MySQL 性能优化方案总结

可以从以下几个方面对 MySQL 进行优化，

效果: SQL 和索引 > 数据库表结构 > 系统配置 > 硬件

但成本从低到高。

1.SQL 和索引优化

1.1SQL

1.1.1 优化 SQL 语句的一般步骤：

①通过 show status 命令了解各种 SQL 的执行效率，

```
show [session | global] status;
```

可以根据需要加上参数来显示 session 级（当前连接，默认）和 global 级（自数据库上次启动至今）的统计结果。

eg:

```
show status like 'Com_%';
```

显示当前连接所有统计参数的值。

Com_xxx 表示每个 xxx 语句执行的次数，通常需要注意的是下面几个参数：

Com_select/Com_insert/Com_update/Com_delte。

②定位执行效率较低的 SQL 语句

·通过 show processlist 命令实时查看当前 SQL 的执行情况；

·通过慢查询日志（结束以后记录）定位出现的问题。

③通过 explain 或 desc 分析低效 SQL 的执行计划

select_type(simple/primary/union/subquery)/table/type/possible_keys/key/key_len/rows/extra

④通过 show profile 分析 SQL

show profile 能帮我们了解时间都耗费到哪里去了。

MySQL 从 5.0.37 版本开始增加了 show profile 和 show profiles 语句的支持，

通过 secect @have_profiling 命令能够看到当前 MySQL 是否支持 profile，

通过 show profiles 我们能够更清楚了解 SQL 执行的过程，

通过 show profile for query 我们能看到执行过程中线程的每个状态和消耗的时间。

⑤通过 trace 分析优化器如何选择执行计划

MySQL5.6 提供了对 SQL 的跟踪 trace,能帮我们了解为什么优化器选择执行 A 计划而不是 B 计划，进一步理解优化器的行为。

⑥确定问题并采取相应的优化措施

1.1.2 两个简单实用的优化方法

①定期分析和检查表

```
analyze table tbl_name;
check table tbl_name;
```

②定期优化表

```
optimize table tbl_name;
```

1.1.3 常用 SQL 的优化

①优化 insert 语句

·如果从同一客户端插入很多行，应该尽量使用多个值表一次性插入；

·如果从不同客户端插入很多行，可以使用 insert delayed 语句先把数据放在内存的队列中，并不真正写入磁盘，比每条语句分别插入快得多；

·当从一个文本文件装载一个表时，使用 load data infile，这通常比使用很多 insert 语句快 20 倍；

·如果在 MyISAM 表中进行批量插入，可以通过增加 bulk_insert_buffer_size 变量值的方法来提高速度。

②优化 order by 语句

MySQL 中有两种排序方式，第一种通过有序索引顺序扫描直接返回有效数据，不需要额外的排序，操作效率较高；第二种对返回的数据进行排序，也就是常说的 Filesort 排序，所有不是通过索引直接返回排序结果的排序都是 filesort 排序。

优化目标：尽量通过索引直接返回有序数据，减少额外的排序。

通过创建合适的索引能减少 filesort 出现，但是某些情况下，条件限制不能让 filesort 消失，那就需要想办法加快 filesort 的操作。filesort 有两种排序算法，一种是一次扫描算法（较快），二种是两次扫描算法。适当加大系统变量 max_length_for_sort_data 的值，能够让 MySQL 选择更优化的 filesort 排序算法；适当加大 sort_buffer_size 排序区，尽量让排序在内存中完成，而不是通过创建临时表放在文件中进行。尽量只使用必要的字段，select 具体的字段名称，而不是 select * 选择所有字段，这样可以减少排序区的使用，提高 SQL 性能。

③优化 group by 语句

MySQL 默认对所有 group by col1,col2…的字段进行排序，可以指定 order by null 禁止排序。

④优化嵌套查询

MySQL5.5 及以下版本，子查询的效率不如连接查询（join），因为 MySQL 不需要在内存中创建临时表来完成这个在逻辑上需要两个步骤的查询工作。

⑤优化 or 查询

对于含有 or 的查询子句，如果要利用索引，则 or 之间的每个条件列都必须使用索引；如果没有索引，可以考虑增加索引。

MySQL 在处理含有 or 的查询时，实际上对 or 的各个字段分别查询后的结果进行了 union 操作。

⑥优化分页查询

·第一种 在索引上完成排序分页操作，然后根据主键关联回原表查询所需要的其他列的内容；

·第二种 在排序字段不会出现重复值的情况下，新增一个参数记录上次查询的最后一条记录，将 limit m,n 转化成 limit n.

⑦使用 SQL 提示

就是在 SQL 语句中加入一些认为提示 ,让 MySQL 按照特定方案执行，以达到优化操作的目的。

·use index 指定 MySQL 参考的索引而忽略别的索引

·ignore index 让 MySQL 忽略某个或某些索引

·force index 强制 MySQL 使用某个特定的索引

⑧其他

·使用 REGEXP，比如代替 like.

·使用 rand()提取随机行

·表的字段尽量不使用自增长变量，在高并发的情况下可能会对 MySQL 的效率有较大影响。

1.2 索引优化

MySQL 的索引在存储引擎层实现，而不是在服务器层。

可以通过 show status like ‘Handler_read%’ 命令来查看索引使用情况。

1.2.1MySQL 中索引的存储类型目前有四种(B-Tree、Hash、空间索引 R-Tree、全文索引 Full-text），具体和表的存储引擎相关；MyISAM 和 InnoDB 存储引擎都支持 B-Tree 和全文索引（Full-text，InnoDB 5.6 +）；MyISAM 还支持空间索引（R-Tree）；Memory/Heap 存储引擎可以支持 HASH 和 B-Tree 索引，不过只有 Memory/Heap 支持 Hash 索引。

1.2.2MySQL 如何使用索引

（1）MySQL 使用索引的典型情景

①匹配全值（match the full value）

②匹配值的范围（match a range of values）

③匹配最左前缀（match a leftmost prefix）最左匹配原则是 MySQL 中 B-Tree 索引使用的首要原则。

④只查询索引（index only query）当然 where 子句中要满足最左匹配原则

⑤匹配列前缀（match a column prefix）使用复合索引的第一列的开头一部分

⑥复合索引中，一部分匹配精确内容 and 其他部分匹配一个范围（match one part exactly and match a range on another part）

⑦列名是索引，那么 column_name is null 就会使用索引,比如 where column_name is null

（2）MySQL 不使用索引的典型情景

①like “%query”不使用 B-Tree 索引，但 like “query%”会使用

B-Tree 索引。

②数据类型出现隐式转换的时候也不会使用索引。尤其当列类型是字符串时，一定记得在 where 条件中把字符串常量值用引号引起来，比如 where last_name = ‘1’；

③使用复合索引时，查询条件不包含索引的最左边部分

④用 or 分割的条件，如果其中一个列中没有索引，则涉及的另一个索引也不会被用到。

⑤如果 MySQL 估计使用索引比全表扫描更慢，则不使用索引。

2.优化数据库对象

2.1 选择合适的存储引擎

2.2 字段选择合适的数据类型

procedure analyse() 可以对当前应用的表进行分析，对数据表中列的数据类型提出优化建议。

2.3 三范式和反三范式

2.4 对表进行水平或者垂直拆分

3.针对存储引擎的优化

优化特定参数

4.优化 MySQL server

4.1MySQL 内存管理和优化

4.2InnoDB log 机制及优化

4.3 调整跟并发相关的 MySQL 参数

（1）max_connections

（2）back_log

（3）table_open_cache

（4）thread_cache_size

（5）innodb_lock_wait_timeout

5.磁盘 I/O 优化

5.1 使用磁盘阵列（RAID）

5.2 使用 Linux 虚拟文件卷模拟 RAID

5.3 符号连接（Symbolic Links）分布 I/O

利用操作系统的符号连接（Symbolic Links）将不同的数据库、表或索引指向不同的物理磁盘，从而达到分布磁盘 I/O 的目的。

5.4 禁止操作系统更新文件的 atime 属性

5.5 用裸设备（Raw Device）存放 InnoDB 的共享表空间

5.6 调整 I/O 调度算法

5.7RAID 卡电池的充放电引起的性能波动

5.8NUMA 架构优化

非一致存储访问结构（Non-Uniform Memory Access, NUMA）

6.应用优化

6.1 使用连接池

6.2 减少对 MySQL 的访问

①理清应用逻辑，能一次取出的数据不用两次；

②使用查询缓存

MySQL 的查询缓存（MySQL query cache）是 4.1 版本之后新增的功能，作用是存储 select 的查询文本和相应结果。如果随后收到一个相同的查询，服务器会从查询缓存中重新得到查询结果，而不再需要解析和执行查询。

查询缓存适用于更新不频繁的表，当表更改（包括表结构和数据）后，查询缓存会被清空。

③在应用端增加 cache 层

④负载均衡

负载均衡（Load Balance）是实际应用中使用非常普遍的一种优化方法，它的机制就是利用某种均衡算法，将固定的负载量分布到不同的服务器上，以此来减轻单台服务器的负载，达到优化的目的。负载均衡可以用

在系统中的各个层面中，从前台的 Web 服务器到中间层的应用服务器，最后到数据层的数据库服务器，都可以使用。

·利用 MySQL 复制分流查询和更新操作

利用 MySQL 的主从复制可以有效地分流更新操作和查询操作，具体的实现是一个主服务器承担更新操作，而多台从服务器承担查询操作，主从之间通过复制实现数据的同步。通过复制来分流查询和更新是减少主数据库负载的一个常用方法，但是这种办法也存在一些问题，最主要的问题是当主数据库上更新频繁或者网络出现问题的时候，主从之间的数据可能存在比较大的延迟更新，从而造成查询结果和主数据库上有所差异。因此在设计应用的时候需要有所考虑。