的。这两个脚本在运行时都会锁表，表的大小取决于执行的快慢，勿在高峰期间运行，可选择凌晨。

## 35.binlog\_ignore\_db引起的同步复制故障

案例：在master上使用binlog\_ignore\_db命令忽略了一个库以后，使用mysql –e执行的所有语句就不写binlog了。

原因：因为忽略某个库的复制有两个参数，一个是binlog\_ignore\_db,另一个是replicate-ignore-db.

区别：binlog\_ignore\_db参数是设置在主库上的，replicate-ignore-db参数是设置在从库上的。

如果想在slave上忽略一个库的复制，最好是不要用binlog\_ignore\_db这个参数，使用replicate-ignore-db=youdb取代。

## 36.恢复slave从机上的某几张表的简要方法

在日常工作中，同步报错是经常碰到的问题，如果修复后还没有解决，通常的方法是在master上重新导出一份，然后再slave上恢复，这个方法是针对整个库不是很大的情况下的，如果是较大时，全部到处再倒入耗时就很长。

案例:有a1，b1，c1三张表的数据跟master上的不一致，操作方法如下：

1. 停止slave复制，命令如下：

Mysql>stop slave;

1. 在主库上导出这三张表，并记录下同步的binlog和POS点：

#mysqldump –uroot -p123456 -q --single-transaction --master-data=2 youdb a1 b1 c1 >./a1\_b1\_c1.sql

1. 查看a1\_b1\_c1.sql文件，找出记录的binlog和POS点：

#more a1\_b1\_c1.sql

例如master\_LOG\_FILE=’MYSQL-bin.002974’,master\_LOG\_POS=550534;

1. 把a1\_b1\_c1复制到slave机器上，并做Change master指向：

Mysql>start slave until master\_LOG\_FILE= ‘MYSQL-bin.002974’,master\_LOG\_POS=550534;

直到sql\_thread线程为NO,这期间的同步报错一律跳过即可，可用如下命令跳过：

Stop slave;set global sql\_slave\_skip\_count=1;start slave;

注意：

这一步是为了保障其他表的数据不丢失，一直同步，知道同步到那个点为止，a1,b1,c1表的数据在之前已经到处生成一份快照，只需要导入后开启同步即可。

1. 在slave机器上导入a1\_b1\_c1.sql，命令如下：

#MYSQL –uroot –p123456 <./a1\_b1\_c1.sql

1. 导入完毕后，开启同步即可：

Mysql>start slave;

这样就恢复了三张表，并且同步也修复了。

## 37.清除slave同步信息

命令：

Mysql>reset slave all;

# 备份与恢复

## 备份分类

按照备份到方式可分为三种，冷备份，热备份和逻辑备份；

冷备份：数据库处于关闭状态，能够较好地保证数据库到完整性；

热备份：数据库处于运行状态，这种备份方法依赖于数据库的日志文件；

逻辑备份：使用mysqldump命令从数据库中提取数据，并将结果写到一个文件上，文件内容为纯文本的SQL语句；

一般情况下，在生产环境中会将mysql配置为一主一从，为了避免影响业务，建议在slave机器上做备份。

## 冷备份

1.关闭mysql服务进程：

#/etc/init.d/mysql stop

2.把data数据目录(包含ibdata1)和日志目录(包含ib\_logfile0,ib\_logfile1,ib\_logfile2)复制到磁带机或者本地的另一块硬盘里；

3.用复制的数据目录和日志目录替换原有的目录；

4.重启mysql服务进程

## 逻辑备份

逻辑备份一般用于数据迁移或者数据量很小时，逻辑备份采用的是数据导出的备份方式。

1.备份过程：

如果需要导出所有数据库，命令如下：

mysqldump -q --single-transatcion -A >all.sql

如果只是要导出其中的某几个数据库，则采用如下命令：

mysqldump -q --single-transatction -B test1 test2 >test1\_test2.sql

如果要导出的是一个库中的几张表，可采用如下命令：

mysqldump -q --single-transaction -B test1 >test11.sql

2.恢复过程

MYSQL -UROOT -P123456 <all.sql

或者登录到mysql里，执行”source all.sql”

注意：

mysqldump增加了一个重要参数：

--dump-slave

使用该参数可在slave端dump数据，建立新的slave，其目的是为了防止对主库造成过大压力。

取代mysqldump的新工具mydumper,安装方法：

#wget <http://launchpadlibrarian.net/77098505/mydumper-0.5.2.tar.gz>

#yum install glib2-devel mysql-devel zlib-devel pcre-devel

#tar -xzvf mydumper-0.5.2.tar.gz

#cd mydumper-0.5.2

#cmake .

#make;make install

### 逻辑备份全量备份脚本

#!/bin/bash

#Mysql全量备份脚本，建议在slave从库上运行，并开启log\_slave\_updates=1

mkdir /backup

cd /backup

dateDIR='date+”%y-%m-%d”'

mkdir -p $dateDIR/data

path=/usr/local/mysql/data

for i in 'mysql -uroot -p123456 -e “show databases” |grep -v “Database”'

do

mysqldump -uroot -p123456 --default-character-set=utf8 \

-q --lock-all-tables --flush-logs -E -R --triggers -B $i | gzip >

/backup/$dateDIR/data/${i}\_${dateDIR}.sql.gz

done

binlog\_rm='tail -n 1 $path/mysql-bin.index | sed 's/.\///''

mysql -uroot -p123456 -e “purge binary logs to '$binlog\_rm'”

这个全量备份脚本，会在导出的时候锁住全局表，并刷新产生一个新的binlog，期间会有写操作等待，直到导出结束后才会写入新产生的binlog里，然后旧的binlog会被删除掉。一般该备份放在凌晨1:00操作。

### 逻辑备份增量备份脚本

#!/bin/bash

#mysql 增量备份脚本，建议在slave从库上运行，并开启log\_slave\_updates=1

cd /backup

dateDIR='date+”%y-%m-%d”'

mkdir -p $dateDIR/date

path=/usr/local/mysql/data

mysqladmin -uroot -p123456 flush-logs

binlog\_cp='head -n -1 $path/mysql-bin.index | sed 's/.\///''

for i in $binlog\_cp

do

mysql -uroot -p123456 -e “\!cp -p $path/$i/backup/$dateDIR/data/;”

done

binlog\_rm='tail -n 1 $path/mysql-bin.index |sed 's/.\///''

mysql -uroot -p123456 -e “purge binary logs to '$binlog\_rm'”

在执行全量备份脚本后，就可以执行这个增量备份脚本了，首先会刷新产生一个新的binlog，然后把之前有变化的binlog复制到备份目录下，复制完以后就会把之前的旧binlog删除掉，期间会有写操作，也会写入到新的binlog里。

## 热备份

xtrabackup中包含两个工具：

xtrabackup是用于热备份InnoDB及XtraDB表中数据的工具，不能备份其他类型的表，也不能备份数据表结构；

innobackupex是将xtrabackup进行封装的perl脚本，它提供了备份myisam表的能力，由于innobackupex的功能更为全面完善，所以一般选择innobackupex来进行备份

下面来看看xtrabackup的安装方法：

#wget <http://www.percona.com/redir/downloads/XtraBackup/XtraBackup-2.1.7/binary/Linux/x86_64/percona-xtrabackup-2.1.7-721-Linux-x86_64.tar.gz>

#tar -xzvf percona-xtrabackup-2.1.7-721-Linux-x86\_64.tar.gz

#cd percona-xtrabackup-2.1.7-Linux-x86\_64/bin

#ll

说明：

innobackupex 是要使用的备份工具；

xtrabackup 是被封装在innobackupex之中的，innobackupex运行时需要调用它

### 1.全量备份

命令如下：

innobackupex --user=root --password=123456 --defaults-file=/etc/my.cnf /back/

/back/: 是备份文件的存放位置

### 2.全量恢复

a.停止mysql数据库：#/etc/init.d/mysql stop

b.删除老数据库中的数据文件和事务日志文件；

c.将备份文件中的日志应用到备份文件中的数据文件上：

innobackupex --defaults-file=/etc/my.cnf –apply-log/back/2015-10-8\_15-44-55

d.将备份文件中的数据恢复到数据库中：

innobackupex --defaults-file=/etc/my.cnf --copy-back /back/2015-10-8\_15-44-55/

e.数据恢复完成后，需要修改相关文件的权限：

chown -R mysql.mysql /usr/local/mysql/data/

f.重启mysql

/etc/init.d/mysql start

### 备份到远程服务器

假设：

本地服务器ip：192.168.8.25

目标服务器ip：192.168.8.26

采用如下命令进行备份：

#innobackupex --defaults-file=/etc/my.cnf --stream=tar /usr/local/mysql/data | ssh

root@192.168.8.26 cat “>” /back/backup.tar

恢复：

进行全量恢复时，其操作步骤基本和签名的普通备份恢复类似

注意：恢复解压缩时，必须使用-i参数：

tar -ixvf backup.tar

解压后按普通备份恢复的步骤进行恢复即可。

### 增量备份

进行增量备份的前提是必须以及做过全量备份，步骤如下：

1先进行全量备份:

Innobackupex –defaults-file=/etc/my.cnf /bak/fullback/

2.再进行增量备份：

Innobackupex --defaults-file =/etc/my.cnf --incremental /bak/incrementbak

--incremental-basedir=/bak/fullback/2013-06-12\_16-46-36/

3.进入到备份目录，可以看到哪份是全量备份，哪份是增量备份：

#cd /bak/fullback/2013-06-12\_16-46-36/

#cat xtrabackup——checkpoints

备份成功后提示：

Innobackupex:Backup created in directory ‘/bak/incrementbak/2013-06-12\_16-59-27’

….

### 增量恢复

先恢复增量事物日志:

Innobackupex --apply-log –redo-only /bak/fullback/2013-06-12\_16-46-36/

--incremental-dir=/bak/incrementbak/2013-06-12\_16-59-27/

2.再恢复全量事物日志：

Innobackupex –apply-log /bak/fullback/2013-06-12\_16-46-36/

3.将备份文件中的数据恢复到数据库中：

Innobackupex --copy-back /bak/fullback/2013-06-12\_16-46-36/

4数据库恢复完成之后，需要修改相关文件的权限：

Chown –R mysql.mysql /usr/local/mysql/data/

启动mysql数据库：

/etc/init.d/mysql start

# 性能调优

## 1.表设计

第一范式：数据库表中的字段都是单一属性，不可再分；这个单一属性由基本类型构成，包括整形、字符型、逻辑型、日期型等；（联系方式可分为住宅电话和手机号是错误的）

第二范式：满足第一范式的前提下，要求实体的属性完全依赖于主关键字；（三人找同一个医生看病时，医生信息出现三次是错误的）

第三范式：满足第二范式的前提下，要求不存在非关键字段对任一候选关键字的传递函数以来。（科室名称依赖科室编号，科室编号以来着医生编号是错误的）

## 2.字段类型的选取

### 2.1数值类型

Mysql中支持5个主要整数类型是TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类型 | 字节 | 最小值(有符号/无符号) | | 最大值(有符号/无符号) | |
| TINYINT | 1 | -128 | 0 | 127 | 255 |
| SMALLINT | 2 | -32768 | 0 | 32767 | 65535 |
| MEDIUMINT | 3 | -8388608 | 0 | 8388607 | 16777215 |
| INT | 4 | -2147483648 | 0 | 2147483647 | 4294967295 |
| BIGINT | 8 | -9223372036854775808 | 0 | 有点大 | |

由上可知：最大个数依次为：3、5、8、10、19。

我们根据业务需求选取合适的类型。

1. 手机号可以采用整型来存储，但由于手机是11位数，所以只有bingint类型的字段才能存储手机号；
2. Ip地址也可以用int整型：

Myslq提供了一个函数：INET\_ATON(),它负责把IP地址转化为数字，而另一个函数INET\_NTOA()负责把数字转换为IP地址。（注意，需要更改表结构为无符号“unsigned”）

1. 根据需求选择最小整数类型

### 2.2.字符类型

Latin1（一个字符一个字节）、Gbk(一个字符2个字节)、UTF8(一个字符3个字节)；

Mysql表中，一条记录的长度最多为65535个字符，

1. 当选择Latin1字符集时，可以存储（65535-2）/2=32767.5；也就是说不能大于32767；
2. 当选择utf8字符集时，可以存储（65535-2）/3=21844；也就是说不能大于21844。

注意：当varchar类型小于255时，需要有一个字节用于存储长度，当varchar类型长度大于255时，需要2个字节存储值的长度。

根据业务需求选择使用char还是varchar，经常变化的值，如家庭住址，由于每个人地址都不同，有长有短，那么用varchar相对比较合适；而对于固定长度的值，如uuid函数，是数字和字母组成的36为长度的字符类型则建议char(32)。

2.3时间类型

在mysql中支持5个时间类型：DATE、TIME、DATETIME、TIMESTAMP和YEAR.

|  |  |  |
| --- | --- | --- |
| 数据类型 | 值 | 存储的字节 |
| Date | ‘0000-00-00 | 3字节 |
| Time | ‘00：00：00 | 3字节 |
| Datetime | ‘0000-00-00 00：00：00’ | 8字节 |
| Timestamp | ‘0000-00-00 00:00:00’ | 4字节 |
| Year | 0000 | 1字节 |

### 2.3快速修改表结构

如：有表t1，name字段设置为varchar(5),这时有变动，需要改为varchar（10）：

1. 创建一张临时表，和原表结构相同，命名为t1\_tmp；
2. 替换.frm表结构文件：

Mysql>flush tables with read lock;

Mysql>system cp /usr/local/mysql/data/book/t1\_tmp.frm /usr/local/mysql/data/book/t1.frm;

1. 插入数据测试
2. Mysql>show create table t1\G;

Pt-online-schema-change在线更改表结构：

这个工具解决了更改表时不锁表的问题，且不会影响到业务。

## 3.采用合适的锁机制

### Mysql的锁有以下几种

1. 表级锁：开销小，加锁快，不会出现死锁，锁定粒度大，发生锁冲突的概率最高，并发度最低，myISAM引擎属于这种类型；
2. 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高，innodb引擎属于这种类型。
3. 页面锁：开销和加锁时间介于表锁和行锁之间，会出现死锁；锁定粒度介于表锁和行锁之间，并发度一般，NDB属于这种类型。

### 表锁的演示

Myisam存储引擎只支持表锁，所以对myisam表进行操作，会存在以下情况：

对myisam表的读操作（加读锁），不会阻塞其他进程对同一表的读请求，但会阻塞对同一表的写请求。只有当读锁释放后，才会执行其他进程的写操作。

对myisam表的写操作（加写锁），会阻塞其他进程对同一表的读和写操作，只有写操作释放后，才会执行其他进程的读写操作。

### 行锁的演示

Innodb存储引擎是通过给索引上的索引项加锁来实现的，这就意味着：只有通过索引条件检索数据，innodb才会使用行级锁，否则仍使用表锁。

### Innodb引擎和myisam引擎的性能对比

区别：

表结构：每张myisam表存放在3个文件中：frm文件存放表格定义，数据文件是MYD(MYData)，索引文件是MYI(MYIndex)；

1. Myisam是非事物安全型的，而innodb是事物安全型的，也就是ACID事物支持；
2. Myisam锁匙表级锁，锁开销最小，而innodb支持行级锁定，锁管理开销大，支持更好的并发写操作；
3. Myisam支持全文索引，而innodb在5.6中也提供了；
4. Myisam简单，管理方便；
5. Myisam是保存成文件的格式，易于跨平台；
6. Innodb表更安全，可以保证事物在不丢失数据的情况下，切换非事物表到事物表

### Innodb工作原理

把数据捞到内存，被用户读写，这样大大增加了性能

## 4事物隔离级别

### 4.1事物的概念

事物处理可以确保事物性单元内的所有操作都成功完成，一个逻辑工作单元要成为事物，必须满足所谓的ACID(原子性、一致性、隔离性和持久性)。

原子性：对于数据修改，要么全都执行，要么全都不执行；

隔离性：在所有的操作没有执行完毕之前，其他会话不能够看到中间改变的过程；

一致性：事物发生前和发生后，根据数据的规则，总额应该匹配；

持久性：事物发生前和发生后，根据数据的规则，总额应该匹配。

### 4.2事物的实现

Mysql在进行事物处理的时候使用的是日志先行的方式来保证事物可快速和持久运行的，也就是在写数据前，需要先写日志。当开始一个事物时，会记录该事物的一个LSN日志序列号；当执行事务时，会往InnoDB\_Log\_Buffer日志缓冲区里插入事物日志(redo log);当事物提交时，会将日志缓冲区里的事物日志刷入磁盘。这个动作是有innodb\_flush\_log\_at\_trx\_commit这个参数控制的。

Innodb\_flush\_log\_at\_trx\_commit=0;表示每个事物提交时，每隔一秒，把事物日志缓冲区的数据写到日志文件中，以及把日志文件的数据刷新到磁盘上；它的性能是最好的，同样安全性也是最差的。当系统宕机时，会丢失1秒钟的数据。

Innodb\_flush\_log\_at\_trx\_commit=1,表示每个事物提交时，把事物日志从缓存区写到日志文件中，并且刷新日志文件的数据到磁盘上。

Innodb\_flush\_log\_at\_trx\_commit=2,表示每个事物提交时，把事物日志数据从缓冲区写到日志文件中，每隔一秒刷新一次日志文件，但不一定刷新到磁盘上，而是取决于操作系统的调度。

### 4.3 4个事物隔离级别

数据库是要被广大客户所共享的，那么数据库操作过程中很可能出现以下几种情况：

更新丢失：两个事物更新同一行，但是第二个事物却途中失败退出，导致对数据的两个修改都失效了。这是因为系统没有执行任何的锁操作，因此并发事务并没有被隔离出来；

脏读：一个事物开始读取了某行数据，另外一个事物已经更新了此数据但没有能够及时提交；

不可重复读：一个事物对同一行数据重复读取两次，但是却得到不同的结果；

两次更新问题：无法重复读取的特例。有两个并发事务同时读取同一行数据，然后其中一个对它进行修改提交，而另一个也进行了修改提交。这就会造成第一次写操作失效。

幻读：事物在操作过程中进行两次查询，第二次查询的结果过包含了第一次查询中未出现的数据（这是因为在两次查询过程中有另外一个事物插入数据）。

为了避免出现上面的几种情况，定义了4个事物隔离级别，不同的隔离级别对事物的处理不同：

1. 读未提交：运行脏读取，但不运行更新丢失。如果一个事物已经开始写数据，则另外一个事物则不运行同时进行写操作，但运行其他事物读此行数据。该隔离级别可以通过“排他写锁”实现；
2. 读提交：允许不可重复读取，但不允许脏读取。读取数据的事物允许其他事物继续访问该行数据，但是未提交的写事物将会禁止其他事物访问该行。
3. 可重复读取：禁止不可重复读取和脏读取，但是有时可能出现幻影数据。读取数据的事物将会禁止写事物（但允许读事物），写事物则禁止任何其他事物。
4. 序列化:提供严格的事物隔离。事物只能一个接着一个地执行。

## 5SQL优化

### 5.1开启慢日志功能：

在my.cnf中加入参数：

Slow\_query\_log=1

Slow\_query\_log\_file=mysql.slow

Long\_query\_time=2(超过2秒的sql会记录下来)

### 5.2 not in子查询优化

生产环境中，应尽量避免使用子查询，可用left join表链接取代之

### 5.3模式匹配 like

在mysql里，like ‘xxx%’ 可以用到索引，但like ‘%xxx%’却不行

### 5.4limit分页优化

优化前：select \* from test1 order by id limit 99999,10;

优化后：select \* from test1 where id>=100000 order by id limit 10;

### 5.5count(\*)统计数据如何加快速度

#### 5.5.1count(辅助索引)快于count(\*)

Select count(\*) from user;

Select count(\*) from user where id>0;(使用id作为辅助索引)

#### 5.5.2Count(distinct)优化

优化distinct最有效的方法是利用索引来做排重操作，先把排重的记录查找出来再通过count统计，这样效果更高

Select count(distinct k)from test;

Select count(\*) from (select distinct k from test)tmp;

### 5.6 0R条件优化

在sql语句里有or条件，则不会用到索引；

Select \* from user where name =‘d’ or age=44;

上面的语句查询的结果集是全表扫描。效率低、且用不到索引,可以改为如下：

Select \* from user where name=’d’

Union all

Select \* from user where age=41;

### 5.7使用ON DUPLICATE KEY UPDATE子句

Mysql中有一种非常高效的主键冲突处理判断，冲突则执行update，不冲突则执行insert。

### 5.8不必要的排序

对于sql用于统计字表的记录的条数的sql语句来说，排序是没有必要的，而且消耗性能。

Insert into user(id,contacteid,isbuddy,ischat) values(1,’sdf’,1,1) ON DUPLICATE KEY UPDATE isbuddy=1,ischat=1;

### 5.9用where子句替换having子句

避免使用having子句，having只会检索出所有记录之后才对结果集进行过滤。这个需要排序、总计等操作。如果能通过where子句限制记录的数目，那就能减少这方面的开销；

一般情况下，having子句中的条件用于对一些集合函数的比较，如count()等。否则都应该写在where子句中。

### 5.10 合理使用索引

a.当我们执行查询的时候，mysql只能使用一个索引，如果查询条件中有多个字段都做了索引，此时mysql只会选择它认为最优的索引；

b.当我们查询条件有多个字段时，可以考虑做联合索引；

c.字段使用函数，将不能用到索引:

select createtime from aa where date(createtime)=curdate();

select createtime from aa where createtime >DATE\_FORMAT(curdate(),‘%Y-%m-%d’);

注意：1.后一条sql语句性能优于第一条，因为前一条语句进行了全表扫描；

2.MYSQL目前还不支持函数索引；

3.当取出的数据量超过表中数据的20%，优化器就不会使用索引，而是全表扫描；

4.如果索引的字段是字符型，那么where条件必须要加上引号，否则不能使用该索引

## my.cnf配置文件优化

在mysql数据库性能调优中，首先要考虑的就是schema设计，这一点非常重要，一个糟糕的schema设计即使实在性能强劲的服务器上运行也会表现出很差的性能。和schema相似，查询语句的设计也会影响mysql的性能，应该避免写出低效的sql查询。最后考虑的就是参数的优化，mysql数据库的默认设置性能非常差，仅仅起一个功能测试的作用。不能再生产环境中运行，因此要对一些参数进行调整。比如，无法使用索引的情况下进行全表扫描、全索引扫描等。在这种时候，mysql会按照数据的存储顺序依次读取数据块，每次读取的数据块首先会暂存在read\_buffer\_size中，当buffer空间被写满或者全部数据读取结束后，再将buffer中的数据返回给上层调用者，以提高效率。

### Per\_thread\_buffer优化

对于per\_thread\_buffers,可以将其理解为oracle的PGA，为每个链接到mysql的用户进程分配的内存。其包含如下几个参数：

#### read\_buffer\_size

该参数用于表的顺序扫描，表示每个线程分配的缓冲区大小。比如，在进行全表扫描时，mysql会按照数据的存储顺序依次读取数据块，每次读取的数据块首先会暂存在read\_buffer\_size中，当buffer空间被写满或者全部数据读取结束后，再将buffer中的数据返回给上层调用者，以提高效率。默认为128KB.这个参数不要设置过大，一般在128~256KB即可。

#### read\_rnd\_buffer\_size

该参数用于表的随机读取，表示每个线程分配的缓冲区大小，比如，按照一个非索引字段做order by排序操作时，就会利用这个缓冲区来暂存读取的数据。默认为256KB。

#### sort\_buffer\_size

在表进行order by和group by排序操作时，由于排序的字段没有索引，会出现Using filesort，为了提高性能，可用此参数增加每个线程分配的缓冲区大小。默认为2MB.这个参数不要设置过大，一般在128~256KB即可。一般出现Using filesort的时候，要通过增加索引来解决。

#### thread\_stack

该参数表示每个线程的堆栈大小。默认为192KB.如果是64位操作系统，设置为256KB即可，这个参数不要设置过大。

#### join\_buffer\_size

表进行join连接操作时，如果关联的字段没有索引，会出现using join buffer，为了提高性能，可用此参数增加每个线程分配的缓冲区大小。默认为128KB。这个参数不熬设置过大。一般在128~256KB即可。一般出现using join buffer的时候，要通过增加索引来解决。

#### binlog\_cache\_size

一般来说，如果数据库中没有什么大事物，写入也不是特别频繁，将其设置为1~2MB是一个合适的选择。如果有很大的事物，可以适当增加这个缓存值，以获得更好的性能。

#### max\_connections

该参数用来设置最大链接数，默认为100.一般设置为512~1000即可。

Per\_thread\_buffers内存的计算公式为以上七个相加，乘以连接数

### global\_buffers优化

对于global\_buffers，可以理解为Oracle的SGA，用于在内存中缓存从数据文件中检索出来的数据块，可以大大提高查询和更新数据的性能。主要由以下几个参数组成：

#### innodb\_buffer\_pool\_size

这个参数是innodb存储引擎的核心参数，默认为128MB,这个参数要设置为物理内存的60%~70%。

#### innodb\_additional\_mem\_pool\_size

该参数用来存储数据字典信息和其他内部数据结构。表越多，需要在这里分配的内存越多。如果innodb用光了这个池内的内存，innodb开始从操作系统分配内存，并且往mysql错误日志中写警告信息，默认值是8MB，当发现错误日志中已经有相关的警告信息时，就应该适当加大该参数的大小。一般设置为16MB.

#### innodb\_log\_buffer\_size

事物日志所用的缓冲区。Innodb在写事物日志的时候，为了提高性能，先将信息写入innodb 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）时，再将日志写到文件（或者同步到磁盘）中。可以通过innodb\_log\_buffer\_size参数设置其可以使用的最大内存空间。默认是8MB，一般设置为16~64MB即可。

#### key\_buffer\_size

该参数用来缓存myisam存储引擎的索引参数。Mysql5.5默认为innodb存储引擎，所以这个参数可以设置小一些，64MB即可。

#### query\_cache\_size

缓存select语句和结果集大小的参数。

注意per\_thread\_buffer内存设置+global\_buffers设置不能大雨实际物理内存，否则当发生并发量很高时会造成内存溢出，系统死机。

### Query cache在不同环境下的使用

Query cache的功能就是缓存select语句和结果集。查询缓存会存储一个select查询的文本与被传送到客户端的相应结果。如果之后接收到一个同样的查询，服务器将会从查询缓存中检索结果，而不是再次分析和执行这个同样的查询。

注意：查询缓存绝不返回过期数据。当数据被修改后，在查询缓存中的任何相关的词条均会被清除。如果有些表并不经常更改，又要经常执行大量相同的查询时，查询缓存将是非常有用的。

1. 如果环境中写操作很少，读操作很频繁，那么打开query\_cache\_type=1,会对性能有明显提升。
2. 如果环境中写操作很多，那么就不合适打开它了，因为每当表的内容更改过，query

Cache缓存的结果集就要随之刷新，频繁的刷新操作反而会使性能降低很多，在这种情况下，就要关闭它（query\_cache\_type=0）,同时设置query\_cache\_size=0.

Innodb数据页16KB和8KB性能对比测试

从5.6开始，一个新参数innodb\_page\_size可以设置innodb数据页为8KB、4KB.默认为16KB.这个参数在一开始初始化时就要加入到my.cnf中，如果已经创建了表，再 修改，启动mysql就会报错。

注意：针对读写很频繁的情况，16KB目前是性能比较好的。

# My.cnf配置案例

下面是72GB内存生产环境下my.cnf配置文件。可以作为一个优化参考：

#mysql configuration for 72G memory

[client]

port=3306

socket=/tmp/mysql.sock

#The MySQL server

##################basic############

[mysqld]

server-id=22

port=3306

user=mysql

basedir=/usr/local/mysql

datadir=/mysqlData/data

tmpdir=/mysqlData/tmp

socket=/tmp/mysql.sock

skip-external-locking

skip-name-resolve

default-storage-engine=INNODB

character-set-server=utf8

wait\_timeout=100

connect\_timeout=20

interactive\_timeout=100

back\_log=500

myisam\_recover

event\_scheduler=ON

##############binlog###############

log-bin=/mysqlLog/logs/mysql-bin

binlog\_format=row

max\_binlog\_size=128M

binlog\_cache\_size=2M

expire-log-days=5

###################relication#############

slave-net-timeout=10

rpl\_semi\_sync\_master\_enabled=1

rpl\_semi\_sync\_master\_wait\_no\_slave=1

rpl\_semi\_sync\_master\_timeout=1000

rpl\_semi\_sync\_slave\_enabled=1

skip-slave-start

log\_slave\_updates=1

relay\_log\_recovery=1

#########slow log##############

slow\_query\_log=1

slow\_query\_log\_file=/mysqlLog/logs/mysql.slow

long\_query\_time=2

########error log############

log-error=/mysqlLog/logs/error.log

#########per\_thread\_buffers############

max\_connections=1024

max\_user\_connection=1000

max\_connect\_errors=10000

key\_buffer\_size=64M

table\_allowed\_packet=128M

table\_cache=3096

table\_open\_cache=6144

table\_definition\_cache=4096

sort\_buffer\_size=512K

read\_buffer\_size=512K

read\_rnd\_buffer\_size=512K

join\_buffer\_size=512K

tmp\_table\_size=64M

max\_heap\_table\_size=64M

query\_cache\_type=0

query\_cache\_size=0

bulk\_insert\_buffer\_size=32M

thread\_cache\_size=64

thread\_concurrency=32

thread\_stack=256K

####################innodb##############

innodb\_data\_home\_dir=/mysqlData/data

innodb\_log\_group\_home\_dir=/mysqlLog/logs

innodb\_data\_file\_path=ibdata1:2G:autoextend

innodb\_buffer\_pool\_size=50G

innodb\_buffer\_pool\_instances=8

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

innodb\_log\_file\_size=1024M

innodb\_log\_buffer\_size=64M

innodb\_log\_file\_in\_group=3

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

innodb\_lock\_wait\_timeout=10

innodb\_sync\_spin\_loops=40

innodb\_support\_xa=1

innodb\_max\_dirty\_pages\_pct=90

innodb\_thread\_concurrency=0

innodb\_thread\_sleep\_delay=500

innodb\_file\_io\_threads=4

innodb\_concurrency\_tickets=1000

log\_bin\_trust\_function\_creators=1

innodb\_flush\_method=O\_DIRECT

innodb\_file\_per\_table

innodb\_read\_io\_threads=16

innodb\_write\_io\_threads=16

innodb\_io\_capacity=2000

innodb\_file\_format=Barracuda

innodb\_purge\_threads=1

innodb\_purge\_batch\_size=32

innodb\_old\_blocks\_pct=75

innodb\_change\_buffering=all

transaction\_isolation=READ-COMMITED

[mysqldump]

quick

max\_allowed\_packet=128M

myisam\_max\_sort\_file\_size=10G

[mysql]

key\_buffer\_size=64M

sort\_buffer\_size=256k

read\_buffer=2M

write\_buffer=2M

[mysqlhotcopy]

interactive-timeout

[mysqld\_safe]

Open-file-limit=28192

附录：

1.增加自适应刷新脏页功能。（默认开启）

2.数据恢复时间加快

3.自适应哈希索引

哈希（hash）是一种非常快的查找方法，在一般情况下这种查找的时间复杂度为O(1)，即一般仅需要一次查找就能定位数据。而B+树的查找次数，取决于B+树的高度，在生产环境中，B+树的高度一般为3～4层，故需要3～4次的查询。

InnoDB存储引擎会监控对表上各索引页的查询。如果观察到建立哈希索引可以带来速度提升，则建立哈希索引，称之为自适应哈希索引 （Adaptive Hash Index，AHI）。AHI是通过缓冲池的B+树页构造而来，因此建立的速度很快，而且不需要对整张表构建哈希索引。InnoDB存储引擎会自动根据访 问的频率和模式来自动地为某些热点页建立哈希索引。

3.sysbench的安装：

下载sysbench：

[sysbench-0.4.10.tar.gz [461461 bytes] - Download Mirrors](http://www.filewatcher.com/m/sysbench-0.4.10.tar.gz.461461-0.html)

#tar zxvf Sysbench-0.4.8.tar.gz

#cd sysbench-0.4.8.tar.gz

#./configure --with-mysql-includes=/usr/local/mysql/include –with-mysql-libs=/usr/local/mysql/lib

#make && make install

4.使用多个回滚提升性能

5.改善清除程序进度

可通过设置参数：innodb\_purge\_threads来配置清除操作是否使用单独到线程，默认情况下设置为0，不使用单独线程，设置为1时，表示使用单独到清除线程。

当设置为1时：

需要结合innodb\_purge\_batch-size参数来使用，默认值是20，最大可设置为5000.建议默认

6.控制自旋锁Spin Lock轮训间隔

与互斥锁到区别：如果资源已经被占用，申请者只能进入睡眠状态

互斥锁： 如果资源已经被占用，不会引起调用者睡眠，而是调用者一直在看该自旋锁到保持者是否已经释放了锁。

为了防止自旋锁循环过快耗费cpu，引入参数：

set global innodb\_spin\_wait\_delay=6;

即：间隔6秒后再循环一次

任何时候都只有一个执行单元获得锁。

7.快速创建、删除、更改索引

8.复制功能加强