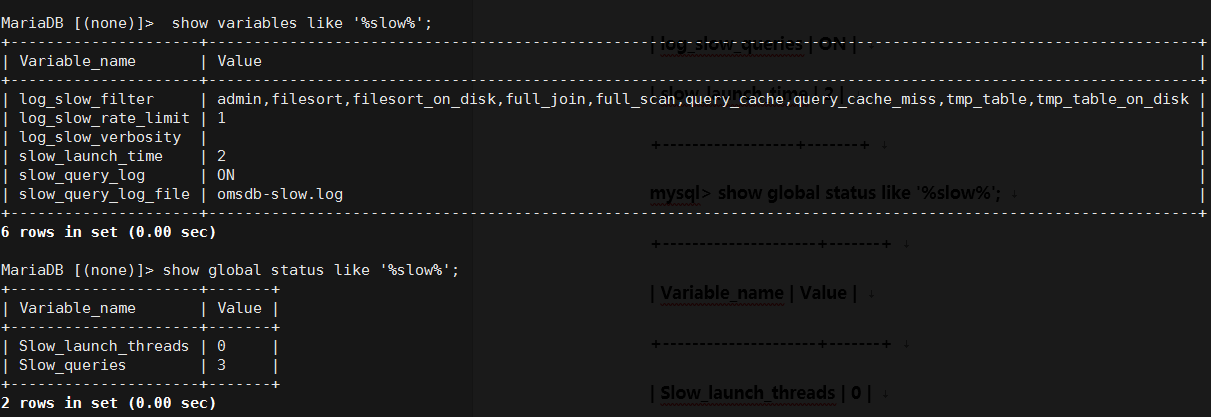
MYSQL性能监控指标分析

监控报告生成方法

|  |
| --- |
| 1. cd /home/stress/mysql\_monior 2. 执行./mysql\_monior.sh 2 3 3. 在同级目录中查看mysql\_monior\_result.html   以上命令中2表示取样间隔（单位为秒），3表示取样次数 |

show global status;可以列出MySQL服务器运行各种状态值，另外，查询MySQL服务器配置信息语句：show variables;

# 慢查询



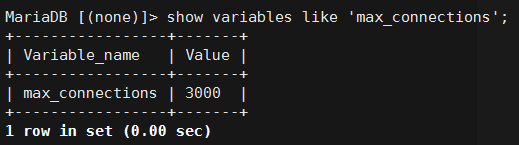
配置中打开了记录慢查询，执行时间超过2秒的即为慢查询，系统显示有3个慢查询，可以分析慢查询日志，找出有问题的SQL语句，慢查询时间不宜设置过长，否则意义不大，最好在5秒以内，如果你需要微秒级别的慢查询，可以考虑给[MySQL](http://lib.csdn.net/base/mysql)打补丁：<http://www.percona.com/docs/wiki/release:start>，记得找对应的版本。

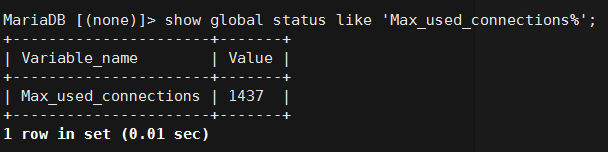
打开慢查询日志可能会对系统性能有一点点影响，如果你的MySQL是主-从结构，可以考虑打开其中一台从服务器的慢查询日志，这样既可以监控慢查询，对系统性能影响又小。

开启慢查询方法：

|  |
| --- |
| **1、修改配置文件（需重启服务）**  log-slow-queries 慢查询日志文件路径  long\_query\_time 超过多少秒的查询就写入日志   * Linux：打开my.cnf配置文件，加入以下代码：   log-slow-queries = /tmp/mysql-slow.log  long\_query\_time = 2   * windows：打开my.ini配置文件，加入以下代码：   log\_slow\_queries  long\_query\_time = 2    **2、执行数据库命令**  mysql> set global slow\_query\_log=ON;  Query OK, 0 rows affected (0.03 sec)    mysql> set global slow\_launch\_time=2;  Query OK, 0 rows affected (0.00 sec) |

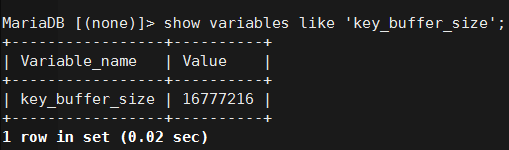
# 连接数

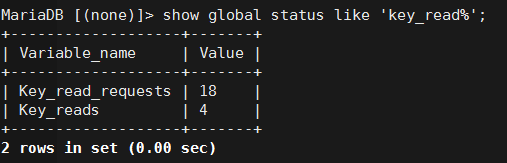
经常会遇见”MySQL: ERROR 1040: Too many connections”的情况，一种是访问量确实很高，MySQL服务器抗不住，这个时候就要考虑增加从服务器分散读压力，另外一种情况是MySQL配置文件中max\_connections值过小：

这台MySQL服务器最大连接数是3000，然后查询一下服务器响应的最大连接数： 　　MySQL服务器过去的最大连接数是1437，没有达到服务器连接数上限3000，应该没有出现1040错误，比较理想的设置是：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，MySQL服务器连接数上限设置的过高了。

# Key buffer命中率

key\_buffer\_size是对MyISAM表性能影响最大的一个参数，下面一台以MyISAM为主要存储引擎服务器的配置： 

分配了16MB内存给key\_buffer\_size

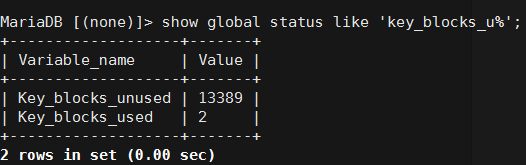
一共有18个索引读取请求，有4个请求在内存中没有找到直接从硬盘读取索引，计算索引未命中缓存的概率：key\_cache\_miss\_rate = 4 / 18 \* 100%；比如上面的数据，key\_cache\_miss\_rate为22.2%，说明缓存命中率较低，可适当增加key\_buffer\_size的值。

key buffer 命中率代表了myisam类型表的索引cache命中率，命中率的大小直接影响myisam类型表的读写性能。key buffer 命中率实际上包括读命中率和写命中率两种，mysql中并没有直接给出这两个命中率的值，但是可以通过如下方式计算：

key\_buffer\_read\_hits=(1-key\_reads/key\_read\_requests) \* 100%

key\_buffer\_write\_hits=(1-key\_writes/key-write-requests)\*100%

命中率过低，说明myisam类型表的读写存在问题。

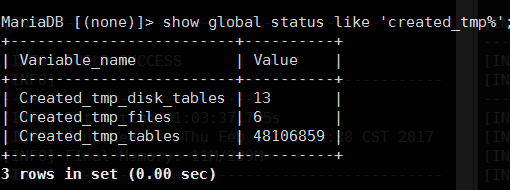
MySQL服务器还提供了key\_blocks\_\*参数：

Key\_blocks\_unused表示未使用的缓存簇(blocks)数，Key\_blocks\_used表示曾经用到的最大的blocks数，比较理想的设置：

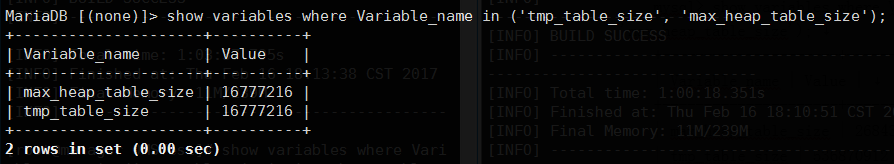
Key\_blocks\_used / (Key\_blocks\_unused + Key\_blocks\_used) \* 100% ≈ 80%

# 临时表

tmp table 主要用于监控mysql使用临时表的量是否过多，是否有临时表过大而不得不从内存中换出到磁盘文件中，临时表使用状态 信息获取如下：



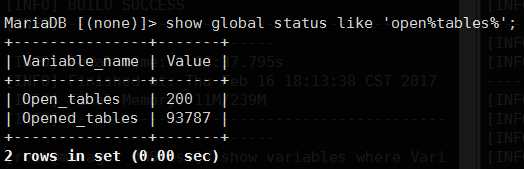
Created\_tmp\_disk\_tables为临时表过大无法在内存中完成，而不得不使用磁盘的次数。若create\_tmp\_tables非常大，则可甬系统排序操作过多，或者可能是连接方式不是很优化。而如果是create\_tmp\_dis\_table/create\_tmp\_tables比率过高，如超过10%，则需要考虑tmp\_table\_size参数是否需要调整大些。建议tmp\_table\_size与max\_heap\_table\_size需要设置成一样大。

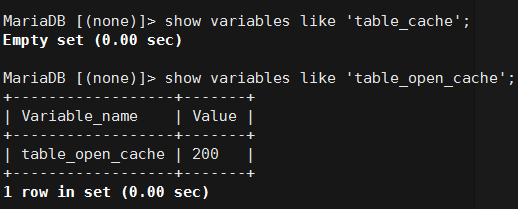
我们再看一下MySQL服务器对临时表的配置：只有16MB以下的临时表才能全部放内存，超过的就会用到硬盘临时表。

# Open Table情况

table\_cache(5.1.3之后这个值叫做table\_open\_cache)指定表调整缓存的大小，当mysql访问某个表时，若表缓存空间还有空间，则将该表就被打开并将数据放入其中，下次访问此表时可以更快的访问表的内容。通过查峰值时间的状态值open\_tables 和 opened\_tables可以决定是否需要增加table\_cache值。需要注意的是table\_cache设置很太高，可能会造成文件描述符不足，从而造成性能不稳定或是连接失败。

table\_cache的当前状态量可以帮助我们判断系统参数table\_open\_cache的设置是否合理。



Open\_tables表示打开表的数量，Opened\_tables表示打开过的表数量，如果Opened\_tables数量过大，说明配置中table\_cache值可能太小，我们查询一下服务器table\_cache值：

比较合适的值为：

Open\_tables / Opened\_tables \* 100% >= 85%

Open\_tables / table\_cache \* 100% <= 95%

# Thread使用情况

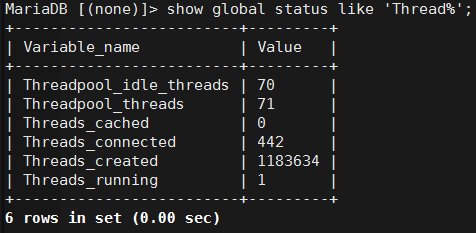
在mysql中，为了尽可能提高客户端连接的过程，实现 了一个thread cache池，将空闲的连接线程存放在其中，而不是请求完成后销毁，当有新的连接请求的时候，mysql首先检查thread cache是否存储空闲的连接线程，如果存在则取出来直接使用，如果没有空闲连接线程，才创建新的线程。

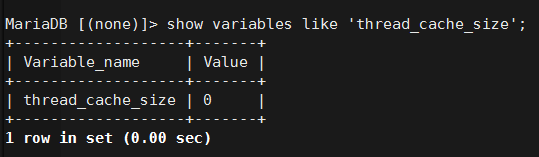
thread cache命中率能直接反应出系统参数thread\_cache\_size设置是否合理。一个合理的thread\_cache\_size参数能够节约大量创建新连接时所需要消耗的资源。

thread cache命中率计算方式如下：

thread\_cache\_hits = (1- threads\_created/connections) \* 100 %;

正常来说，thread cache命中率在90%以上才算合理。

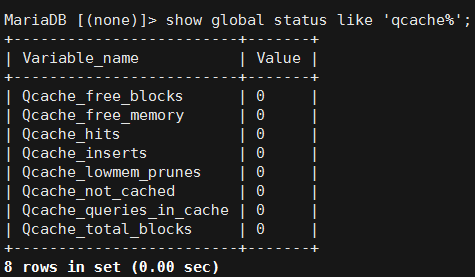


Threads\_created表示创建过的线程数，如果发现Threads\_created值过大的话，表明MySQL服务器一直在创建线程，这也是比较耗资源，可以适当增加配置文件中thread\_cache\_size值，查询服务器thread\_cache\_size配置：

# 查询缓存(query cache)

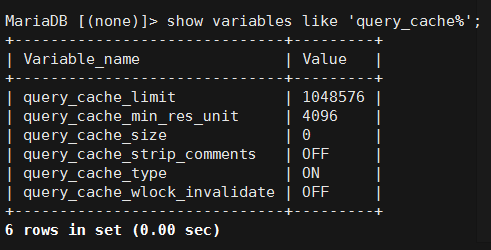
query cache 是mysql的查询cache，在my.cnf配置文件若打开，则可以对查询过的语句结果进行cache。对于一些用户数不高或一次性统计平台建议关闭查询缓存。若开启query cache，则对query cache 命中率进行监控也是需要的，它可以告诉我们是数据库是否在正确使用query cache。query cache命中率计算如下：

query\_cache\_hits =(Qcache\_hits/(Qcache\_hits+ Com\_select))\* 100%;



MySQL查询缓存变量解释：

* Qcache\_free\_blocks：缓存中相邻内存块的个数。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，从而得到一个空闲块。
* Qcache\_free\_memory：缓存中的空闲内存。
* Qcache\_hits：每次查询在缓存中命中时就增大
* Qcache\_inserts：每次插入一个查询时就增大。命中次数除以插入次数就是不中比率。
* Qcache\_lowmem\_prunes：缓存出现内存不足并且必须要进行清理以便为更多查询提供空间的次数。这个数字最好长时间来看;如果这个数字在不断增长，就表示可能碎片非常严重，或者内存很少。(上面的 free\_blocks和free\_memory可以告诉您属于哪种情况)
* Qcache\_not\_cached：不适合进行缓存的查询的数量，通常是由于这些查询不是 SELECT 语句或者用了now()之类的函数
* Qcache\_queries\_in\_cache：当前缓存的查询(和响应)的数量
* Qcache\_total\_blocks：缓存中块的数量。

我们再查询一下服务器关于query\_cache的配置： 

各字段的解释：

* query\_cache\_limit：超过此大小的查询将不缓存
* query\_cache\_min\_res\_unit：缓存块的最小大小
* 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\_invalidate：当有其他客户端正在对MyISAM表进行写操作时，如果查询在query cache中，是否返回cache结果还是等写操作完成再读表获取结果
* 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](http://lib.csdn.net/base/hadoop)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%

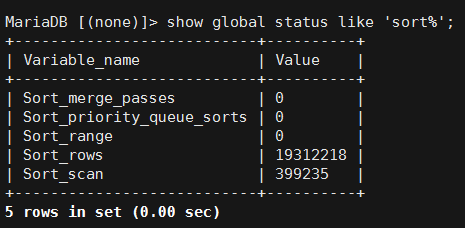
如果查询缓存碎片率超过20%，可以用FLUSH QUERY CACHE整理缓存碎片，或者试试减小query\_cache\_min\_res\_unit，如果你的查询都是小数据量的话。

查询缓存利用率 = (query\_cache\_size - Qcache\_free\_memory) / query\_cache\_size \* 100%

查询缓存利用率在25%以下的话说明query\_cache\_size设置的过大，可适当减小;查询缓存利用率在80%以上而且Qcache\_lowmem\_prunes > 50的话说明query\_cache\_size可能有点小，要不就是碎片太多。

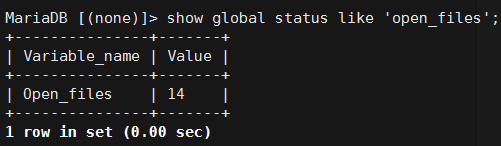
查询缓存命中率 = Qcache\_hits/(Qcache\_hits + Com\_select) \* 100%

# 排序使用情况

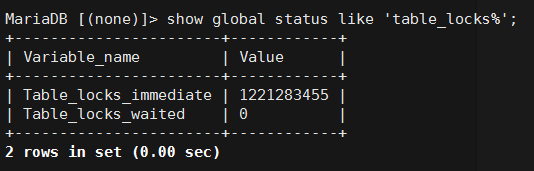


Sort\_merge\_passes 包括两步。MySQL 首先会尝试在内存中做排序，使用的内存大小由系统变量 Sort\_buffer\_size 决定，如果它的大小不够把所有的记录都读到内存中，MySQL 就会把每次在内存中排序的结果存到临时文件中，等 MySQL 找到所有记录之后，再把临时文件中的记录做一次排序。这再次排序就会增加 Sort\_merge\_passes。实际上，MySQL 会用另一个临时文件来存再次排序的结果，所以通常会看到 Sort\_merge\_passes 增加的数值是建临时文件数的两倍。因为用到了临时文件，所以速度可能会比较慢，增加 Sort\_buffer\_size 会减少 Sort\_merge\_passes 和 创建临时文件的次数。但盲目的增加 Sort\_buffer\_size 并不一定能提高速度，见 How fast can you sort data with MySQL?(引自<http://qroom.blogspot.com/2007/09/mysql-select-sort.html>，貌似被墙) mysql  
 　　另外，增加read\_rnd\_buffer\_size(3.2.3是record\_rnd\_buffer\_size)的值对排序的操作也有一点的好处，参见：<http://www.mysqlperformanceblog.com/2007/07/24/what-exactly-is-read_rnd_buffer_size/>

# 文件打开数(open\_files)

      比较合适的设置：Open\_files / open\_files\_limit \* 100% <= 75%

# 表锁情况



Table\_locks\_immediate表示立即释放表锁数，Table\_locks\_waited表示需要等待的表锁数，如果Table\_locks\_immediate / Table\_locks\_waited > 5000，最好采用InnoDB引擎，因为InnoDB是行锁而MyISAM是表锁，对于高并发写入的应用InnoDB效果会好些。

mysql的锁有表锁和行锁，myisam最小锁为表锁，innodb最小锁为行锁，可以通过以下命令获取锁定次数、锁定造成其他线程等待次数，以及锁定等待时间信息。

mysql> show status like '%lock%';

+------------------------------------------+---------+

| Variable\_name                            | Value   |

+------------------------------------------+---------+

| Com\_lock\_tables                          | 0       |

| Com\_unlock\_tables                        | 0       |

| Innodb\_row\_lock\_current\_waits            | 0       |

| Innodb\_row\_lock\_time                     | 0       |

| Innodb\_row\_lock\_time\_avg                 | 0       |

| Innodb\_row\_lock\_time\_max                 | 0       |

| Innodb\_row\_lock\_waits                    | 0       |

| Key\_blocks\_not\_flushed                   | 0       |

| Key\_blocks\_unused                        | 13396   |

| Key\_blocks\_used                          | 19      |

| Performance\_schema\_locker\_lost           | 0       |

| Performance\_schema\_rwlock\_classes\_lost   | 0       |

| Performance\_schema\_rwlock\_instances\_lost | 0       |

| Qcache\_free\_blocks                       | 0       |

| Qcache\_total\_blocks                      | 0       |

| Table\_locks\_immediate                    | 1570736 |

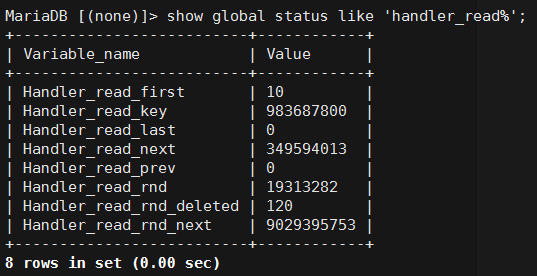
| Table\_locks\_waited                       | 7294    |

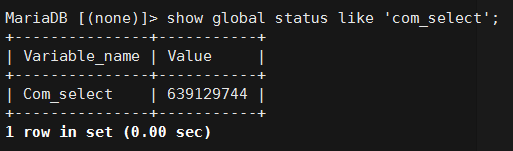
+------------------------------------------+---------+

如当Table\_locks\_waited与Table\_locks\_immediate的比值较大，则说明我们的表锁造成的阻塞比较严重，可能需要调整Query语句，或者更改存储引擎，亦或者需要调整业务逻辑。当然，具体改善方式必须根据实际场景来判断。而Innodb\_row\_lock\_waits较大，则说明Innodb的行锁也比较严重，且影响了其他线程的正常处理。同样需要查找出原因并解决。造成Innodb行锁严重的原因可能是Query语句所利用的索引不够合理（Innodb行锁是基于索引来锁定的），造成间隙锁过大。也可能是系统本身处理能力有限，则需要从其他方面来考虑解决。

|  |
| --- |
| 查看MySQL数据库的死锁日志  1、输入命令show engine innodb status \G;  2、在打印出来的信息中找到“LATEST DETECTED DEADLOCK”一节内容  3、分析死锁原因 |

# 表扫描情况



各字段解释参见<http://hi.baidu.com/thinkinginlamp/blog/item/31690cd7c4bc5cdaa144df9c.html>，调出服务器完成的查询请求次数

计算表扫描率：Handler\_read\_rnd\_next / Com\_select

如果表扫描率超过4000，说明进行了太多表扫描，很有可能索引没有建好，增加read\_buffer\_size值会有一些好处，但最好不要超过8MB。

# 其他：

mysql数据库的性能状态监控点非常多，其中很多量都是不能忽视的必须监控的量，且90%以上的内容 可以在连接上mysql后执行show status 或是 show veriables的输出值 获得，需要注意的是以上的命令获得的状态值实际上是累计值，所以如果 要计算时段内的变化 量还需要稍加处理，下面看下几项需要重点关注的性能状态：

## innodb buffer 命中率

这里innodb buffer 所指的是innodb\_buffer\_pool，也就是用来缓存innodb类型表和索引的内在空间。类似key buffer，同样可以根据mysql提供的相应的状态信息计算其命中率：

innodb\_buffer\_read\_hits=(1-innodb\_buffer\_pool\_reads/innodb\_buffer\_pool\_read\_requests) \* 100%;

获取所需状态变量值：

mysql> show status like 'innodb\_buffer\_pool\_read%';

+---------------------------------------+----------+

| Variable\_name                         | Value    |

+---------------------------------------+----------+

| Innodb\_buffer\_pool\_read\_ahead\_rnd     | 0        |

| Innodb\_buffer\_pool\_read\_ahead         | 0        |

| Innodb\_buffer\_pool\_read\_ahead\_evicted | 0        |

| Innodb\_buffer\_pool\_read\_requests      | 27269024 |

| Innodb\_buffer\_pool\_reads              | 827      |

+---------------------------------------+----------+

命中率过低，说明innodb类型表的读写存在问题。

## binlog cache

若打开binlog日志功能，则需要考虑binlog cache问题。binlog不是一有数据就写到binlog中，而是先写入到binlog cache中，再写入到binlog中。

  Binlog\_cache\_disk\_use为binlog使用硬盘使用量， Binlog\_cache\_use  为binlog已使用的量。若 Binlog\_cache\_disk\_use大于0，则说明binlog\_cache不够用。

mysql> show status like 'binlog\_cache%';

+-----------------------+-------+

| Variable\_name         | Value |

+-----------------------+-------+

| Binlog\_cache\_disk\_use | 58    |

| Binlog\_cache\_use      | 39785 |

+-----------------------+-------+

## innodb\_log\_waits

innodb\_log\_waits状态变量直接反应innodb log buffer 空间不足造成等待的次数。innodb\_log\_waits直接反应系统的写入性能，当值 达到每秒1次时，就需要增加innodb\_log\_buffer\_size的值，适当的增加不会造成内存不足的问题。

## 复制延时量

复制延时量：复制延时量将直接影响了Slave数据库处于不一致状态的时间长短。如果我们是通过Slave来提供读服务，就不得不重视这个延时量。我们可以通过在Slave节点上执行“SHOWSLAVESTATUS”命令，取Seconds\_Behind\_Master项的值来了解Slave当前的延时量（单位：秒）。当然，该值的准确性依赖于复制是否处于正常状态。每个环境下的Slave所允许的延时长短与具体环境有关，所以复制延时多长时间是合理的，只能由读者朋友根据各自实际的应用环境来判断。

mysql> show slave status\G

  Seconds\_Behind\_Master: NULL