Neusoft

东软机密

MySQL 数据库性能优化心得

王消冰，
解决方案技术中心

Neusoft

目录

一、概述

MySQL 数据库使目前使用人数最多的开源关系型数据库系统，其 InnoDB 引擎具有支持事务完整性以及更大的事务并发能力等特性，适用于较大规模的 OLTP 应用。因此，INNODB 引擎在使用 MYSQL 作为数据库的应用中被广泛使用。而由于使用不当或数据量不断增大，系统复杂性提高等原因造成的性能下降的情况时有发生。因此，有必要总结 InnoDB 引擎下的 MySQL 数据库优化的理论与方法，以便为保障业务系统的高效运行提供帮助，本文重点论述 Mysql 数据库的性能优化技术。

二、需求分析

在 Web 应用程序体系架构中，数据持久层（通常是一个关系数据库）是关键的核心部分，它对系统的性能有非常重要的影响。MySQL 是目前使用最多的开源数据库，但是 MySQL 数据库的默认设置性能非常的差，无法满足实际业务系统对数据库的性能需求。因此在产品中使用 MySQL 数据库必须进行必要的优化。

三、内容和范围

任何数据库优化都是一个复杂的任务，MySQL 数据库也不例外，在该文档中重点讨论如下范围的性能优化内容：

MySQL 数据库相关的存储引擎优化，数据库设计和查询优化，服务器端优化，常见参数的优化，语句优化，结构优化。

四、MySQL 数据库优化技术

4.1、存储引擎的优化

MySQL 支持不同的存储引擎，主要使用的有 MyISAM 和 InnoDB。

4.1.1 MyISAM

MyISAM 管理非事务表。它提供高速存储和检索，以及全文搜索能力。MyISAM 在所有 MySQL 配置里被支持，它是默认的存储引擎，除非配置 MySQL 默认使用另外一个引擎。

4.1.1.1 MyISAM 特性

- **MyISAM 属性**

1) 不支持事务，宕机会破坏表
2) 使用较小的内存和磁盘空间
3) 基于表的锁，并发更新数据会出现严重性能问题
4) MySQL 只缓存 Index，数据由 OS 缓存

- **MyISAM 适用范围**

1) 日志系统
2) 只读或者绝大部分是读操作的应用
3) 全表扫描
4) 批量导入数据
5) 没有事务的低并发读/写

4.1.1.2 MyISAM 优化要点

- **基本优化**

1) 声明列为 NOT NULL，可以减少磁盘存储。
2) 使用 optimize table 做碎片整理，回收空闲空间。注意仅仅在非常大的数据变化后运行。
3) Deleting/updating/adding 大量数据的时候禁止使用 index。使用 ALTER TABLE t DISABLE KEYS。
4) 设置 myisam_max_[extra]_sort_file_size 足够大，可以显著提高 repair table 的速度。

- **MyISAM Table Locks 优化**

1) 避免并发 insert，update。
2) 可以使用 insert delayed，但是有可能丢失数据。
3) 优化查询语句。
4) 水平分区。

5) 垂直分区。

6) 如果都不起作用，使用 InnoDB。

- **MyISAM Key Cache 优化**

1) 设置 key_buffer_size variable。MyISAN 最主要的 cache 设置，用于缓存 MyISAM 表格的 index 数据，该参数只对 MyISAM 有影响。通常在只使用 MyISAM 的 Server 中设置 25-33%的内存大小。

2) 可以使用几个不同的 Key Caches（对一些 hot data）。

a) SET GLOBAL test.key_buffer_size=512*1024;

b) CACHE INDEX t1.i1, t2.i1, t3 IN test;

3) Preload index 到 Cache 中可以提高查询速度。因为 preloading index 是顺序的，所以非常快。

a) LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;

4.1.2 InnoDB

InnoDB 给 MySQL 提供了具有提交，回滚和崩溃恢复能力的事务安全（ACID 兼容）存储引擎。InnoDB 提供 row level lock，并且也在 SELECT 语句提供一个 Oracle 风格一致的非锁定读。这些特色增加了多用户部署和性能。没有在 InnoDB 中扩大锁定的需要，因为在 InnoDB 中 row level lock 适合非常小的空间。InnoDB 也支持 FOREIGN KEY 约束。在 SQL 查询中，你可以自由地将 InnoDB 类型的表与其它 MySQL 的表的类型混合起来，甚至在同一个查询中也可以混合。

InnoDB 是为在处理巨大数据量时获得最大性能而设计的。它的 CPU 使用效率非常高。

InnoDB 存储引擎已经完全与 MySQL 服务器整合，InnoDB 存储引擎为在内存中缓存数据和索引而维持它自己的缓冲池。 InnoDB 存储它的表 & 索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这与 MyISAM 表不同，比如在 MyISAM 表中每个表被存在分离的文件中。InnoDB 表可以是任何大小，即使在文件尺寸被限制为 2GB 的操作系统上。

许多需要高性能的大型数据库站点上使用了 InnoDB 引擎。著名的 Internet 新闻站点 Slashdot.org 运行在 InnoDB 上。 Mytrix, Inc.在 InnoDB 上存储超过 1TB 的数据，还有一些其它站点在 InnoDB 上处理平均每秒 800 次插入/更新的负荷。

4.1.2.1 InnoDB 特性

- **InnoDB 属性**

1）支持事务，ACID，外键。

2) Row level locks。

3）支持不同的隔离级别。

4）和 MyISAM 相比需要较多的内存和磁盘空间。

5）没有键压缩。

6）数据和索引都缓存在内存 hash 表中。

- **InnoDB 适用范围**

1）需要事务的应用。

2）高并发的应用。

3）自动恢复。

4）较快速的基于主键的操作。

4.1.2.2 InnoDB 优化要点

1）尽量使用 short，integer 的主键。

2) Load/Insert 数据时按主键顺序。如果数据没有按主键排序，先排序然后再进行数据库操作。

3）在 Load 数据是为设置 SET UNIQUE_CHECKS=0，SET FOREIGN_KEY_CHECKS=0，可以避免外键和唯一性约束检查的开销。

4）使用 prefix keys。因为 InnoDB 没有 key 压缩功能。

4.1.2.3 InnoDB 服务器端优化设定

innodb_buffer_pool_size：这是 InnoDB 最重要的设置，对 InnoDB 性能有决定性的影响。默认的设置只有 8M，所以默认的数据库设置下面 InnoDB 性能很差。在只有 InnoDB 存储引擎的数据库服务器上面，可以设置 60-80%的内存。更精确一点，在内存容量允许的情况下面设置比 InnoDB tablespaces 大 10%的内存大小。

innodb_data_file_path：指定表数据和索引存储的空间，可以是一个或者多个文件。最后一个数据文件必须是自动扩充的，也只有最后一个文件允许自动扩充。这样，当空间用完后，自动扩充数据文件就会自动增长（以 8MB 为单位）以容纳额外的数据。例如：innodb_data_file_path=/disk1/ibdata1:900M;/disk2/ibdata2:50M:autoextend 两个数据文件放在不同的磁盘上。数据首先放在 ibdata1 中，当达到 900M 以后，数据就放在

ibdata2 中。一旦达到 50MB，ibdata2 将以 8MB 为单位自动增长。如果磁盘满了，需要在另外的磁盘上面增加一个数据文件。

innodb_autoextend_increment：默认是 8M，如果一次 insert 数据量比较多的话，可以适当增加.

innodb_data_home_dir：放置表空间数据的目录，默认在 mysql 的数据目录，设置到和 MySQL 安装文件不同的分区可以提高性能。

innodb_log_file_size：该参数决定了 recovery speed。太大的话 recovery 就会比较慢，太小了影响查询性能，一般取 256M 可以兼顾性能和 recovery 的速度。

innodb_log_buffer_size：磁盘速度是很慢的，直接将 log 写道磁盘会影响 InnoDB 的性能，该参数设定了 log buffer 的大小，一般 4M。如果有大的 blob 操作，可以适当增大。

innodb_flush_logs_at_trx_commit=2： 该参数设定了事务提交时内存中 log 信息的处理。

1) =1 时，在每个事务提交时，日志缓冲被写到日志文件，对日志文件做到磁盘操作的刷新。Truly ACID。速度慢。

2) =2 时，在每个事务提交时，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新。只有操作系统崩溃或掉电才会删除最后一秒的事务，不然不会丢失事务。

3) =0 时， 日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新。任何 mysqld 进程的崩溃会删除崩溃前最后一秒的事务

innodb_file_per_table：可以存储每个 InnoDB 表和它的索引在它自己的文件中。

transaction-isolation=READ-COMITTED： 如果应用程序可以运行在 READ-COMMITED 隔离级别，做此设定会有一定的性能提升。

innodb_flush_method： 设置 InnoDB 同步 IO 的方式：

1) Default – 使用 fsync（）。

2) O_SYNC 以 sync 模式打开文件，通常比较慢。

3) O_DIRECT，在 Linux 上使用 Direct IO。可以显著提高速度，特别是在 RAID 系统上。避免额外的数据复制和 double buffering（mysql buffering 和 OS buffering）。

innodb_thread_concurrency： InnoDB kernel 最大的线程数。

1) 最少设置为(num_disks+num_cpus)*2。

2) 可以通过设置成 1000 来禁止这个限制

4.2、数据库设计和查询优化

在 MySQL 性能优化中，首先要考虑的就是 Database Schema 设计，这一点是非常重

要的。一个糟糕的 Schema 设计即使在性能调优的 MySQL Server 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。这一节将详细讨论这两方面的优化。

4.2.1 Schema 设计

Schema 的优化取决于将要运行什么样的 query，不同的 query 会有不同的 Schema 优化方案。4.2.2 节将介绍 Query 设计的优化。Schema 设计同样受到预期数据集大小的影响。Schema 设计时主要考虑：标准化，数据类型，索引。

4.2.1.1 标准化

标准化是在数据库中组织数据的过程。其中包括，根据设计规则创建表并在这些表间建立关系；通过取消冗余度与不一致相关性，该设计规则可以同时保护数据并提高数据的灵活性。通常数据库标准化是让数据库设计符合某一级别的范式，通常满足第三范式即可。也有第四范式（也称为 Boyce Codd 范式，BCNF)）与第五范式存在，但是在实际设计中很少考虑。忽视这些规则可能使得数据库的设计不太完美，但这不应影响功能。

- **标准化的特点：**

1）所有的"对象"都在它自己的 table 中，没有冗余。
2）数据库通常由 E-R 图生成。
3）简洁，更新属性通常只需要更新很少的记录。
4) Join 操作比较耗时。
5) Select，sort 优化措施比较少。
6）适用于 OLTP 应用。

- **非标准化的特点：**

1）在一张表中存储很多数据，数据冗余。
2）更新数据开销很大，更新一个属性可能会更新很多表，很多记录。
3）在删除数据是有可能丢失数据。
4) Select，order 有很多优化的选择。
5）适用于 DSS 应用。

标准化和非标准化都有各自的优缺点，通常在一个数据库设计中可以混合使用，一部分表格标准化，一部分表格保留一些冗余数据：

1）对 OLTP 使用标准化，对 DSS 使用非标准化
2）使用物化视图。MySQL 不直接支持该数据库特性，但是可以用 MyISAM 表代替。

3) 冗余一些数据在表格中，例如将 ref_id 和 name 存在同一张表中。但是要注意更新问题。

4) 对于一些简单的对象，直接使用 value 作为建。例如 IP address 等

5) Reference by PRIMARY/UNIQUE KEY。

4.2.1.2 数据类型

最基本的优化之一就是使表在磁盘上占据的空间尽可能小。这能带来性能非常大的提升，因为数据小，磁盘读入较快，并且在查询过程中表内容被处理所占用的内存更少。同时，在更小的列上建索引，索引也会占用更少的资源。

可以使用下面的技术可以使表的性能更好并且使存储空间最小：

1) 使用正确合适的类型，不要将数字存储为字符串。

2) 尽可能地使用最有效(最小)的数据类型。MySQL 有很多节省磁盘空间和内存的专业化类型。

3) 尽可能使用较小的整数类型使表更小。例如，MEDIUMINT 经常比 INT 好一些，因为 MEDIUMINT 列使用的空间要少 25%。

4) 如果可能，声明列为 NOT NULL。它使任何事情更快而且每列可以节省一位。注意如果在应用程序中确实需要 NULL，应该毫无疑问使用它，只是避免 默认地在所有列上有它。

5) 对于 MyISAM 表，如果没有任何变长列(VARCHAR、TEXT 或 BLOB 列)，使用固定尺寸的记录格式。这比较快但是不幸地可能会浪费一些空间。即使你已经用 CREATE 选项让 VARCHAR 列 ROW_FORMAT=fixed，也可以提示想使用固定长度的行。

6) 使用 sample character set，例如 latin1。尽量少使用 utf-8，因为 utf-8 占用的空间是 latin1 的 3 倍。可以在不需要使用 utf-8 的字段上面使用 latin1，例如 mail，url 等。

4.2.1.3 索引

所有 MySQL 列类型可以被索引。对相关列使用索引是提高 SELECT 操作性能的最佳途径。使用索引应该注意以下几点：

1) MySQL 只会使用前缀，例如 key(a, b) …where b=5 将使用不到索引。

2) 要选择性的使用索引。在变化很少的列上使用索引并不是很好，例如性别列。

3) 在 Unique 列上定义 Unique index。

4) 避免建立使用不到的索引。

5) 在 Btree index 中（InnoDB 使用 Btree），可以在需要排序的列上建立索引。

6) 避免重复的索引。

7) 避免在已有索引的前缀上建立索引。例如：如果存在 index（a，b）则去掉 index（a）。

8) 控制单个索引的长度。使用 key（name（8））在数据的前面几个字符建立索引。

9) 越是短的键值越好，最好使用 integer。

10）在查询中要使用到索引（使用 explain 查看），可以减少读磁盘的次数，加速读取数据。

11）相近的键值比随机好。Auto_increment 就比 uuid 好。

12) Optimize table 可以压缩和排序 index，注意不要频繁运行。

13) Analyze table 可以更新数据。

4.2.2 查询设计

查询语句的优化是一个 Case by case 的问题，不同的 sql 有不同的优化方案，在这里我只列出一些通用的技巧。

1）在有 index 的情况下，尽量保证查询使用了正确的 index。可以使用 EXPLAIN select …查看结果，分析查询。

2）查询时使用匹配的类型。例如 select * from a where id=5， 如果这里 id 是字符类型，同时有 index，这条查询则使用不到 index，会做全表扫描，速度会很慢。正确的应该是 … where id=”5”，加上引号表明类型是字符。

3）使用--log-slow-queries –long-query-time=2 查看查询比较慢的语句。然后使用 explain 分析查询，做出优化。

4.3、服务器端优化

4.3.1 MySQL 安装

MySQL 有很多发行版本，最好使用 MySQL AB 发布的二进制版本。也可以下载源代码进行编译安装，但是编译器和类库的一些 bug 可能会使编译完成的 MySQL 存在潜在的问题。

如果安装 MySQL 的服务器使用的是 Intel 公司的处理器，可以使用 intel c++编译的版本。（在相关的技术资料中有如下的性能对比说明：使用 intel C++编译器编译的 MySQL 查询速度比正常版本快 30%左右）。Intel c++编译版本可以在 MySQL 官方网站下载。

4.3.2 服务器设置优化

MySQL 默认的设置性能很差，所以要做一些参数的调整。这一节介绍一些通用的参数调整，不涉及具体的存储引擎（存储引擎 MyISAM，InnoDB 的相关优化已经在 4.1 中介绍）。

--character-set：如果是单一语言使用简单的 character set 例如 latin1。尽量少用

Utf-8，utf-8 占用空间较多。

--memlock：锁定 MySQL 只能运行在内存中，避免 swapping，但是如果内存不够时有可能出现错误。

--max_allowed_packet：要足够大，以适应比较大的 SQL 查询，对性能没有太大影响，主要是避免出现 packet 错误。

--max_connections：server 允许的最大连接。太大的话会出现 out of memory。

--table_cache：MySQL 在同一时间保持打开的 table 的数量。打开 table 开销比较大。一般设置为 512。

--query_cache_size： 用于缓存查询的内存大小。

--datadir：mysql 存放数据的根目录，和安装文件分开在不同的磁盘可以提高一点性能。

4.4、参数调整

4.4.1 内存参数

任何数据系统，内存设置都会直接影响到系统的性能，mysql 也不例外。因为内存的读写速度要远高于磁盘，所以应该尽量让数据的读写直接在内存中进行，但由于系统资源有限，过度分配内存资源有可能造成 SWAP 等引起性能低下的事件，所以需要兼顾服务器的配置进行相关设置。

1．innodb_buffer_pool_size

此参数的作用是设置缓存 innodb 表的索引，数据，插入数据时的缓冲池大小。这个池的作用与 DB2 或 Oracle 的 buffer pool 是基本一样的，本参数设置的合理与否，与数据库总体性能有非常大的关系。

该参数的默认值只有 8M，显然不能适应日常业务的需要，本参数应该遵循以下分配原则：

1）如果是一个专用ＤＢ服务器，那么他可以占到内存的 70%-80%。

2）由于本参数不能动态更改，所以分配前应从全方位进行考虑。

3）要注意资源的平衡，分配过大，会使 Swap 占用过多，致使 Mysql 的查询效率低下。

本参数分配值的使用情况可以根据 show innodb status\G;中的输出去确认使用情况。

```
----------------------
BUFFER POOL AND MEMORY
----------------------
```

```
Total memory allocated 4668764894;
```

2. innodb_additional_mem_pool

本参数用来设置存放 Innodb 的内部目录的内存区域的大小（类似于 Oracle 里 shared pool 中 dictionary cache 部分），由于系统可以自动调整此值，所以不需要过多分配。通常比较大数据设置 16M基本就可满足要求，如果表比较多，可以适当的增大。如果这个值自动增加，会在 error log 中有所记录。可以用 show innodb status\G;去查看运行中的ＤＢ是什么状态（参考 BUFFER POOL AND MEMORY 段中），然后可以调整到适当的值。

```
---------------------
BUFFER POOL AND MEMORY
---------------------
Total memory allocated 4668764894; in additional pool      allocated
16777216
```

根据以上结果，可以参考 in additional pool allocated 1677721 根据具体的参数情况，可以适当的调整。

4.4.2 日志与事务控制参数

关系型数据库的日志与事务管理也是影响性能的重要因素之一，尤其是以事务为主的 OLTP 系统中，日志读写比较频繁，进行合理的设置会降低系统 IO 和其他资源开销，增进性能。

1.innodb_log_file_size

本参数的作用是指定日志的大小。本参数应该遵循以下分配原则：

几个日志成员大小加起来差不多和你的 innodb_buffer_pool_size 相等。上限为每个日值上限大小为 4G.一般控制在几个ＬＯＧ文件相加大小在２Ｇ以内为佳。具体情况还需要以事务大小，数据大小为依据。分配的大小和数据库的写入速度，事务大小，实例崩溃恢复速度有很大的关系。如果日志太小，日志切换会比较频繁，影响性能。而日志过大的话，在进行实例崩溃恢复时，将耗费较长的时间。需要在两者之间进行权衡

2.innodb_log_files_in_group

本参数的作用是指定数据库有几个日志组，默认为 2 个，因为有可能出跨日志的大事务，所以一般来说，建议使用 3-4 个日志组。

3.innodb_log_buffer_size

本参数的作用是设置事务 Log 缓存大小（类似于 Oracle 的 log buffer），一旦提交事务，则将该池中的内容写到磁盘的日志文件上。本参数应该遵循如下分配原则：池里面的内存一般一秒钟写到磁盘一次。具体写入方式和系统的事务提交方式有关。OTLP 系统一般最大指定

为 3 M比较合适。

可以参考以下参数：Innodb_os_log_written，如果这个值增长过快，可以适当的增加innodb_log_buffer_size。另外如果需要处理大量的 Text，或是 Blob 字段，可以考虑增加这个参数的值。

4.innodb_flush_logs_at_trx_commit

本参数的作用是控制事务的提交方式，以下为该参数的详细说明：

（1）这个参数的设置对ｉｎｎｏｄｂ的性能有很大的影响，所以在这里给多说明一下。当这个值为 1 时：innodb 的事务 LOG 在每次提交后写入日值文件，并对日值做刷新到磁盘。这个可以做到不丢任何一个事务。

（2）当这个值为 2 时：在每个提交，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新,在对日志文件的刷新在值为 2 的情况也每秒发生一次。但需要注意的 是，由于进程调用方面的问题，并不能保证每秒 100%的发生。从而在性能上是最快的。但操作系统崩溃或掉电才会删除最后一秒的事务。

（3）当这个值为 0 时：日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新，但是在一个事务提交不做任何操作。mysqld 进程的崩溃会删除崩溃前最后一秒的事务。

这个参数只有 3 个值，０，１，２请确认一下自已能接受的级别。默认为 1，本参数建议只在 Master-Slave 结构中的 Slave 库上进行修改。处于性能考虑，可以设置为 0 或是 2，但会丢失一秒钟的事务。从以上分析，当这个值不为 1 时，可以取得较好的性能，但遇到异常会有损失，所以需要根据自已的情况去衡量。

4.4.3 存储，IO 相关参数

1.innodb_open_files

本参数的作用是限制 Innodb 能打开的表的数据，本参数的默认值为 300，如果库里的表特别多的情况，请增加此参数的值。

2.innodb_flush_method

本参数的作用是设置innodb引擎与操作系统进行IO交互的模式,即刷新日志和数据的方法，innodb_flush_method 有三个值：

（1）Fdatasync（默认值）

InnoDB 使用 fsync()函数去更新日志和数据文件

（2）O_DSYNC

InnoDB 使用 O_SYNC 模式打开并更新日志文件，用 fsync()函数去更新数据文件

（3）O_DIRECT

InnoDB 使用 O_DIRECT 模式打开数据文件，用 fsync()函数去更新日志和数据文件。(不使用操作系统缓存，类似于裸设备机制)

3.innodb_max_dirty_pages_pct

本参数的作用是控制 Innodb 的脏页在缓冲中的百分比，将脏页的百分比控制在参数值之下，比如该参数的值是 50，则脏页在缓冲中最多占 50%，如果超过，将会把脏数据写到磁盘中，此值的范围可以在 1-100 之间,默认为 90.当 Innodb 的内存分配过大，致使 Swap 占用严重时，可以适当的减小调整这个值，使 Swap 空间释放出来。建议此值设置在 90%最小在 15%之间。需要进行一定的权衡，如果设置太大，缓存中每次更新需要置换的数据页太多，太小，可以存放的脏数据页太小，性能受到一定影响。

4.5 语句调优

MySQL 的语句级调优与其他关系型数据库基本相同，主要是合理利用索引，临时表以及对数据类型进行正确的选取

4.5.1 合理使用索引

1.因为要兼顾查询与事务性操作平衡，所以一个表的索引数量最好不要超过 6 个，因为对索引的增加，删除，修改都会重新排序，影响性能，过多的索引会导致额外的性能及存储空间上的开销

2.字段值比较单一时，典型的例如性别这种字段，不宜建立索引，因为如果字段中存在大量重复值的时候，优化器更倾向于进行表扫描，强行建立索引非但没有任何作用，还会浪费空间。

3.MySQL 中也可以使用类似于 Oracle hint 类似的方法强制使用索引，示例：

```
select id from t with(index(索引名)) where num=12
```

4.应尽量避免在 where 子句中对字段进行函数或计算操作，这样会使索引失效。

5.对于经常在谓词（where 条件）中组合使用的字段，可以考虑使用组合索引，这样可以提高查询的性能。

4.5.2 合理使用临时表

1.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只

有主键索引)。

2.避免频繁创建和删除临时表，以减少系统表资源的消耗。

3.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先 create table，然后 insert。

4.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

4.5.3 选择合适的数据类型

1.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引)。

2.避免频繁创建和删除临时表，以减少系统表资源的消耗

3.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先 create table，然后 insert。

4.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

4.6 结构调优

一般来说，对 MySQL 数据库的结构进行合理的设计和优化，是效率最高的优化方式。但是这种优化方式一般需要较大的人力物力投入，在进行之前，一定要做好调研工作，否则将会承担很大的风险。对 MYSQL 数据库进行结构主要利用以下技术:

4.6.1．表分区技术

表分区技术是一种水平分割数据的技术，可以根据不同的条件将一个表分割成许多部分，这样在进行数据访问的时候，数据进行数据扫描时，会缩小范围，增进效率，例如：一个 3 年积累了 3600 万条记录的表，以年度月份为分区条件，分成 36 个分区，这样每个分区有 100 万的数据量。这样进行数据检索时，扫描的数据量只是原来的 1/36。

MySQL 的分区表支持 Range，List，Hash，Key 等分区方式，其中以 RANGE 最为常用:

- Range（范围）– 这种模式允许将数据划分不同范围。例如可以将一个表通过年份划分成若干个分区。
- Hash（哈希）– 这中模式允许通过对表的一个或多个列的 Hash Key 进行计算，最后通过这个 Hash 码不同数值对应的数据区域进行分区。例如可以建立一个对表主键进行分区的表。
- Key（键值）-上面 Hash 模式的一种延伸，这里的 Hash Key 是 MySQL 系统产生的。
- List（预定义列表）– 这种模式允许系统通过预定义的列表的值来对数据进行分割。
- Composite（复合模式） – 以上模式的组合使用。

分区的限制(截止 **5.1.44** 版)

- 只能对数据表的整型列进行分区，或者数据列可以通过分区函数转化成整型列
- 最大分区数目不能超过 1024
- 如果含有唯一索引或者主键，则分区列必须包含在所有的唯一索引或者主键在内
- 不支持外键
- 不支持全文索引（fulltext）

4.6.2．SQL 路由技术

amoeba 是一个以 MySQL 为底层数据存储，并对应用提供 MySQL 协议接口的 proxy。它集中地响应应用的请求，依据用户事先设置的规则，将 SQL 请求发送到特定的数据库上执行。基于此可以实现负载均衡、读写分离、高可用性等需求。

在进行数据库设计的时候，可以将不同的表按照功能的区别放在不同的数据库上，再利用 Amoeba 进行 SQL 路由，从而正确的访问表。这样不仅从内存与处理器资源上对整个系统进行了扩展，同时也在一定程度上解决了传统数据库集群（如 Oracle RAC）存储瓶颈的问题。这种方案本质上是采用了分布式数据库的理念。

读写分离是利用 Mysql 的 Replication 技术，将数据库设计成一个 Master-Slave 的架构。然后将 Master 库的数据不断同步到 Slave 库中。这样，Slave 库除了可以作为 Master 库的一个备份外，也可以利用 Amoeba 将所有查询路由到 Slave 库上执行，作为查询数据库来使用。从而减轻了 Master 库的负载，提升性能。

五、小结

MySQL 数据库的性能优化，是一个相对系统复杂的工作，在此仅对其中的几个方面进行了简单的介绍，仅作为 MySQL 数据库优化的参考。该文档将在后续的实践工作中不断完善。