

MySQL 优化笔记

1. MySQL 的优化遵循如下步骤

硬件—>系统—>根据业务需要合理的 SQL 语句—>优化 MySQL (my.cnf) —>应用程序

2. 优化硬件

通常硬件是优化的最佳入口，因为 MySQL 内部使用大量 64 位的整数，64 位的 CPU 将提供更好的性能。优化的次序一般是内存、快速硬盘、CPU 能力；网络延迟与吞吐量同样重要；Raid10 要比 Raid5 快；

3. 优化操作系统和软件

使用更稳定高效的内核，64位要比32位快，使用合适的文件系统，ext3要比ext2快很多；

不要交换区。如果内存不足，增加更多的内存或配置你的系统使用较少内存。

不要使用NFS磁盘(会有NFS锁定的问题)。

增加系统和MySQL服务器的打开文件数量。(在safe mysqld脚本中加入ulimit -n #)。

增加系统的进程和线程数量。

如果你有相对较少的大表，告诉文件系统不要将文件打碎在不同的磁道上(Solaris)。

使用支持大文件的文件系统(Solaris)。

选择使用哪种文件系统。在Linux上的Reiserfs对于打开、读写都非常快。文件检查只需几秒钟。

通常新版本的效率不如旧版本，但是可以利用新版本的新功能来从另一方面得到性能上的提升；

软件安装时采用静态编译；

4. 优化应用

在编写应用时，应该决定什么是最重要的：

速度

操作系统间的可移植性

SQL服务器间的可移植性

使用持续的连接

缓存应用中的数据以减少SQL服务器的负载。

不要查询应用中不需要的列。

不要使用SELECT * FROM table_name...

测试应用的所有部分，但将大部分精力放在在可能最坏的合理的负载下的测试整体应用。通过以一种模块化的方式进行，你应该能用一个快速“哑模块”替代找到的瓶颈，然后很容易地标出下一个瓶颈。

如果在一个批处理中进行大量修改，使用LOCK TABLES。例如将多个UPDATES或DELETES集中在一起。

5. 优化MySQL

挑选编译器和编译选项。

为你的系统寻找最好的启动选项。

通读MySQL参考手册并阅读Paul DuBios的《MySQL》一书。（已有中文版-译注）

多用EXPLAIN SELECT、SHOW VARIABLES、SHOW STATUS和SHOW PROCESSLIST。

了解查询优化器的工作原理。

优化表的格式。

维护你的表(myisamchk、CHECK TABLE、OPTIMIZE TABLE)

不要使用表级或列级的GRANT，除非你确实需要。

6. 维护

如果可能，偶尔运行一下OPTIMIZE table，这对大量更新的变长行非常重要。

偶尔用myisamchk -a更新一下表中的键码分布统计。记住在做之前关掉MySQL。

如果有碎片文件，可能值得将所有文件复制到另一个磁盘上，清除原来的磁盘并拷回文件。

如果遇到问题，用myisamchk或CHECK table检查表。

用mysqladmin -i10 processlist extended-status监控MySQL的状态。

用MySQL GUI客户程序，你可以在不同的窗口内监控进程列表和状态。

使用mysqladmin debug获得有关锁定和性能的信息。

7. 重要的 MySQL 启动选项

`back_log` 如果需要大量新连接，修改它。

`thread_cache_size` 如果需要大量新连接，修改它。

`key_buffer_size` 索引页池，可以设成很大。

`bdb_cache_size` BDB 表使用的记录和键码高速缓存。

`table_cache` 如果有很多的表和并发连接，修改它。

`delay_key_write` 如果需要缓存所有键码写入，设置它。

`log_slow_queries` 找出需花大量时间的查询。

`max_heap_table_size` 用于 GROUP BY

`sort_buffer` 用于 ORDER BY 和 GROUP BY

`myisam_sort_buffer_size` 用于 REPAIR TABLE

`join_buffer_size` 在进行无键码的联结时使用。

8. 优化表

MySQL拥有一套丰富的类型。你应该对每一列尝试使用最有效的类型。

ANALYSE过程可以帮助你找到表的最优类型：`SELECT * FROM table_name PROCEDURE ANALYSE()`。

对于不保存NULL值的列使用NOT NULL，这对你想索引的列尤其重要。

将ISAM类型的表改为MyISAM。

如果可能，用固定的表格式创建表。

不要索引你不想用的东西。

利用MySQL能按一个索引的前缀进行查询的事实。如果你有索引INDEX(a, b)，你不需要在a上的索引。

不在长CHAR/VARCHAR列上创建索引，而只索引列的一个前缀以节省存储空间。`CREATE TABLE table_name (hostname CHAR(255) not null, index(hostname(10)))`

对每个表使用最有效的表格式。

在不同表中保存相同信息的列应该有同样的定义并具有相同的列名。

9. MySQL如何存储数据

数据库以目录存储。

表以文件存储。

列以变长或定长格式存储在文件中。对BDB表，数据以页面形式存储。

支持基于内存的表。

数据库和表可在不同的磁盘上用符号连接起来。

在Windows上，MySQL支持用.sym文件内部符号连接数据库。

10. MySQL 表类型

HEAP 表：固定行长的表，只存储在内存中并用 HASH 索引进行索引。

ISAM 表：MySQL 3.22 中的早期 B-tree 表格式。

MyIASM：IASM 表的新版本，有如下扩展：

二进制层次的可移植性。

NULL 列索引。

对变长行比 ISAM 表有更少的碎片。

支持大文件。

更好的索引压缩。

更好的键吗统计分布。

更好和更快的 auto_increment 处理。

来自 Sleepcat 的 Berkeley DB (BDB) 表：事务安全 (有 BEGIN WORK/COMMIT|ROLLBACK)。

11. MySQL 行类型 (专指 IASM/MyIASM 表)

如果所有列是定长格式 (没有 VARCHAR、BLOB 或 TEXT)，MySQL 将以定长表格式创建表，否则表以动态长度格式创建。

定长格式比动态长度格式快很多并更安全。

动态长度行格式一般占用较少的存储空间，但如果表频繁更新，会产生碎片。

在某些情况下，不值得将所有 VARCHAR、BLOB 和 TEXT 列转移到另一个表中，只是获得主表上的更快速度。

利用 myiasmchk (对 ISAM, pack_iasm)，可以创建只读压缩表，这使磁盘使用率最小，但使用慢速磁盘时，这非常不错。压缩表充分地利用将不再更新的日志表

12. MySQL 高速缓存（所有线程共享，一次性分配）

键码缓存: `key_buffer_size`, 默认 8M。

表缓存: `table_cache`, 默认 64。

线程缓存: `thread_cache_size`, 默认 0。

主机名缓存: 可在编译时修改, 默认 128。

内存映射表: 目前仅用于压缩表。

注意: MySQL 没有行高速缓存, 而让操作系统处理。

13. MySQL 缓存区变量（非共享，按需分配）

`sort_buffer`: ORDER BY/GROUP BY

`record_buffer`: 扫描表。

`join_buffer_size`: 无键联结

`myisam_sort_buffer_size`: REPAIR TABLE

`net_buffer_length`: 对于读 SQL 语句并缓存结果。

`tmp_table_size`: 临时结果的 HEAP 表大小。

14. MySQL 安装参数

```
/usr/sbin/useradd -s /sbin/nologin -d /dev/null mysql
./configure --prefix=/opt/mysql --enable-profiling --enable-thread-safe-client --without-debug
--with-charset=utf8 --with-extra-charsets=all --with-unix-socket-path=/tmp/mysql.sock
--with-plugins=partition,blackhole,heap,innobase,myisam,ndbcluster && make && make install
chown -R mysql:mysql /opt/mysql
./scripts/mysql_install_db --user=mysql
/opt/mysql/bin/mysqld_safe &
```

15. My.cnf 配置参数优化

下面的配置基本上能承受 1000qps 的压力, myisam+innodb

- `external-locking = FALSE`
- `back_log = 1024`
- `max_connections = 1200`
- `max_connect_errors = 1024`
- `open_files_limit = 4096`
- `max_allowed_packet = 24M`
- `read_rnd_buffer_size = 4M`
- `read_buffer_size = 4M`
- `join_buffer_size = 4M`
- `sort_buffer_size = 2M`
- `query_cache_limit = 2M`
- `query_cache_size = 400M`
- `query_cache_min_res_unit=2k`

- `thread_cache_size = 1200`
- `thread_concurrency = 4`
- `thread_stack = 128K`
- `tmp_table_size = 256M`
- `max_tmp_tables = 256`
- `bulk_insert_buffer_size = 4M`
- `binlog_cache_size = 2M`
- `max_binlog_size = 128M`
- `max_binlog_cache_size= 512M`
- `log-queries-not-using-indexes`
- `long_query_time = 1`
- `innodb_data_home_dir = /usr/local/mysql/data/`
- `innodb_data_file_path = ibdata1:10M:autoextend`
- `innodb_log_group_home_dir = /logs/mysql/ #data和log目录分开`
- `innodb_file_per_table= 1`
- `innodb_buffer_pool_size = 1300M`
- `innodb_log_file_size = 256M`
- `innodb_log_buffer_size = 16M`
- `innodb_lock_wait_timeout = 100`
- `innodb_flush_log_at_trx_commit = 2 #设成0, 大致会快4.4倍`
- `innodb_flush_method = 'O_DIRECT'`
- `set-variable="transaction-isolation=READ-COMMITTED"`
- `innodb_file_io_threads = 4`
- `innodb_thread_concurrency = 4`
- `innodb_log_files_in_group = 3`
- `innodb_max_dirty_pages_pct = 90`

Query_cache优化

```
mysql> show variables like '%query_cache%';
```

Variable_name	Value
have_query_cache	YES
query_cache_limit	2097152
query_cache_min_res_unit	4096
query_cache_size	67108864
query_cache_type	ON
query_cache_wlock_invalidate	OFF

have_query_cache 是否支持query_cache
query_cache_limit 存放单条query最大result set, 默认1MB
query_cache_min_res_unit 每个result set存放的最小内存大小
query_cache_size 系统中用于query_cache的大小

```
mysql> show status like 'Qcache%';
```

Variable_name	Value
Qcache_free_blocks	8417
Qcache_free_memory	52737904
Qcache_hits	726010439
Qcache_inserts	998691973
Qcache_lowmem_prunes	3421917
Qcache_not_cached	27775423
Qcache_queries_in_cache	13256
Qcache_total_blocks	34996

Qcache_free_blocks 目前剩余多少blocks, 如果此值偏大, 说明query cache内存碎片较多需整理;

Qcache_free_memory query cache中目前剩余内存大小;

Qcache_hits 多少次命中;

Qcache_inserts 多少次未命中然后被插入query cache

***query cache 命中率 = Qcache_hits / (Qcache_hits + Qcache_inserts)**

Qcache_lowmem_prunes 多少条因为内存不足被清除出query cache

*** Qcache_lowmem_prunes和Qcache_free_memory相互结合可以清楚的了解系统中query cache的内存大小是否足够**

MyISAM引擎优化

*Query通过索引检索表数据过程

1. query请求，直接读取key cache中的cache block，有则返回；
2. 没有则到.MYI文件中以file block方式读取数据；
3. 然后以完全相同的格式存入key cache做为cache block；
4. 然后再将key cache中的数据返回；

```
mysql> show status like 'Key%';
```

Variable_name	Value
Key_blocks_not_flushed	0
Key_blocks_unused	5779253
Key_blocks_used	3184154
Key_read_requests	19813453982
Key_reads	136159256
Key_write_requests	2249688875
Key_writes	1567010771

Key_blocks_not_flushed 已经更改但未刷新到磁盘的dirty cache block；

Key_blocks_unused 目前未被使用的cache block数目；

Key_blocks_used 已经使用了的cache block数目；

Key_read_requests cache block被请求读取的总次数；

Key_reads 在cache block中找不到需要读取的key，到“myi”文件读取的次数；

Key_write_requests 被请求修改的总次数；

Key_writes 在cache block中找不到需要修改的key信息后，到“myi”文件写入再修改的次数；

Key_buffer使用率 = $\text{Key_blocks_used} / (\text{Key_blocks_used} + \text{Key_blocks_unused}) * 100\%$

Key_buffer读命中率 = $1 - \text{Key_reads} / \text{Key_read_requests} * 100\%$

Key_buffer使用率应该在99%以上，如果过低说明key_buffer_size设置过高，mysql根本用不完；

Key_buffer读命中率应该尽可能高，如果过低说明key_buffer_size设置过低，mysql无法在cache block中添加更多数据；

如果系统主要以写为主，尤其有大量insert语句时，为了提高insert效率，可以讲concurrent_insert设置为2，也就是告诉MyISAM，不管在表中是否有删除行留下的空余空间，都在尾部进行并发插入，使insert和select互补干扰；

一般来说，在每次做了较大的数据删除操作之后都需要做一次optimize优化整理，每个季度有一次optimize操作；

InnoDB引擎优化

```
mysql> show status like 'InnoDB_buffer_pool%';
```

Variable_name	Value
InnoDB_buffer_pool_pages_data	759816
InnoDB_buffer_pool_pages_dirty	0
InnoDB_buffer_pool_pages_flushed	149429754
InnoDB_buffer_pool_pages_free	475
InnoDB_buffer_pool_pages_misc	26141
InnoDB_buffer_pool_pages_total	786432
InnoDB_buffer_pool_read_ahead_rnd	32
InnoDB_buffer_pool_read_ahead_seq	6981
InnoDB_buffer_pool_read_requests	51081643071
InnoDB_buffer_pool_reads	78664
InnoDB_buffer_pool_wait_free	0
InnoDB_buffer_pool_write_requests	12410261444

InnoDB Buffer pool的read命中率 = $\frac{\text{InnoDB_buffer_pool_reads} - \text{InnoDB_buffer_pool_wait_free}}{\text{InnoDB_buffer_pool_reads}} * 100\%$