**MySQL优化**

# 课程计划

1. MySQL介绍
2. MySql流程
3. 查询优化
4. 索引的使用
5. 存储优化
6. 数据库结构优化
7. 硬件优化
8. MySql缓存
9. 服务器参数

# MySQL介绍

MySQL是由MySQLAB公司（目前已经被SUN公司收归麾下）自主研发的，目前IT行业最流行的开放源代码的数据库管理系统之一，它同时也是一个支持多线程高并发多用户的关系型数据库管理系统。

MySQL数据库以其简单高效可靠的特点，在最近短短几年的时间就从一个名不见经传的数据库系统，变成一个在IT行业几乎是无人不知的开源数据库管理系统。

目前MySQL已经成为最为流行的开源关系数据库系统，并且一步一步地占领了原有商业数据库的市场。可以看到Google，Facebook，Twitter，百度，新浪，腾讯，淘宝，网易，久游等绝大多数互联网公司数据库都是用的MySQL数据库，甚至将其作为核心应用的数据库系统。

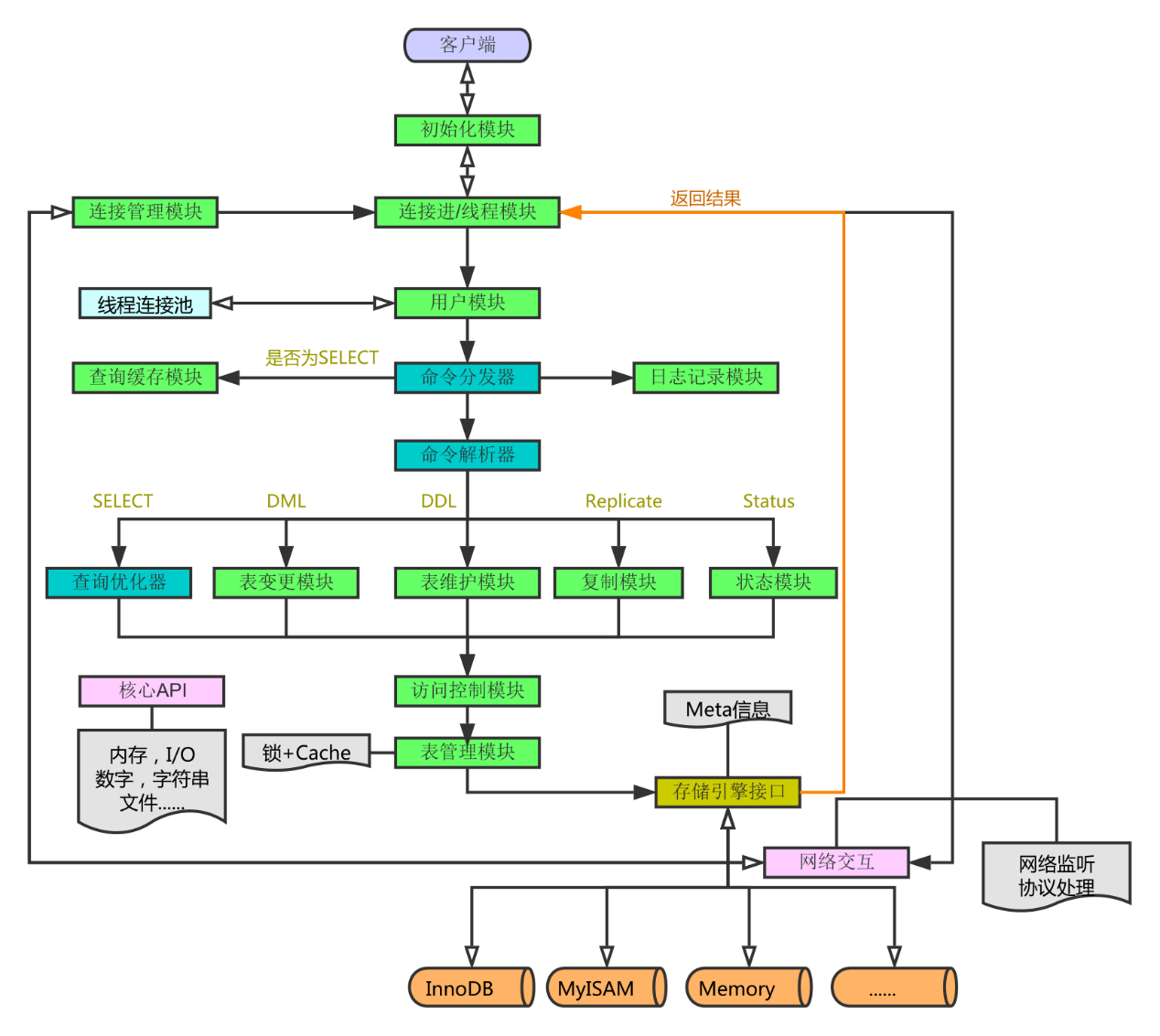
而MySQL数据库也不再仅仅应用于Web项目，其扮演的角色更为丰富。在网络游戏领域中，大部分的后台数据库都采用MySQL数据库，如大家比较熟悉的劲舞团、魔兽世界等。很少能看到有哪个网络游戏数据库不是采用MySQL数据库的。此外，MySQL数据库已成功应用于中国外汇交易中心、中国移动、国家电网等许多项目中。

一个数据库应用系统（这里的数据库应用系统概指所有使用数据库的系统）的性能瓶颈最容易出现在数据的操作方面，而数据库应用系统的大部分数据操作都是通过数据库管理软件所提供的相关接口来完成的。所以数据库管理软件也就很自然的成为了数据库应用系统的性能瓶颈所在，这是当前业界比较普遍的一个看法。

我们将这里进行一个较为全面的分析，让大家了解到一个数据库应用系统的性能到底与哪些地方有关，让大家寻找出各自应用系统的出现性能问题的根本原因，而尽可能清楚的知道该如何去优化自己的应用系统。

# MySQL架构

下图是MySQL架构的总览图



## 查询执行流程

查询执行的流程是怎样的：

1. 连接

1.1客户端发起一条Query请求，监听客户端的‘连接管理模块’接收请求

1.2将请求转发到‘连接进/线程模块’

1.3调用‘用户模块’来进行授权检查

1.4通过检查后，‘连接进/线程模块’从‘线程连接池’中取出空闲的被缓存的连接线程和客户端请求对接，如果失败则创建一个新的连接请求

2. 处理

2.1先查询缓存，检查Query语句是否完全匹配，

2.2查询缓存失败则转交给‘命令解析器’

2.3再转交给对应的模块处理

2.4如果是SELECT查询还会经由‘查询优化器’做大量的优化，生成执行计划

2.5模块收到请求后，通过‘访问控制模块’检查所连接的用户是否有访问目标表和目标字段的权限

2.6有则调用‘表管理模块’，先是查看table cache中是否存在，有则直接对应的表和获取锁，否则重新打开表文件

2.8根据表的meta数据，获取表的存储引擎类型等信息，通过接口调用对应的存储引擎处理

2.9上述过程中产生数据变化的时候，若打开日志功能，则会记录到相应二进制日志文件中

3. 结果

3.1Query请求完成后，将结果集返回给‘连接进/线程模块’

3.2返回的也可以是相应的状态标识，如成功或失败等

3.3‘连接进/线程模块’进行后续的清理工作，并继续等待请求或断开与客户端的连接

# 什么是优化

* 合理安排资源、调整系统参数使MySQL运行更快、更节省资源。
* 优化是多方面的，包括查询、表设计、服务器等。
* 原则：减少系统瓶颈，减少资源占用，增加系统的反应速度。

# 查询优化

在优化MySQL时，通常需要对数据库进行分析。常见的分析手段有慢查询日志，EXPLAIN 分析查询，通过定位分析性能的瓶颈，才能更好的优化数据库系统的性能。

## 慢查询

### 慢查询日志开启

在配置文件my.cnf或my.ini中在[mysqld]一行下面加入两个配置参数

log-slow-queries=/data/mysqldata/slow-query.log

long\_query\_time=5

注：log-slow-queries参数为慢查询日志存放的位置，一般这个目录要有mysql的运行帐号的可写权限，一般都将这个目录设置为mysql的数据存放目录；

long\_query\_time=5中的5表示查询超过五秒才记录；

还可以在my.cnf或者my.ini中添加log-queries-not-using-indexes参数，表示记录下没有使用索引的查询。

### 慢查询分析

我们可以通过打开log文件查看得知哪些SQL执行效率低下

从日志中，可以发现查询时间超过5 秒的SQL，而小于5秒的没有出现在此日志中。

如果慢查询日志中记录内容很多，可以使用mysqldumpslow工具（MySQL客户端安装自带）来对慢查询日志进行分类汇总。mysqldumpslow对日志文件进行了分类汇总，显示汇总后摘要结果。

进入log的存放目录，运行

[root@mysql\_data]#mysqldumpslow slow-query.log

Reading mysql slow query log from slow-query.log

Count: 2 Time=11.00s (22s) Lock=0.00s (0s) Rows=1.0 (2), root[root]@mysql

select count(N) from t\_user;

mysqldumpslow命令

/path/mysqldumpslow -s c -t 10 /database/mysql/slow-query.log

这会输出记录次数最多的10条SQL语句，其中：

-s, 是表示按照何种方式排序，c、t、l、r分别是按照记录次数、时间、查询时间、返回的记录数来排序，ac、at、al、ar，表示相应的倒叙

-t, 是top n的意思，即为返回前面多少条的数据；

-g, 后边可以写一个正则匹配模式，大小写不敏感的；

例如：

/path/mysqldumpslow -s r -t 10 /database/mysql/slow-log

得到返回记录集最多的10个查询。

/path/mysqldumpslow -s t -t 10 -g “left join” /database/mysql/slow-log

得到按照时间排序的前10条里面含有左连接的查询语句。

使用mysqldumpslow命令可以非常明确的得到各种我们需要的查询语句，对MySQL查询语句的监控、分析、优化是MySQL优化非常重要的一步。开启慢查询日志后，由于日志记录操作，在一定程度上会占用CPU资源影响mysql的性能，但是可以阶段性开启来定位性能瓶颈。

## EXPLAIN

在MySQL中可以使用EXPLAIN查看SQL执行计划，用法：EXPLAIN SELECT \* FROM products

### id

SELECT识别符。这是SELECT查询序列号。这个不重要。

### select\_type

表示SELECT语句的类型。

例如：

1. SIMPLE  
   表示简单查询，其中不包含连接查询和子查询。
2. PRIMARY  
   表示主查询，或者是最外面的查询语句。
3. UNION  
   表示连接查询的第2个或后面的查询语句。

### table

表示查询的表。

### type

表示表的连接类型。

以下的连接类型的顺序是从最佳类型到最差类型：

1. system  
   表仅有一行，这是const类型的特列，平时不会出现，这个也可以忽略不计。
2. const  
   数据表最多只有一个匹配行，因为只匹配一行数据，所以很快，常用于
3. eq\_ref  
   mysql手册是这样说的:"对于每个来自于前面的表的行组合，从该表中读取一行。这可能是最好的联接类型，除了const类型。它用在一个索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY"。eq\_ref可以用于使用=比较带索引的列。
4. ref  
   查询条件索引既不是UNIQUE也不是PRIMARY KEY的情况。ref可用于=或<或>操作符的带索引的列。
5. ref\_or\_null  
   该联接类型如同ref，但是添加了MySQL可以专门搜索包含NULL值的行。在解决子查询中经常使用该联接类型的优化。

上面这五种情况都是很理想的索引使用情况。

1. index\_merge  
   该联接类型表示使用了索引合并优化方法。在这种情况下，key列包含了使用的索引的清单，key\_len包含了使用的索引的最长的关键元素。
2. unique\_subquery  
   该类型替换了下面形式的IN子查询的ref: value IN (SELECT primary\_key FROM single\_table WHERE some\_expr)   
   unique\_subquery是一个索引查找函数,可以完全替换子查询,效率更高。
3. index\_subquery  
   该联接类型类似于unique\_subquery。可以替换IN子查询,但只适合下列形式的子查询中的非唯一索引: value IN (SELECT key\_column FROM single\_table WHERE some\_expr)
4. range  
   只检索给定范围的行,使用一个索引来选择行。
5. index  
   该联接类型与ALL相同,除了只有索引树被扫描。这通常比ALL快,因为索引文件通常比数据文件小。
6. ALL  
   对于每个来自于先前的表的行组合,进行完整的表扫描。（性能最差）

### possible\_keys

指出MySQL能使用哪个索引在该表中找到行。

如果该列为NULL，说明没有使用索引，可以对该列创建索引来提高性能。

### key

显示MySQL实际决定使用的键(索引)。如果没有选择索引,键是NULL。

### key\_len

显示MySQL决定使用的键长度。如果键是NULL,则长度为NULL。

注意：key\_len是确定了MySQL将实际使用的索引长度。

### ref

显示使用哪个列或常数与key一起从表中选择行。

### rows

显示MySQL认为它执行查询时必须检查的行数。

### Extra

该列包含MySQL解决查询的详细信息

* Distinct:MySQL发现第1个匹配行后,停止为当前的行组合搜索更多的行。
* Not exists:MySQL能够对查询进行LEFT JOIN优化,发现1个匹配LEFT JOIN标准的行后,不再为前面的的行组合在该表内检查更多的行。
* range checked for each record (index map: #):MySQL没有发现好的可以使用的索引,但发现如果来自前面的表的列值已知,可能部分索引可以使用。
* Using filesort:MySQL需要额外的一次传递,以找出如何按排序顺序检索行。
* Using index:从只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的列信息。
* Using temporary:为了解决查询,MySQL需要创建一个临时表来容纳结果。
* Using where:WHERE 子句用于限制哪一个行匹配下一个表或发送到客户。
* Using sort\_union(...), Using union(...), Using intersect(...):这些函数说明如何为index\_merge联接类型合并索引扫描。
* Using index for group-by:类似于访问表的Using index方式,Using index for group-by表示MySQL发现了一个索引,可以用来查 询GROUP BY或DISTINCT查询的所有列,而不要额外搜索硬盘访问实际的表。

# 索引使用

### MySQL索引

#### B-Tree索引

一般来说，MySQL中的B-Tree索引的物理文件大多都是以二叉树的结构来存储的，也就是所有实际需要的数据都存放于树的叶子节点，而且到任何一个叶子节点的最短路径的长度都是完全相同的。

#### R-Tree索引

RTREE在mysql很少使用，支持该类型的存储引擎只有MyISAM、BDb、InnoDb、NDb、Archive几种。相对于BTREE，RTREE的优势在于范围查找.

#### Hash索引

Hash索引在MySQL中使用的并不是很多，目前主要是Memory存储引擎使用，而且在Memory存储引擎中将Hash索引作为默认的索引类型。所谓Hash索引，实际上就是通过一定的Hash算法，将需要索引的键值进行Hash运算，然后将得到的Hash值存入一个Hash表中。然后每次需要检索的时候，都会将检索条件进行相同算法的Hash运算，然后再和Hash表中的Hash值进行比较并得出相应的信息。

Hash索引仅仅只能满足“=”,“IN”和“<=>”查询，不能使用范围查询；

Hash索引无法被利用来避免数据的排序操作；

Hash索引不能利用部分索引键查询；

Hash索引在任何时候都不能避免表扫面；

Hash索引遇到大量Hash值相等的情况后性能并不一定就会比B-Tree索引高；

#### Full-text索引

Full-text索引也就是我们常说的全文索引，目前在MySQL中仅有MyISAM存储引擎支持，而且也并不是所有的数据类型都支持全文索引。目前来说，仅有CHAR，VARCHAR和TEXT这三种数据类型的列可以建Full-text索引。

### 创建索引

是否需要创建索引，几点原则：

* 较频繁的作为查询条件的字段应该创建索引；
* 唯一性太差的字段不适合单独创建索引，即使频繁作为查询条件；
* 更新非常频繁的字段不适合创建索引；
* 不会出现在WHERE子句中的字段不该创建索引；

索引能够极大的提高数据检索效率，也能够改善排序分组操作的性能，但是我们不能忽略的一个问题就是索引是完全独立于基础数据之外的一部分数据，更新数据会带来的IO量和调整索引所致的计算量的资源消耗。

### 使用索引

#### 使用联合索引的查询

MySQL可以为多个字段创建索引，一个索引可以包括16个字段。对于联合索引，只有查询条件中使用了这些字段中第一个字段时，索引才会生效。

#### 使用OR关键字的查询

查询语句的查询条件中只有OR关键字，且OR前后的两个条件中的列都是索引时，索引才会生效，否则，索引不生效。

# 存储优化

存储数据时，影响存储速度的主要是索引、唯一性校验、一次存储的数据条数等。

存储数据的优化，不同的存储引擎优化手段不一样，在MySQL中常用的存储引擎有，MyISAM和InnoDB，两者的区别：

## 存储引擎介绍

### MyISAM存储引擎

MyISAM存储引擎是一种非事务性的引擎，提供高速存储和检索，以及全文搜索能力，适合数据仓库等查询频繁的应用。

每一个表都被存放为三个以表名命名的物理文件。有存放表结构定义信息的.frm文件，还有存放了表的数据.MYD文件和存放索引数据的.MYI文件。

### Innodb 存储引擎

Innodb 存储引擎是事务安全的, 因此如果需要一个事务安全的存储引擎,建议使用它。如果你的数据执行大量的INSERT或UPDATE，出于性能方面的考虑应该使用InnoDB表。

InnoDB 给 MySQL 提供了具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。InnoDB 提供了行锁(locking on row level)，提供与 Oracle 类型一致的不加锁读取(non-locking read in SELECTs)。这些特性均提高了多用户并发操作的性能表现。

在InnoDB表中不需要扩大锁定(lock escalation)，因为 InnoDB 的列锁定(row level locks)适宜非常小的空间。InnoDB 是 MySQL 上提供外键约束(FOREIGN KEY constraints)的表引擎。

InnoDB 的设计目标是处理大容量数据库系统，它的 CPU 利用率是其它基于磁盘的关系数据库引擎所不能比的。在技术上，InnoDB 是一套放在 MySQL 后台的完整数据库系统，InnoDB 在主内存中建立其专用的缓冲池用于高速缓冲数据和索引。

InnoDB 把数据和索引存放在表空间里，可能包含多个文件，这与MyISAM不一样。InnoDB 表的大小只受限于操作系统的文件大小，一般为 2 GB。InnoDB所有的表都保存在同一个数据文件 ibdata1 中(也可能是多个文件，或者是独立的表空间文件),相对来说比较不好备份。备份的方案可以是拷贝数据文件、备份 binlog，或者用 mysqldump。

## MyISAM和Innodb的区别

InnoDB和MyISAM是许多人在使用MySQL时最常用的两个表类型，这两个表类型各有优劣，视具体应用而定。基本的差别为：MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持。MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快，但是不提供事务支持，而InnoDB提供事务支持已经外部键等高级数据库功能。

### 具体实现的差别：

* MyISAM是非事务安全型的，而InnoDB是事务安全型的。
* MyISAM锁的粒度是表级，而InnoDB支持行级锁定。
* MyISAM支持全文类型索引，而InnoDB不支持全文索引。
* MyISAM相对简单，所以在效率上要优于InnoDB，小型应用可以考虑使用MyISAM。
* MyISAM表是保存成文件的形式，在跨平台的数据转移中使用MyISAM存储会省去不少的麻烦。
* InnoDB表比MyISAM表更安全，可以在保证数据不会丢失的情况下，切换非事务表到事务表（alter table tablename type=innodb）。

### 应用场景

* 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。如果应用中需要执行大量的SELECT查询，那么MyISAM是更好的选择。
* InnoDB用于事务处理应用程序，具有众多特性，包括ACID事务支持。如果应用中需要执行大量的INSERT或UPDATE操作，则应该使用InnoDB，这样可以提高多用户并发操作的性能。

## MyISAM存储优化

### 禁用索引

对于非空表，插入记录时，MySQL会根据表的索引对插入的记录建立索引。如果插入大量数据，建立索引会降低插入数据速度。

为了解决这个问题，可以在批量插入数据之前禁用索引，数据插入完成后再开启索引。

禁用索引的语句：

*ALTER TABLE table\_name DISABLE KEYS*

开启索引语句：

*ALTER TABLE table\_name ENABLE KEYS*

对于空表批量插入数据，则不需要进行操作，因为MyISAM引擎的表是在导入数据后才建立索引。

### 禁用唯一性检查

唯一性校验会降低插入记录的速度，可以在插入记录之前禁用唯一性检查，插入数据完成后再开启。

禁用唯一性检查的语句：*SET UNIQUE\_CHECKS = 0;*

开启唯一性检查的语句：SET UNIQUE\_CHECKS = 1;

### 批量插入数据

插入数据时，可以使用一条INSERT语句插入一条数据，也可以插入多条数据。





第二种方式的插入速度比第一种方式快。

### 使用LOAD DATA INFILE

当需要批量导入数据时，使用LOAD DATA INFILE语句比INSERT语句插入速度快很多。

## InnoDB

### 禁用唯一性检查

用法和MyISAM一样。

### 禁用外键检查

插入数据之前执行禁止对外键的检查，数据插入完成后再恢复，可以提供插入速度。

禁用：SET foreign\_key\_checks = 0;

开启：SET foreign\_key\_checks = 1;

### 禁止自动提交

插入数据之前执行禁止事务的自动提交，数据插入完成后再恢复，可以提高插入速度。

禁用：SET autocommit = 0;

开启：SET autocommit = 1;

# 数据库结构优化

## 优化表结构

* + 尽量将表字段定义为NOT NULL约束，这时由于在MySQL中含有空值的列很难进行查询优化，NULL值会使索引以及索引的统计信息变得很复杂。
  + 对于只包含特定类型的字段，可以使用enum、set 等符合数据类型。
  + 数值型字段的比较比字符串的比较效率高得多，字段类型尽量使用最小、最简单的数据类型。例如P地址可以使用int类型。
  + 尽量使用TINYINT、SMALLINT、MEDIUM\_INT作为整数类型而非INT，如果非负则加上UNSIGNED
  + VARCHAR的长度只分配真正需要的空间
  + 尽量使用TIMESTAMP而非DATETIME，
  + 单表不要有太多字段，建议在20以内
  + 合理的加入冗余字段可以提高查询速度。

## 表拆分

### 垂直拆分

垂直拆分按照字段进行拆分，其实就是把组成一行的多个列分开放到不同的表中，这些表具有不同的结构，拆分后的表具有更少的列。例如用户表中的一些字段可能经常访问，可以把这些字段放进一张表里。另外一些不经常使用的信息就可以放进另外一张表里。

插入的时候使用事务，也可以保证两表的数据一致。缺点也很明显，由于拆分出来的两张表存在一对一的关系，需要使用冗余字段，而且需要join操作，我们在使用的时候可以分别取两次，这样的来说既可以避免join操作，又可以提高效率。

### 水平拆分

水平拆分按照行进行拆分，常见的就是分库分表。以用户表为例，可以取用户ID，然后对ID取10的余数，将用户均匀的分配进这 0-9这10个表中。查找的时候也按照这种规则，又快又方便。

有些表业务关联比较强，那么可以使用按时间划分的。例如每天的数据量很大，需要每天新建一张表。这种业务类型就是需要高速插入，但是对于查询的效率不太关心。表越大，插入数据所需要索引维护的时间也就越长。

## 分区

使用分区是大数据处理后的产物。比如系统用户的注册推广等等，会产生海量的日志，当然也可以按照时间水平拆分，建立多张表。但在实际操作中，容易发生忘记切换表导致数据错误。

分区适用于例如日志记录，查询少。一般用于后台的数据报表分析。对于这些数据汇总需求，需要很多日志表去做数据聚合，我们能够容忍1s到2s的延迟，只要数据准确能够满足需求就可以。

MySQL主要支持4种模式的分区：range分区、list预定义列表分区，hash 分区，key键值分区。

## 读写分离

大型网站会有大量的并发访问，如果还是传统的数据结构，或者只是单单靠一台服务器扛，如此多的数据库连接操作，数据库必然会崩溃，数据丢失的话，后果更是不堪设想。这时候，我们需要考虑如何减少数据库的联接。

我们发现一般情况对数据库而言都是“读多写少”，也就说对数据库读取数据的压力比较大，这样分析可以采用数据库集群的方案。其中一个是主库，负责写入数据，我们称为写库；其它都是从库，负责读取数据，我们称为读库。这样可以缓解一台服务器的访问压力

## 数据库集群

如果访问量非常大，虽然使用读写分离能够缓解压力，但是一旦写操作一台服务器都不能承受了，这个时候我们就需要考虑使用多台服务器实现写操作。

例如可以使用MyCat搭建MySql集群，对ID求3的余数，这样可以把数据分别存放到3台不同的服务器上，由MyCat负责维护集群节点的使用。

# 硬件优化

是服务器的硬件性能直接决定着MySQL数据库的性能，硬件的性能瓶颈，直接决定MySQL数据库的运行速度和效率。

可以从以下几个方面考虑：

1. 配置较大的内存。足够大的内存，是提高MySQL数据库性能的方法之一。内存的IO比硬盘快的多，可以增加系统的缓冲区容量，使数据在内存停留的时间更长，以减少磁盘的IO。
2. 磁盘I/O相关

* 使用SSD或者PCIe SSD设备，至少获得数百倍甚至万倍的IOPS提升；
* 购置阵列卡同时配备CACHE及BBU模块，可明显提升IOPS
* 尽可能选用RAID-10，而非RAID-5
* 使用机械盘的话，尽可能选择高转速的，例如选用15000RPM，而不是7200RPM的盘

1. 配置CPU相关

　　 在服务器的BIOS设置中，可调整下面的几个配置：

* 选择Performance Per Watt Optimized(DAPC)模式，发挥CPU最大性能；
* 关闭C1E和C States等选项，提升CPU效率；
* Memory Frequency（内存频率）选择Maximum Performance；

# MySQL缓存

为了提高查询速度，我们可以通过不同的方式去缓存我们的结果从而提高响应效率。当我们的数据库打开了Query Cache（简称QC）功能后，数据库在执行SELECT语句时，会将其结果放到QC中，当下一次处理同样的SELECT请求时，数据库就会从QC取得结果，而不需要去数据表中查询。如果缓存命中率非常高的话，有测试表明在极端情况下可以提高效率238%。

但一个缓存机制是否有效，效果如何，却是一个需要好好思考的问题。Query Cache有如下规则，如果数据表被更改，那么和这个数据表相关的全部Cache全部都会无效，并删除之。这里“数据表更改”包括: INSERT, UPDATE, DELETE, TRUNCATE, ALTER TABLE, DROP TABLE, or DROP DATABASE等。

举个例子，如果数据表item访问频繁，那么意味着它的很多数据会被QC缓存起来，但是每一次item数据表的更新，无论更新是不是影响到了cache 的数据，都会将全部和item表相关的cache清除。如果你的数据表更新频繁的话，那么Query Cache将会成为系统的负担。有实验表明，糟糕时，QC会降低系统13%的处理能力。

### 全局缓存

数据库属于IO密集型的应用程序，其主职责就是数据的管理及存储工作。而我们知道，从内存中读取一个数据库的时间是微秒级别，而从一块普通硬盘上读取一个 IO是在毫秒级别，二者相差3个数量级。所以，要优化数据库，首先第一步需要优化的就是IO，尽可能将磁盘IO转化为内存IO,也就是使用缓存

启动MySQL时就要分配并且总是存在的全局缓存，可以在MySQL的my.conf或者my.ini文件的[mysqld]组中配置。

目前有：key\_buffer\_size(默认值：402653184,即384M)、innodb\_buffer\_pool\_size(默认值：134217728即：128M)、innodb\_additional\_mem\_pool\_size（默认值：8388608即：8M）、innodb\_log\_buffer\_size(默认值：8388608即：8M)、query\_cache\_size(默认值：33554432即：32M)等五个。总共：560M.

#### key\_buffer\_size

key\_buffer\_size是用于索引块的缓冲区大小，增加它可得到更好处理的索引(对所有读和多重写)，对MyISAM表性能影响最大的一个参数。如果你使它太大，系统将开始换页并且真的变慢了。

严格说是它决定了数据库索引处理的速度，尤其是索引读的速度。对于内存在4GB左右的服务器该参数可设置为256M或384M.

#### innodb\_buffer\_pool\_size

innodb\_buffer\_pool\_size:主要针对InnoDB表性能影响最大的一个参数。功能与Key\_buffer\_size一样。InnoDB占用的内存，除innodb\_buffer\_pool\_size用于存储页面缓存数据外，另外正常情况下还有大约8%的开销，主要用在每个缓存页帧的描述、adaptive hash等数据结构，如果不是安全关闭，启动时还要恢复的话，还要另开大约12%的内存用于恢复，两者相加就有差不多21%的开销。

#### innodb\_additional\_mem\_pool\_size

innodb\_additional\_mem\_pool\_size 设置了InnoDB存储引擎用来存放数据字典信息以及一些内部数据结构的内存空间大小，所以当我们一个MySQL Instance中的数据库对象非常多的时候，是需要适当调整该参数的大小以确保所有数据都能存放在内存中提高访问效率的。

#### innodb\_log\_buffer\_size

innodb\_log\_buffer\_size这是InnoDB存储引擎的事务日志所使用的缓冲区。类似于Binlog Buffer

InnoDB在写事务日志的时候，为了提高性能，也是先将信息写入Innofb Log Buffer中，当满足innodb\_flush\_log\_trx\_commit参数所设置的相应条件(或者日志缓冲区写满)之后，才会将日志写到文件 (或者同步到磁盘)中。可以通过innodb\_log\_buffer\_size 参数设置其可以使用的最大内存空间。

InnoDB 将日志写入日志磁盘文件前的缓冲大小。理想值为 1M 至 8M。大的日志缓冲允许事务运行时不需要将日志保存入磁盘而只到事务被提交(commit)。 因此，如果有大的事务处理，设置大的日志缓冲可以减少磁盘I/O。 这个参数实际上还和另外的flush参数相关。一般来说不建议超过32MB

#### query\_cache\_size

query\_cache\_size: 主要用来缓存MySQL中的ResultSet，也就是一条SQL语句执行的结果集，所以仅仅只能针对select语句。

当我们打开了 Query Cache功能，MySQL在接受到一条select语句的请求后，如果该语句满足Query Cache的要求，MySQL会直接根据预先设定好的HASH算法将接受到的select语句以字符串方式进行hash，然后到Query Cache中直接查找是否已经缓存。如果已经在缓存中，该select请求就会直接将数据返回，从而省略了后面所有的步骤(如SQL语句的解析，优化器优化以及向存储引擎请求数据等)，极大的提高性能。

当然，Query Cache也有一个致命的缺陷，那就是当某个表的数据有任何任何变化，都会导致所有引用了该表的select语句在Query Cache中的缓存数据失效。所以，当我们的数据变化非常频繁的情况下，使用Query Cache可能会得不偿失

### 局部缓存

除了全局缓冲，MySql还会为每个连接发放连接缓冲。个连接到MySQL服务器的线程都需要有自己的缓冲。大概需要立刻分配256K，甚至在线程空闲时，它们使用默认的线程堆栈，网络缓存等。事务开始之后，则需要增加更多的空间。运行较小的查询可能仅给指定的线程增加少量的内存消耗，然而如果对数据表做复杂的操作例如扫描、排序或者需要临时表，则需分配大约read\_buffer\_size，

sort\_buffer\_size，read\_rnd\_buffer\_size，tmp\_table\_size 大小的内存空间. 不过它们只是在需要的时候才分配，并且在那些操作做完之后就释放了。

#### read\_buffer\_size

read\_buffer\_size是MySql读入缓冲区大小。对表进行顺序扫描的请求将分配一个读入缓冲区，MySql会为它分配一段内存缓冲区。read\_buffer\_size变量控制这一缓冲区的大小。如果对表的顺序扫描请求非常频繁，并且你认为频繁扫描进行得太慢，可以通过增加该变量值以及内存缓冲区大小提高其性能.

#### sort\_buffer\_size

sort\_buffer\_size是MySql执行排序使用的缓冲大小。如果想要增加ORDER BY的速度，首先看是否可以让MySQL使用索引而不是额外的排序阶段。如果不能，可以尝试增加sort\_buffer\_size变量的大小

#### read\_rnd\_buffer\_size

read\_rnd\_buffer\_size 是MySql的随机读缓冲区大小。当按任意顺序读取行时(例如，按照排序顺序)，将分配一个随机读缓存区。进行排序查询时，MySql会首先扫描一遍该缓冲，以避免磁盘搜索，提高查询速度，如果需要排序大量数据，可适当调高该值。但MySql会为每个客户连接发放该缓冲空间，所以应尽量适当设置该值，以避免内存开销过大。

#### tmp\_table\_size

tmp\_table\_size是MySql的heap （堆积）表缓冲大小。所有联合在一个DML指令内完成，并且大多数联合甚至可以不用临时表即可以完成。大多数临时表是基于内存的(HEAP)表。具有大的记录长度的临时表 (所有列的长度的和)或包含BLOB列的表存储在硬盘上。

如果某个内部heap（堆积）表大小超过tmp\_table\_size，MySQL可以根据需要自动将内存中的heap表改为基于硬盘的MyISAM表。还可以通过设置tmp\_table\_size选项来增加临时表的大小。也就是说，如果调高该值，MySql同时将增加heap表的大小，可达到提高联接查询速度的效果。

#### record\_buffer:

record\_buffer每个进行一个顺序扫描的线程为其扫描的每张表分配这个大小的一个缓冲区。如果你做很多顺序扫描，你可能想要增加该值。

### 其它缓存

#### table\_cache

TABLE\_CACHE(5.1.3及以后版本又名TABLE\_OPEN\_CACHE)，table\_cache指定表高速缓存的大小。每当MySQL访问一个表时，如果在表缓冲区中还有空间，该表就被打开并放入其中，这样可以更快地访问表内容。

不能盲目地把table\_cache设置成很大的值。如果设置得太高，可能会造成文件描述符不足，从而造成性能不稳定或者连接失败。

4.2.3.2 thread\_cache\_size (服务器线程缓存)

默认的thread\_cache\_size=8，,这个值表示可以重新利用保存在缓存中线程的数量,当断开连接时如果缓存中还有空间,那么客户端的线程将被放到缓存中,如果线程重新被请求，那么请求将从缓存中读取,如果缓存中是空的或者是新的请求，那么这个线程将被重新创建,如果有很多新的线程，

增加这个值可以改善系统性能.通过比较 Connections 和 Threads\_created 状态的变量，可以看到这个变量的作用。

# MySQL服务器参数

通过优化MySQL的参数可以提高资源利用率，从而达到提高MySQL服务器性能的目的。MySQL的配置参数都在my.conf或者my.ini文件的[mysqld]组中，常用的参数如下：

## back\_log

back\_log值指出在MySQL暂时停止回答新请求之前的短时间内多少个请求可以被存在堆栈中（每个连接256kb,占用：125M）。也就是说，如果MySql的连接数据达到max\_connections时，新来的请求将会被存在堆栈中，以等待某一连接释放资源，该堆栈的数量即back\_log，如果等待连接的数量超过back\_log，将不被授予连接资源。

## wait\_timeout

当MySQL连接闲置超过一定时间后将会被强行关闭。MySQL默认的wait-timeout值为8个小时。

设置这个值是非常有意义的，比如你的网站有大量的MySQL链接请求（每个MySQL连接都是要内存资源开销的），由于你的程序的原因有大量的连接请求空闲啥事也不干，白白占用内存资源，或者导致MySQL超过最大连接数从来无法新建连接导致“Too many connections”的错误。在设置之前你可以查看一下你的MYSQL的状态（可用showprocesslist)，如果经常发现MYSQL中有大量的Sleep进程，则需要 修改wait-timeout值了。

## max\_connections

max\_connections是指MySql的最大连接数，如果服务器的并发连接请求量比较大，建议调高此值，以增加并行连接数量，当然这建立在机器能支撑的情况下，因为如果连接数越多，介于MySql会为每个连接提供连接缓冲区，就会开销越多的内存，所以要适当调整该值，不能盲目提高设值。

MySQL服务器允许的最大连接数16384

## max\_user\_connections

max\_user\_connections是指每个数据库用户的最大连接针对某一个账号的所有客户端并行连接到MYSQL服务的最大并行连接数。简单说是指同一个账号能够同时连接到mysql服务的最大连接数。设置为0表示不限制。

## thread\_concurrency

thread\_concurrency的值的正确与否, 对mysql的性能影响很大, 在多个cpu(或多核)的情况下，错误设置了thread\_concurrency的值, 会导致mysql不能充分利用多cpu(或多核), 出现同一时刻只能一个cpu(或核)在工作的情况。thread\_concurrency应设为CPU核数的2倍。

## skip-name-resolve

skip-name-resolve：禁止MySQL对外部连接进行DNS解析，使用这一选项可以消除MySQL进行DNS解析的时间。但需要注意，如果开启该选项，则所有远程主机连接授权都要使用IP地址方式，否则MySQL将无法正常处理连接请求！

## default-storage-engine

default-storage-engine= InnoDB(设置InnoDB类型，另外还可以设置MyISAM类型)设置创建数据库及表默认存储类型