[**MySQL SQL优化**](http://www.cnblogs.com/ggjucheng/archive/2012/11/11/2765465.html)

**前言**

有人反馈之前几篇文章过于理论缺少实际操作细节，这篇文章就多一些可操作性的内容吧。

注：这篇文章是以 MySQL 为背景，很多内容同时适用于其他关系型数据库，需要有一些索引知识为基础。

**优化目标**

　　1.减少 IO 次数

　　IO永远是数据库最容易瓶颈的地方，这是由数据库的职责所决定的，大部分数据库操作中超过90%的时间都是 IO 操作所占用的，减少 IO 次数是 SQL 优化中需要第一优先考虑，当然，也是收效最明显的优化手段。

　　2.降低 CPU 计算

　　除了 IO 瓶颈之外，SQL优化中需要考虑的就是 CPU 运算量的优化了。order by, group by,distinct … 都是消耗 CPU 的大户(这些操作基本上都是 CPU 处理内存中的数据比较运算)。当我们的 IO 优化做到一定阶段之后，降低 CPU 计算也就成为了我们 SQL 优化的重要目标

**优化方法**

　　改变 SQL 执行计划

　　明确了优化目标之后，我们需要确定达到我们目标的方法。对于 SQL 语句来说，达到上述2个目标的方法其实只有一个，那就是改变 SQL 的执行计划，让他尽量“少走弯路”，尽量通过各种“捷径”来找到我们需要的数据，以达到 “减少 IO 次数” 和 “降低 CPU 计算” 的目标

**常见误区**

1.count(1)和count(primary\_key) 优于 count(\*)

　　很多人为了统计记录条数，就使用 count(1) 和 count(primary\_key) 而不是 count(\*) ，他们认为这样性能更好，其实这是一个误区。对于有些场景，这样做可能性能会更差，应为数据库对 count(\*) 计数操作做了一些特别的优化。

2.count(column) 和 count(\*) 是一样的

　　这个误区甚至在很多的资深工程师或者是 DBA 中都普遍存在，很多人都会认为这是理所当然的。实际上，count(column) 和 count(\*) 是一个完全不一样的操作，所代表的意义也完全不一样。

　　count(column) 是表示结果集中有多少个column字段不为空的记录

　　count(\*) 是表示整个结果集有多少条记录

3.select a,b from … 比 select a,b,c from … 可以让数据库访问更少的数据量

　　这个误区主要存在于大量的开发人员中，主要原因是对数据库的存储原理不是太了解。

　　实际上，大多数关系型数据库都是按照行(row)的方式存储，而数据存取操作都是以一个固定大小的IO单元(被称作 block 或者 page)为单位，一般为4KB，8KB… 大多数时候，每个IO单元中存储了多行，每行都是存储了该行的所有字段(lob等特殊类型字段除外)。

　　所以，我们是取一个字段还是多个字段，实际上数据库在表中需要访问的数据量其实是一样的。

　　当然，也有例外情况，那就是我们的这个查询在索引中就可以完成，也就是说当只取 a,b两个字段的时候，不需要回表，而c这个字段不在使用的索引中，需要回表取得其数据。在这样的情况下，二者的IO量会有较大差异。

4.order by 一定需要排序操作

　　我们知道索引数据实际上是有序的，如果我们的需要的数据和某个索引的顺序一致，而且我们的查询又通过这个索引来执行，那么数据库一般会省略排序操作，而直接将数据返回，因为数据库知道数据已经满足我们的排序需求了。

　　实际上，利用索引来优化有排序需求的 SQL，是一个非常重要的优化手段

　　延伸阅读：MySQL ORDER BY 的实现分析，MySQL 中 GROUP BY 基本实现原理以及 MySQL DISTINCT 的基本实现原理这3篇文章中有更为深入的分析，尤其是第一篇

5.执行计划中有 filesort 就会进行磁盘文件排序

　　有这个误区其实并不能怪我们，而是因为 MySQL 开发者在用词方面的问题。filesort 是我们在使用 explain 命令查看一条 SQL 的执行计划的时候可能会看到在 “Extra” 一列显示的信息。

　　实际上，只要一条 SQL 语句需要进行排序操作，都会显示“Using filesort”，这并不表示就会有文件排序操作。

**基本原则**

1.尽量少 join

　　MySQL 的优势在于简单，但这在某些方面其实也是其劣势。MySQL 优化器效率高，但是由于其统计信息的量有限，优化器工作过程出现偏差的可能性也就更多。对于复杂的多表 Join，一方面由于其优化器受限，再者在 Join 这方面所下的功夫还不够，所以性能表现离 Oracle 等关系型数据库前辈还是有一定距离。但如果是简单的单表查询，这一差距就会极小甚至在有些场景下要优于这些数据库前辈。

2.尽量少排序

　　排序操作会消耗较多的 CPU 资源，所以减少排序可以在缓存命中率高等 IO 能力足够的场景下会较大影响 SQL 的响应时间。

　　对于MySQL来说，减少排序有多种办法，比如：

　　上面误区中提到的通过利用索引来排序的方式进行优化

　　减少参与排序的记录条数

　　非必要不对数据进行排序

　　…

3.尽量避免 select \*

　　很多人看到这一点后觉得比较难理解，上面不是在误区中刚刚说 select 子句中字段的多少并不会影响到读取的数据吗?

　　是的，大多数时候并不会影响到 IO 量，但是当我们还存在 order by 操作的时候，select 子句中的字段多少会在很大程度上影响到我们的排序效率，这一点可以通过我之前一篇介绍 MySQL ORDER BY 的实现分析的文章中有较为详细的介绍。

　　此外，上面误区中不是也说了，只是大多数时候是不会影响到 IO 量，当我们的查询结果仅仅只需要在索引中就能找到的时候，还是会极大减少 IO 量的。

4.尽量用 join 代替子查询

　　虽然 Join 性能并不佳，但是和 MySQL 的子查询比起来还是有非常大的性能优势。MySQL 的子查询执行计划一直存在较大的问题，虽然这个问题已经存在多年，但是到目前已经发布的所有稳定版本中都普遍存在，一直没有太大改善。虽然官方也在很早就承认这一问题，并且承诺尽快解决，但是至少到目前为止我们还没有看到哪一个版本较好的解决了这一问题。

5.尽量少 or

　　当 where 子句中存在多个条件以“或”并存的时候，MySQL 的优化器并没有很好的解决其执行计划优化问题，再加上 MySQL 特有的 SQL 与 Storage 分层架构方式，造成了其性能比较低下，很多时候使用 union all 或者是union(必要的时候)的方式来代替“or”会得到更好的效果。

6.尽量用 union all 代替 union

　　union 和 union all 的差异主要是前者需要将两个(或者多个)结果集合并后再进行唯一性过滤操作，这就会涉及到排序，增加大量的 CPU 运算，加大资源消耗及延迟。所以当我们可以确认不可能出现重复结果集或者不在乎重复结果集的时候，尽量使用 union all 而不是 union。

7.尽量早过滤

　　这一优化策略其实最常见于索引的优化设计中(将过滤性更好的字段放得更靠前)。

　　在 SQL 编写中同样可以使用这一原则来优化一些 Join 的 SQL。比如我们在多个表进行分页数据查询的时候，我们最好是能够在一个表上先过滤好数据分好页，然后再用分好页的结果集与另外的表 Join，这样可以尽可能多的减少不必要的 IO 操作，大大节省 IO 操作所消耗的时间。

8.避免类型转换

　　这里所说的“类型转换”是指 where 子句中出现 column 字段的类型和传入的参数类型不一致的时候发生的类型转换：

　　人为在column\_name 上通过转换函数进行转换

　　直接导致 MySQL(实际上其他数据库也会有同样的问题)无法使用索引，如果非要转换，应该在传入的参数上进行转换

　　由数据库自己进行转换

　　如果我们传入的数据类型和字段类型不一致，同时我们又没有做任何类型转换处理，MySQL 可能会自己对我们的数据进行类型转换操作，也可能不进行处理而交由存储引擎去处理，这样一来，就会出现索引无法使用的情况而造成执行计划问题。

9.优先优化高并发的 SQL，而不是执行频率低某些“大”SQL

　　对于破坏性来说，高并发的 SQL 总是会比低频率的来得大，因为高并发的 SQL 一旦出现问题，甚至不会给我们任何喘息的机会就会将系统压跨。而对于一些虽然需要消耗大量 IO 而且响应很慢的 SQL，由于频率低，即使遇到，最多就是让整个系统响应慢一点，但至少可能撑一会儿，让我们有缓冲的机会。

10.从全局出发优化，而不是片面调整

　　SQL 优化不能是单独针对某一个进行，而应充分考虑系统中所有的 SQL，尤其是在通过调整索引优化 SQL 的执行计划的时候，千万不能顾此失彼，因小失大。

11.尽可能对每一条运行在数据库中的SQL进行 explain

　　优化 SQL，需要做到心中有数，知道 SQL 的执行计划才能判断是否有优化余地，才能判断是否存在执行计划问题。在对数据库中运行的 SQL 进行了一段时间的优化之后，很明显的问题 SQL 可能已经很少了，大多都需要去发掘，这时候就需要进行大量的 explain 操作收集执行计划，并判断是否需要进行优化。

方案2

MYSQL 应该是最流行了 WEB 后端数据库。WEB 开发语言最近发展很快，PHP， Ruby, Python, Java 各有特点，虽然 NOSQL 最近越來越多的被提到，但是相信大部分架构师还是会选择 MYSQL 来做数据存储。

MYSQL 如此方便和稳定，以至于我们在开发 WEB 程序的时候很少想到它。即使想到优化也是程序级别的，比如，不要写过于消耗资源的 SQL 语句。但是除此之外，在整个系统上仍然有很多可以优化的地方。

**1. 选择合适的存储引擎: InnoDB**

除非你的数据表使用来做只读或者全文检索 (相信现在提到全文检索，没人会用 MYSQL 了)，你应该默认选择 InnoDB 。

你自己在测试的时候可能会发现 MyISAM 比 InnoDB 速度快，这是因为： MyISAM 只缓存索引，而 InnoDB 缓存数据和索引，MyISAM 不支持事务。但是 如果你使用 innodb\_flush\_log\_at\_trx\_commit = 2 可以获得接近的读取性能 (相差百倍) 。

**1.1 如何将现有的 MyISAM 数据库转换为 InnoDB:**

复制代码代码如下:

mysql -u [USER\_NAME] -p -e "SHOW TABLES IN [DATABASE\_NAME];" | tail -n +2 | xargs -I '{}' echo "ALTER TABLE {} ENGINE=InnoDB;" > alter\_table.sql  
perl -p -i -e 's/(search\_[a-z\_]+ ENGINE=)InnoDB//1MyISAM/g' alter\_table.sql  
mysql -u [USER\_NAME] -p [DATABASE\_NAME] < alter\_table.sql

**1.2 为每个表分别创建 InnoDB FILE：**

复制代码代码如下:

innodb\_file\_per\_table=1

这样可以保证 ibdata1 文件不会过大，失去控制。尤其是在执行 mysqlcheck -o –all-databases 的时候。

**2. 保证从内存中读取数据，讲数据保存在内存中**

**2.1 足够大的 innodb\_buffer\_pool\_size**

推荐将数据完全保存在 innodb\_buffer\_pool\_size ，即按存储量规划 innodb\_buffer\_pool\_size 的容量。这样你可以完全从内存中读取数据，最大限度减少磁盘操作。

**2.1.1 如何确定 innodb\_buffer\_pool\_size 足够大，数据是从内存读取而不是硬盘？**  
**方法 1**

[?](http://www.jb51.net/article/70111.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | mysql> SHOW GLOBAL STATUS LIKE 'innodb\_buffer\_pool\_pages\_%';  +----------------------------------+--------+  | Variable\_name          | Value |  +----------------------------------+--------+  | Innodb\_buffer\_pool\_pages\_data  | 129037 |  | Innodb\_buffer\_pool\_pages\_dirty  | 362  |  | Innodb\_buffer\_pool\_pages\_flushed | 9998  |  | Innodb\_buffer\_pool\_pages\_free  | 0   | !!!!!!!!  | Innodb\_buffer\_pool\_pages\_misc  | 2035  |  | Innodb\_buffer\_pool\_pages\_total  | 131072 |  +----------------------------------+--------+  6 rows in set (0.00 sec) |

发现 Innodb\_buffer\_pool\_pages\_free 为 0，则说明 buffer pool 已经被用光，需要增大 innodb\_buffer\_pool\_size

InnoDB 的其他几个参数：

复制代码代码如下:

innodb\_additional\_mem\_pool\_size = 1/200 of buffer\_pool  
innodb\_max\_dirty\_pages\_pct 80%

**方法 2**

或者用iostat -d -x -k 1 命令，查看硬盘的操作。

**2.1.2 服务器上是否有足够内存用来规划**执行 echo 1 > /proc/sys/vm/drop\_caches 清除操作系统的文件缓存，可以看到真正的内存使用量。

**2.2 数据预热**

默认情况，只有某条数据被读取一次，才会缓存在 innodb\_buffer\_pool。所以，数据库刚刚启动，需要进行数据预热，将磁盘上的所有数据缓存到内存中。数据预热可以提高读取速度。

对于 InnoDB 数据库，可以用以下方法，进行数据预热:

**1. 将以下脚本保存为 MakeSelectQueriesToLoad.sql**

[?](http://www.jb51.net/article/70111.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | SELECT DISTINCT    CONCAT('SELECT ',ndxcollist,' FROM ',db,'.',tb,    ' ORDER BY ',ndxcollist,';') SelectQueryToLoadCache    FROM    (      SELECT        engine,table\_schema db,table\_name tb,        index\_name,GROUP\_CONCAT(column\_name ORDER BY seq\_in\_index) ndxcollist      FROM      (        SELECT          B.engine,A.table\_schema,A.table\_name,          A.index\_name,A.column\_name,A.seq\_in\_index        FROM          information\_schema.statistics A INNER JOIN          (            SELECT engine,table\_schema,table\_name            FROM information\_schema.tables WHERE            engine='InnoDB'          ) B USING (table\_schema,table\_name)        WHERE B.table\_schema NOT IN ('information\_schema','mysql')        ORDER BY table\_schema,table\_name,index\_name,seq\_in\_index      ) A      GROUP BY table\_schema,table\_name,index\_name    ) AA  ORDER BY db,tb  ; |

**2. 执行**

复制代码代码如下:

mysql -uroot -AN < /root/MakeSelectQueriesToLoad.sql > /root/SelectQueriesToLoad.sql

**3. 每次重启数据库，或者整库备份前需要预热的时候执行：**

mysql -uroot < /root/SelectQueriesToLoad.sql > /dev/null 2>&1

**2.3 不要让数据存到 SWAP 中**

如果是专用 MYSQL 服务器，可以禁用 SWAP，如果是共享服务器，确定 innodb\_buffer\_pool\_size 足够大。或者使用固定的内存空间做缓存，使用 memlock 指令。

**3. 定期优化重建数据库**

mysqlcheck -o –all-databases 会让 ibdata1 不断增大，真正的优化只有重建数据表结构：

[?](http://www.jb51.net/article/70111.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE mydb.mytablenew LIKE mydb.mytable;  INSERT INTO mydb.mytablenew SELECT \* FROM mydb.mytable;  ALTER TABLE mydb.mytable RENAME mydb.mytablezap;  ALTER TABLE mydb.mytablenew RENAME mydb.mytable;  DROP TABLE mydb.mytablezap; |

**4. 减少磁盘写入操作**

**4.1 使用足够大的写入缓存 innodb\_log\_file\_size**

但是需要注意如果用 1G 的 innodb\_log\_file\_size ，假如服务器当机，需要 10 分钟来恢复。

推荐 innodb\_log\_file\_size 设置为 0.25 \* innodb\_buffer\_pool\_size

**4.2 innodb\_flush\_log\_at\_trx\_commit**

这个选项和写磁盘操作密切相关：

innodb\_flush\_log\_at\_trx\_commit = 1 则每次修改写入磁盘  
innodb\_flush\_log\_at\_trx\_commit = 0/2 每秒写入磁盘

如果你的应用不涉及很高的安全性 (金融系统)，或者基础架构足够安全，或者 事务都很小，都可以用 0 或者 2 来降低磁盘操作。

**4.3 避免双写入缓冲**

复制代码代码如下:

innodb\_flush\_method=O\_DIRECT

**5. 提高磁盘读写速度**

RAID0 尤其是在使用 EC2 这种虚拟磁盘 (EBS) 的时候，使用软 RAID0 非常重要。

**6. 充分使用索引**

**6.1 查看现有表结构和索引**

复制代码代码如下:

SHOW CREATE TABLE db1.tb1/G

**6.2 添加必要的索引**

索引是提高查询速度的唯一方法，比如搜索引擎用的倒排索引是一样的原理。

索引的添加需要根据查询来确定，比如通过慢查询日志或者查询日志,或者通过 EXPLAIN 命令分析查询。

复制代码代码如下:

ADD UNIQUE INDEX  
ADD INDEX

**6.2.1 比如，优化用户验证表：**  
添加索引

复制代码代码如下:

ALTER TABLE users ADD UNIQUE INDEX username\_ndx (username);  
ALTER TABLE users ADD UNIQUE INDEX username\_password\_ndx (username,password);

每次重启服务器进行数据预热

复制代码代码如下:

echo “select username,password from users;” > /var/lib/mysql/upcache.sql

添加启动脚本到 my.cnf

复制代码代码如下:

[mysqld]  
init-file=/var/lib/mysql/upcache.sql

**6.2.2 使用自动加索引的框架或者自动拆分表结构的框架**比如，Rails 这样的框架，会自动添加索引，Drupal 这样的框架会自动拆分表结构。会在你开发的初期指明正确的方向。所以，经验不太丰富的人一开始就追求从 0 开始构建，实际是不好的做法。

**7. 分析查询日志和慢查询日志**

记录所有查询，这在用 ORM 系统或者生成查询语句的系统很有用。

复制代码代码如下:

log=/var/log/mysql.log

注意不要在生产环境用，否则会占满你的磁盘空间。

记录执行时间超过 1 秒的查询：

复制代码代码如下:

long\_query\_time=1  
log-slow-queries=/var/log/mysql/log-slow-queries.log

**8. 激进的方法，使用内存磁盘**

现在基础设施的可靠性已经非常高了，比如 EC2 几乎不用担心服务器硬件当机。而且内存实在是便宜，很容易买到几十G内存的服务器，可以用内存磁盘，定期备份到磁盘。

将 MYSQL 目录迁移到 4G 的内存磁盘

[?](http://www.jb51.net/article/70111.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | mkdir -p /mnt/ramdisk  sudo mount -t tmpfs -o size=4000M tmpfs /mnt/ramdisk/  mv /var/lib/mysql /mnt/ramdisk/mysql  ln -s /tmp/ramdisk/mysql /var/lib/mysql  chown mysql:mysql mysql |

**9. 用 NOSQL 的方式使用 MYSQL**

B-TREE 仍然是最高效的索引之一，所有 MYSQL 仍然不会过时。

用 HandlerSocket 跳过 MYSQL 的 SQL 解析层，MYSQL 就真正变成了 NOSQL。

**10. 其他**

单条查询最后增加 LIMIT 1，停止全表扫描。  
将非”索引”数据分离，比如将大篇文章分离存储，不影响其他自动查询。  
不用 MYSQL 内置的函数，因为内置函数不会建立查询缓存。  
PHP 的建立连接速度非常快，所有可以不用连接池，否则可能会造成超过连接数。当然不用连接池 PHP 程序也可能将  
连接数占满比如用了 @ignore\_user\_abort(TRUE);  
使用 IP 而不是域名做数据库路径，避免 DNS 解析问题

以上就是10个MySQL性能调优的方法，希望对大家的学习有所帮助。