链滴

MySQL 优化建议

作者：Hefery

原文链接：https://ld246.com/article/1641180096325

来源网站：链滴

MySQL性能影响

数据库设计、SQL查询速度

- QPS：单位时间内所处理的SQL查询量
- TPS：单位时间内所处理的事务量
- 并发量：同时处理的查询请求的量

大量的并发和超高的CPU使用率风险：

大量的并发：数据库连接数被占满（max connections默认100）

超高的CPU使用率：因CPU资源耗尽而出现宕机

MySQL

存储引擎

参数配置

服务器硬件

- 内存：高并发：数据库连接数被占满
- CPU：高CPU使用率：CPU资源耗尽而宕机（不支持多CPU对同一SQL并发处理）
- 磁盘IO：PCIe > SSD > Raid10 > 磁盘 > SAN

磁盘IO性能突然下降（使用更快的磁盘设备）

其它大量消耗磁盘性能的计划任务（调整计划任务，做好磁盘维护）

网卡流量

网卡IO：减少从服务器数量、分级缓存、避免使用“select *”、分离业务网络和服务器网络

- 减少从服务器的数量
- 进行分级缓存
- 避免使用" select * "进行查询
- 分离业务网络和服务器网络

大表

大表：记录行数巨大，单表超过千万行 或 表数据文件巨大，表数据文件超过10G

影响数据库性能

- 慢查询：很难在一定的时间内过滤出所需要的数据
- DDL：

建立索引需要很长的时间，建立索引会锁表(MySQL<5.5)、不会锁表但会引起主从延迟(MySQL>=5.5)

修改表结构需要长时间锁表，会造成长时间的主从延迟，影响正常的数据操作

处理大表

- 分库分表

难点：分表主键的选择；分表后跨分区数据的查询和统计

- 大表的历史数据归档

优点：减少对前后端业务的影响

难点：归档时间点的选择；如何进行归档操作

大事务

大事务：运行时间比较长，操作的数据比较多的事务

风险

- 锁定太多的数据，造成大量的阻塞和锁超时
- 回滚时所需时间比较长
- 执行时间长，容易造成主从延迟

处理大事务

- 避免一次处理太多的数据
- 移出不必要在事务中的 SELECT 操作

MySQL常用存储引擎

MyISAM

MySQL 5.5之前默认存储引擎

MyISAM 存储引擎表由 MYD 和 MYI 组成（frm：记录表结构信息）

特性

- 并发性和锁级别：读或写都需锁表，并发性不友好
- 表损坏修复：check table 表名; repair table 表名;
- MyISAM支持的索引类型
- MyISAM支持数据压缩：myisampack命令，myisampack -b -f XXX.MYI

限制

- 版本 < MySQL 5.0时默认表大小为 4G，存储大表需修改 MAX_POWS和AVG_ROW_LENGTH
- 版本 > MySQL 5.0时默认表大小为 256TB

场景

- 非事务型应用
- 只读类应用：频繁使用全表 count 语句；增删改频率不高，查询非常频繁
- 空间类应用

Innodb

MySQL5.5之前默认存储引擎

Innodb使用**表空间**进行数据存储：show variables like 'Innodb_file_per_table'

- ON：独立表空间，tablename.ibd
- OFF：系统表空间，ibdataX

特性

- 事务性存储引擎，完全支持事务ACID特性
- 支持行级锁，最大程度地支持并发，行级锁是由存储引擎层实现的
- Redo Log和Undo Log

限制

- 系统表空间无法简单地收缩文件大小
- 独立表空间通过 optimize table 命令收缩系统文件
- 系统表空间会产生 IO 瓶颈
- 独立表空间可以同时向多个文件刷新数据

场景

- 增删改操作频繁
- 可靠性要求较高，支持事务

建议

- 使用独立表
- 表转移：系统表转到独立表
    1. 使用mysqldump导出所有的数据库表数据
    2. 停止MySQL服务器，修改参数，删除Innodb相关文件
    3. 重启MySQL服务，重建Innodb系统表空间
    4. 重新导入数据

选择存储引擎

- 事务支持
- 数据备份

- 崩溃恢复
- 引擎特性

存储引擎	事务	锁粒度	应用
MyISAM	不支持	并发插入的表级锁	select、insert 写操作频繁
MRG_MyISAM	不支持	并发插入的表级锁	数据仓库局查找过多
Innodb	支持	MVCC行级锁	事务处理
Archive	不支持	行级锁	日志记录要随机读取、更新、删除
Ndb cluster	支持	行级锁	高可用部分应用

MySQL优化建议

原则

- 减少系统的瓶颈
- 减少资源的占用
- 增加系统的反应速度

优化方面

- 找出系统的瓶颈，提高 MySQL 数据库整体的性能
- 需要合理的结构设计和参数调整，以提高用户操作响应的速度
- 尽可能节省系统资源，以便系统可以提供更大负荷的服务

优化数据库结构

数据库结构优化目的

- 减少数据冗余
- 尽量避免数据维护中出现更新，插入和删除异常

数据库设计步骤

- 需求分析：全面了解产品设计的存储需求、数据处理需求、数据的安全性和完整性
- 逻辑设计：设计数据的逻辑存储结构，即数据实体之间的逻辑关系
- 物理设计：跟据所使用的数据库特点进行表结构设计
- 维护优化：跟据实际情况对索引、存储结构等进行优化

数据库范式

- 第一范式：数据库表中的所有字段都只具有单一属性，单一属性的列是由基本的数据类型所构成的
- 第二范式：要求一个表中只具有一个业务主键，符合第二范式的表中不能存在非主键列对只对部分键的依赖关系
- 第三范式：每一个非主属性既不部分依赖于也不传递依赖于业务主键，也就是在第二范式的基础上除了非主属性对主键的传递依赖

不要过分的反范式化为表建立太多的列

不要过分的范式化造成太多的表关联

将字段很多且部分字段不频繁使用，可以将这些字段分离出一张新表

分析经常联合查询的表的字段，使用这些字段建立中间表，将原来联合查询的表数据插入到中间表，用中间表查询提高查询效率

优化插入记录的速度

- MyISAM

禁用索引：插入前先禁用索引，插完再恢复索引

禁用唯一性检查：先禁用SET UNIQUE_CHECKS=0，再开启SET UNIQUE_CHECKS=1

使用批量插入：使用INSERT插入多条数据

- InnoDB

禁用唯一性检查：先禁用SET UNIQUE_CHECKS=0，再开启SET UNIQUE_CHECKS=1

禁用外键检查：先禁用SET foreign_key_checks=0，再开启SET foreign_key_checks=1

禁用自动提交：先禁用SET autocommit=0，再开启SET autocommit=1

优化表

- 分析表：ANALYZE TABLE tbl_name

Table：库名.tbl_name

Op：analyze，进行分析操作

Msg_type：状态status、信息info、注意note、警告warning、错误error

Msg_text：OK，显示信息

- 检查表：CHECK TABLE tbl_name [QUICK|FAST|MEDIUM|EXTENDED|CHANGED]

Table：库名.tbl_name

Op：check，进行检查操作

Msg_type：状态status、信息info、注意note、警告warning、错误error

Msg_text：OK，显示信息

- 优化表：OPTIMIZE TABLE tbl_name

通过 OPTIMIZE TABLE语句可以消除删除和更新造成的文件碎片OPTIMIZE TABLE语句在执行过程中会给表加上只读锁

- 修复表：REPAIR TABLE tbl_name

如何修改大表的表结构

对表中的列的字段类型进行修改

改变字段的宽度时还是会锁表

无法解决主从数据库延迟的问题

```
pt-online-schema-change
--alter="MODIFY c VARCHAR(50) NOT NULL DEFAULT ''"
--user=root --password=hefery D=数据库名,t=表名
--charset=utf8
--execute
```

设计MySQL高可用架构

读写分离

负载均衡

优化数据库索引

索引优化策略

- 索引列上不能使用表达式或函数

out_date为索引：select ... from product where to_days(out_date)-to_days(current_date)<=30 -> select..... from product where out_date<=date_add(current_date, interval 30 day)

- 索引列的选择：索引的选择性是不重复的索引值和表的记录数的比值
- 联合索引

如何选择索引列的顺序：经常会被使用到的列优先、选择性高的列优先、宽度小的列优先

索引使用原则

- 数据量很大，而且经常被查询的数据表可以设置索引
- 索引只添加在经常被用作检索条件的字段上面：姓名、编号……
- 尽量在数值类型字段建立索引，不要在大字段上创建索引：字符串
- 一张表的索引数最好不超过6个

索引优化查询

- 索引的列顺序和Order By子句的顺序完全一致
- 索引中所有列的方向（升序，降序）和Order by子句完全一致
- Order by中的字段全部在关联表中的第一张表中

在 WHERE 和 GROUP BY 涉及的字段建立索引

WHERE 条件避免使用 NULL，可使用 0/-1

WHERE 条件避免使用 != 或 <>，转化为 < <= = > >= BETWEEN-AND

WHERE 条件避免使用 OR 连接查询条件

WHERE 条件避免使用 [NOT] IN，使用EXISTS代替

select id from stu where id in(select id from dtu)

select id from stu where exists(select 1 from dtu where id=stu.id)

WHERE 条件避免使用 LIKE：

select id from stu where name like '%abc'　# 索引失效

select id from stu where name like 'abc%'　# 索引生效

WHERE 条件避免使用表达式或函数操作

WHERE 条件避免使用

优化SQL查询

获取有性能问题SQL

- 通过用户反馈获取存在性能问题的SQL
- 通过慢查日志获取存在性能问题的SQL

```
slow_query_log　　# MySQL配置文件设置启动停止记录慢查日志
slow_query_log_file　# 指定慢查日志的存储路径及文件
long_query_time　　# 指定记录慢查日志SQL执行时间的伐值，建议0.001S
log_queries_not_using_indexes　#　是否记录未使用索引的SQL

mysqldumpslow -s r -t 10 slow-mysql.log -s order (c | t | l | r | at | al | ar)
pt-query-digest --explain h=127.0.0.1 slow-mysql.log
```

- 实时获取存在性能问题的SQL

```
SELECT * FROM information_schema.`PROCESSLIST` WHERE time > 60
```

如何确定查询处理各个阶段所消耗的时间

```
set profiling = 1;
执行查询语句
show profiles;
show profile for query 1;
```

优化not in和<>查询

```
SELECT customer_id, first name, last_name, email FROM customer
WHERE customer_id NOT IN(SELECT customer_id FROM payment)

SELECT a.customer _id,a.first_name,a.last_name,a.email FROM customer a
LEFT JOIN payment b
ON a.customer_id=b.customer_id
WHERE b.customer_id IS NULL
```

优化MySQL性能参数

```
# 连接MySQL服务器次数
SHOW STATUS LIKE '%CONNECTION%'
# 服务器上线时间
SHOW STATUS LIKE '%Uptime%'
# 慢查询次数
SHOW STATUS LIKE 'Slow_queries'
# 查询查询操作次数
SHOW STATUS LIKE 'Com_select'
# 查询修改操作次数
SHOW STATUS LIKE 'Com_update'
# 查询删除操作次数
SHOW STATUS LIKE 'Com_delete'
```

优化MySQL服务器

硬件优化

- 配置较大的内存：内存的速度比磁盘I/O快得多，可以通过增加系统的缓冲区容量，使数据在内存留的时间更长，以减少磁盘I/O
- 配置高速磁盘系统，读盘的等待时间，提高响应速度
- 合理分布磁盘I/O：把磁盘I/O分散在多个设备上，减少资源竞争，提高并行操作能力
- 配置多处理器：MySQL是多线程数据库，多处理器可同时执行多个线程

参数优化

- 内存配置相关参数

确定可以使用的内存的上限

确定MySQL的每个连接使用的内存

确定需要为操作系统保留多少内存

如何为缓存池分配内存：

```
Innodb_buffer_pool_size：总内存-（每个线程所需要的内存*连接数）- 系统保留内存
key_buffer_size：
```

- IO相关配置参数

```
Innodb_log_file_size：
Innodb_log_files_in_group：
事务日志总大小 = Innodb_log_files_in_group * Innodb_log_file_size
Innodb_log_buffer_size
Innodb_flush_log_at_trx_commit：
      0：每秒进行一次log写入 cache，并flush log到磁盘
      1：默认，在每次事务提交执行log写入cache，并flush log到磁盘
      2：建议，每次事务提交执行log数据写入到cache，每秒执行一次flush log到磁盘
Innodb_flush_method = O_DIRECT
Innodb_file_per table = 1
Innodb_doublewrite = 1
delay_key_write
   OFF：每次写操作后刷新键缓冲中的脏块到磁盘
```

原文链接：MySQL 优化建议

ON：只对在键表时指定了 delay_ key_write选项的表使用延迟刷新
ALL：对所有 MYISAM表都使用延迟键写入

- 安全相关配置参数

expire_ logs_days：指定自动清理 binlog 的天数
max_allowed_packet：控制 MySQL可以接收的包的大小
skip_name_resolve：禁用DNS查找
sysdate_is_now：确保 sysdate() 返回确定性日期
read_only：禁止非 super权限的用户写权限
skip_slave_start：禁用 Slave 自动恢复
sql_mode：设置 MySQL 所使用的SQL模式
strict_trans_tables
no_engine_subtitution
no_zero_date
no_zero_in_date
only_full_group_by

- 其它常用配置参数

sync_binlog：控制MySQL如何向磁盘刷新 binlog
tmp_table_size 和 max_ heap_table_size：控制内存临时表大小
max_connections：控制允许的最大连接数
key_buffer_size：索引缓冲区大小，取决于内存大小，太大导致OS频繁换页，降低系统性能
table_cache：同时打开表的数量
query_cache_size：查询缓冲区大小
sort_buffer_size：排序缓冲区大小
read_buffer_size：每个线程连续扫描时为扫描的每个表分配的缓冲区大小
read_rnd_buffer_size：为每个线程保留的缓冲区大小，主要用于存储按特定顺序读取的记录
Innodb_buffer_pool_size：InnoDB类型的表和索引最大缓存
Innodb_flush_log_at_trx_commit：何时将缓冲区的数据写入数据文件，并将日志文件写入磁盘
max_connections：控制允许的最大连接数
back_log：MySQL暂停回答新请求之前短时间内，多少个请求可以被存入堆栈
interactive_timeout：MySQL在关闭一个交互的连接之前所要等待的秒数
thread_cache_size：当客户端断开之后，服务器处理此客户的线程将会缓存起来以响应下一个客户不是销毁
wait_timeout：MySQL关闭一个非交互的连接之前所要等待的秒数
tmp_table_size 和 max_ heap_table_size：控制内存临时表大小