史上更全的 MySQL 高性能优化实战总结!

一、前言

MySQL对于很多Linux从业者而言，是一个非常棘手的问题，多数情况都是因为对数据库出现问题的情况和处理思路不清晰。在进行MySQL的优化之前必须要了解的就是MySQL的查询过程，很多的查询优化工作实际上就是遵循一些原则让MySQL的优化器能够按照预想的合理方式运行而已。

今天给大家体验MySQL的优化实战!

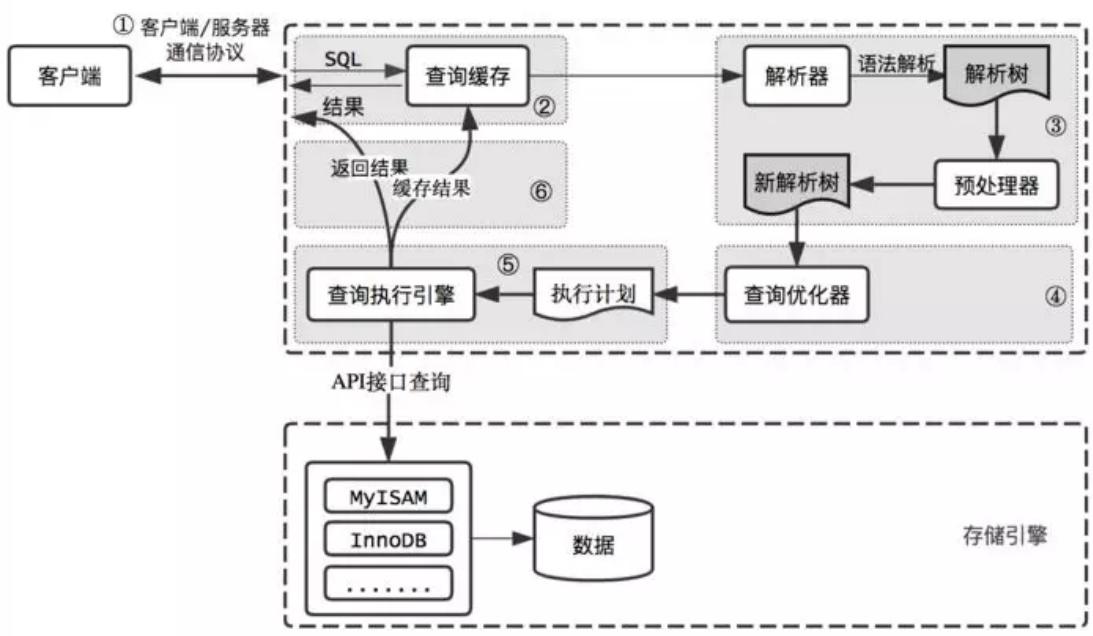

图 - MySQL查询过程

二、优化的哲学

注意：优化有风险，涉足需谨慎!

2.1、优化可能带来的问题

- 优化不总是对一个单纯的环境进行，还很可能是一个复杂的已投产的系统。
- 优化手段本来就有很大的风险，只不过你没能力意识到和预见到！
- 任何的技术可以解决一个问题，但必然存在带来一个问题的风险！
- 对于优化来说解决问题而带来的问题,控制在可接受的范围内才是有成果。
- 保持现状或出现更差的情况都是失败！

2.2、优化的需求

- 稳定性和业务可持续性,通常比性能更重要！
- 优化不可避免涉及到变更，变更就有风险！
- 优化使性能变好，维持和变差是等概率事件！
- 切记优化,应该是各部门协同，共同参与的工作，任何单一部门都不能对数据库进行优化！
- 所以优化工作,是由业务需要驱使的！！！

2.3、优化由谁参与

在进行数据库优化时，应由数据库管理员、业务部门代表、应用程序架构师、应用程序设计人员、应用程序开发人员、硬件及系统管理员、存储管理员等，业务相关人员共同参与。

三、优化思路

3.1、优化什么

在数据库优化上有两个主要方面：即安全与性能。

- 安全 ---> 数据可持续性
- 性能 ---> 数据的高性能访问

3.2、优化的范围有哪些

存储、主机和操作系统方面:

- 主机架构稳定性
- I/O规划及配置
- Swap交换分区
- OS内核参数和网络问题

应用程序方面:

- 应用程序稳定性
- SQL语句性能
- 串行访问资源
- 性能欠佳会话管理
- 这个应用适不适合用MySQL

数据库优化方面:

- 内存
- 数据库结构(物理&逻辑)
- 实例配置

说明：不管是在，设计系统，定位问题还是优化，都可以按照这个顺序执行。

3.3、优化维度

数据库优化维度有四个：

硬件、系统配置、数据库表结构、SQL及索引。

优化选择：

- 优化成本:硬件>系统配置>数据库表结构>SQL及索引
- 优化效果:硬件<系统配置<数据库表结构<SQL及索引

四、优化工具有啥?

4.1、数据库层面

检查问题常用工具：

```
msyqladmin        #mysql客户端，可进行管理操作
mysqlshow         #功能强大的查看shell命令
show [SESSION | GLOBAL] variables  #查看数据库参数信息
SHOW [SESSION | GLOBAL] STATUS     #查看数据库的状态信息
information_schema                 #获取元数据的方法s
SHOW ENGINE INNODB STATUS Innodb   #引擎的所有状态
SHOW PROCESSLIST    #查看当前所有连接session状态
explain             #获取查询语句的执行计划s
how index           #查看表的索引信息
slow-log            #记录慢查询语句
mysqldumpslow       #分析slowlog文件的
```

不常用但好用的工具：

```
zabbix           #监控主机、系统、数据库（部署zabbix监控平台）
pt-query-digest  #分析慢日志
mysqlslap        #分析慢日志
sysbench         #压力测试工具
mysql profiling              #统计数据库整体状态工具
Performance Schema mysql     #性能状态统计的数据
workbench      #管理、备份、监控、分析、优化工具（比较费资源）
```

4.2、数据库层面问题解决思路

一般应急调优的思路：

针对突然的业务办理卡顿，无法进行正常的业务处理！需要立马解决的场景！

- 1、show processlist
- 2、explain select id ,name from stu where name='clsn'; # ALL id name age sex

select id,name from stu where id=2-1 函数 结果集>30;
show index from table;

- 3、通过执行计划判断，索引问题（有没有、合不合理）或者语句本身问题
- 4、show status like '%lock%'; # 查询锁状态

kill SESSION_ID; # 杀掉有问题的session

常规调优思路：

针对业务周期性的卡顿，例如在每天10-11点业务特别慢，但是还能够使用，过了这段时间就好了。

- 1、查看slowlog，分析slowlog，分析出查询慢的语句。
- 2、按照一定优先级，进行一个一个的排查所有慢语句。
- 3、分析top sql，进行explain调试，查看语句执行时间。
- 4、调整索引或语句本身。

4.3、系统层面

cpu方面：

vmstat、sar top、htop、nmon、mpstat

内存：

free 、ps -aux 、

IO设备（磁盘、网络）：

iostat 、 ss 、 netstat 、 iptraf、iftop、lsof、

vmstat 命令说明：

- Procs：r显示有多少进程正在等待CPU时间。b显示处于不可中断的休眠的进程数量。在等待I/O

- Memory：swpd显示被交换到磁盘的数据块的数量。未被使用的数据块，用户缓冲数据块，用于操作系统的数据块的数量
- Swap：操作系统每秒从磁盘上交换到内存和从内存交换到磁盘的数据块的数量。s1和s0最好是0
- Io：每秒从设备中读入b1的写入到设备b0的数据块的数量。反映了磁盘I/O
- System：显示了每秒发生中断的数量(in)和上下文交换(cs)的数量
- Cpu：显示用于运行用户代码，系统代码，空闲，等待I/O的CPU时间

iostat命令说明

实例命令： iostat -dk 1 5

iostat -d -k -x 5 （查看设备使用率（%util）和响应时间（await））

- tps：该设备每秒的传输次数。“一次传输”意思是“一次I/O请求”。多个逻辑请求可能会被合并为“一次I/O请求”。
- iops ： 硬件出厂的时候，厂家定义的一个每秒最大的IO次数,"一次传输"请求的大小是未知的。
- kB_read/s：每秒从设备（drive expressed）读取的数据量；
- KB_wrtn/s：每秒向设备（drive expressed）写入的数据量；
- kB_read：读取的总数据量；
- kB_wrtn：写入的总数量数据量；这些单位都为Kilobytes。

4.4、系统层面问题解决办法

你认为到底负载高好，还是低好呢?

在实际的生产中，一般认为 cpu只要不超过90%都没什么问题 。

当然不排除下面这些特殊情况：

问题一：cpu负载高，IO负载低

- 内存不够
- 磁盘性能差
- SQL问题 ------>去数据库层，进一步排查sql问题
- IO出问题了（磁盘到临界了、raid设计不好、raid降级、锁、在单位时间内tps过高）
- tps过高: 大量的小数据IO、大量的全表扫描

问题二：IO负载高，cpu负载低

- 大量小的IO 写操作：
- autocommit ，产生大量小IO
- IO/PS,磁盘的一个定值，硬件出厂的时候，厂家定义的一个每秒最大的IO次数。
- 大量大的IO 写操作
- SQL问题的几率比较大

问题三：IO和cpu负载都很高

硬件不够了或sql存在问题

五、基础优化

5.1、优化思路

定位问题点:

硬件 --> 系统 --> 应用 --> 数据库 --> 架构（高可用、读写分离、分库分表）

处理方向：

明确优化目标、性能和安全的折中、防患未然

5.2、硬件优化

主机方面：

- 根据数据库类型，主机CPU选择、内存容量选择、磁盘选择
- 平衡内存和磁盘资源
- 随机的I/O和顺序的I/O
- 主机 RAID卡的BBU(Battery Backup Unit)关闭

cpu的选择：

- cpu的两个关键因素：核数、主频
- 根据不同的业务类型进行选择：
- cpu密集型：计算比较多，OLTP 主频很高的cpu、核数还要多
- IO密集型：查询比较，OLAP 核数要多，主频不一定高的

内存的选择：

- OLAP类型数据库，需要更多内存，和数据获取量级有关。
- OLTP类型数据一般内存是cpu核心数量的2倍到4倍，没有最佳实践。

存储方面：

- 根据存储数据种类的不同，选择不同的存储设备
- 配置合理的RAID级别(raid5、raid10、热备盘)
- 对与操作系统来讲，不需要太特殊的选择，最好做好冗余（raid1）（ssd、sas 、sata）

raid卡：主机raid卡选择：

- 实现操作系统磁盘的冗余（raid1）
- 平衡内存和磁盘资源
- 随机的I/O和顺序的I/O
- 主机 RAID卡的BBU(Battery Backup Unit)要关闭。

网络设备方面：

使用流量支持更高的网络设备（交换机、路由器、网线、网卡、HBA卡）

注意：以上这些规划应该在初始设计系统时就应该考虑好。

5.3、服务器硬件优化

- 1、物理状态灯：
- 2、自带管理设备：远程控制卡（FENCE设备：ipmi ilo idarc），开关机、硬件监控。
- 3、第三方的监控软件、设备（snmp、agent）对物理设施进行监控
- 4、存储设备：自带的监控平台。EMC2（hp收购了）， 日立（hds），IBM低端OEM hds，高端存储是自己技术，华为存储

5.4、系统优化

Cpu：

基本不需要调整，在硬件选择方面下功夫即可。

内存：

基本不需要调整，在硬件选择方面下功夫即可。

SWAP：

MySQL尽量避免使用swap。阿里云的服务器中默认swap为0

IO：

- raid、no lvm、 ext4或xfs、ssd、IO调度策略
- Swap调整(不使用swap分区)

```
/proc/sys/vm/swappiness的内容改成0（临时）
/etc/sysctl.conf上添加vm.swappiness=0（永久）
```

这个参数决定了Linux是倾向于使用swap，还是倾向于释放文件系统cache。在内存紧张的情况下，数值越低越倾向于释放文件系统cache。当然，这个参数只能减少使用swap的概率，并不能避免Linux使用swap。

修改MySQL的配置参数innodb_flush_method，开启O_DIRECT模式。这种情况下，InnoDB的buffer pool会直接绕过文件系统cache来访问磁盘，但是redo log依旧会使用文件系统cache。值得注意的是，Redo log是覆写模式的，即使使用了文件系统的cache，也不会占用太多

IO调度策略：

5.5、系统参数调整

Linux系统内核参数优化：

```
vim /etc/sysctl.conf
net.ipv4.ip_local_port_range = 1024 65535 # 用户端口范围
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_fin_timeout = 30
fs.file-max=65535 # 系统最大文件句柄，控制的是能打开文件最大数量
```

用户限制参数（mysql可以不设置以下配置）：

```
vim /etc/security/limits.conf
 * soft nproc 65535
 * hard nproc 65535
 * soft nofile 65535
 * hard nofile 65535
```

5.6、应用优化

业务应用和数据库应用独立,防火墙：iptables、selinux等其他无用服务(关闭)：

```
chkconfig --level 23456 acpid off
chkconfig --level 23456 anacron off
chkconfig --level 23456 autofs off
chkconfig --level 23456 avahi-daemon off
chkconfig --level 23456 bluetooth off
chkconfig --level 23456 cups off
chkconfig --level 23456 firstboot off
chkconfig --level 23456 haldaemon off
chkconfig --level 23456 hplip off
chkconfig --level 23456 ip6tables off
chkconfig --level 23456 iptables off
chkconfig --level 23456 isdn off
chkconfig --level 23456 pcscd off
chkconfig --level 23456 sendmail off
chkconfig --level 23456 yum-updatesd off
```

安装图形界面的服务器不要启动图形界面 runlevel 3,另外，思考将来我们的业务是否真的需要MySQL，还是使用其他种类的数据库。用数据库的最高境界就是不用数据库。

六、数据库优化

SQL优化方向：

执行计划、索引、SQL改写

架构优化方向：

高可用架构、高性能架构、分库分表

6.1、数据库参数优化

调整：

实例整体（高级优化，扩展）

```
thread_concurrency    #并发线程数量个数
sort_buffer_size      #排序缓存
read_buffer_size      #顺序读取缓存
read_rnd_buffer_size  #随机读取缓存
key_buffer_size       #索引缓存
thread_cache_size     #线程缓存(1G—>8, 2G—>16, 3G—>32, >3G—>64)
```

连接层（基础优化）

设置合理的连接客户和连接方式

```
max_connections         #最大连接数，看交易笔数设置
max_connect_errors      #最大错误连接数，能大则大
connect_timeout         #连接超时
max_user_connections    #最大用户连接数
skip-name-resolve       #跳过域名解析
wait_timeout            #等待超时
back_log                #可以在堆栈中的连接数量
```

SQL层（基础优化）

- query_cache_size： 查询缓存
- OLAP类型数据库,需要重点加大此内存缓存.
- 但是一般不会超过GB.
- 对于经常被修改的数据，缓存会立马失效。
- 我们可以实用内存数据库（redis、memecache），替代他的功能。

6.2、存储引擎层（innodb基础优化参数）

```
default-storage-engine
# 没有固定大小，50%测试值，看看情况再微调。
# 但是尽量设置不要超过物理内存70% innodb_file_per_table=(1,0)
innodb_buffer_pool_size
innodb_flush_log_at_trx_commit=(0,1,2) #1是最安全的，0是性能最高，2折中
binlog_sync
Innodb_flush_method=(O_DIRECT, fdatasync)
innodb_log_buffer_size      #100M以下
innodb_log_file_size        #100M 以下
innodb_log_files_in_group   #5个成员以下,一般2-3个够用（iblogfile0-N）
innodb_max_dirty_pages_pct  #达到百分之75的时候刷写 内存脏页到磁盘。 log_bin
max_binlog_cache_size       #可以不设置
max_binlog_size             #可以不设置
innodb_additional_mem_pool_size     #小于2G内存的机器，推荐值是20M。32G内存以上100M
```

7 参考文献

[1] https://www.cnblogs.com/zishengY/p/6892345.html

[2] https://www.jianshu.com/p/d7665192aaaf