35.MySQL-全面优化√

0.面试回答-mysql数据库做过哪些优化?

```
我们从几个层面进行回答。
之前公司上线了新业务，我站在整体业务角度，进行了全面优化方案的撰写。其中包含了
硬件方面选择了戴尔服务器R730,40核cpu,128G的内存 10台服务器 8块sas盘（两块做raid1
操作系统方面：采用centos7.6版本，关闭防火墙与selinux,关闭numa，关闭THP，选择xfs文

```

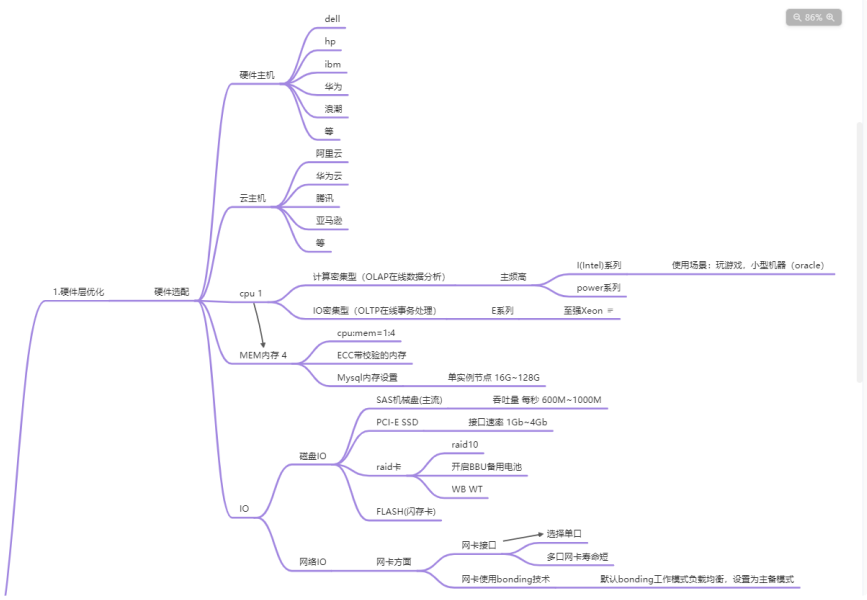
1.硬件层优化

1.1 标准化数据库专用服务器

> 帮助公司和运维团队,选择最合适MySQL数据库运行的服务器硬件,从品牌、CPU、MEM、IO设备、
> 网络设备、存储设备等各个层次进行合理建议.而不是上采购人员、商务人员或根本不懂数据库的人员制
> 定服务器标准。杜绝类似：内存小了、磁盘没法用、不符合最低3-5年扩展性硬件等此类问题出现。

1.2 标准化服务器硬件带来的收益

> 出现业务系统故障或性能问题。可以让拍错或者优化时间大大缩减。帮助管理员可以快速根据基准值
> 结合经验，定位瓶颈问题。



2. 操作系统及配置优化

2.1 标准化数据库操作系统

> 目前，互联网企业广泛应用centos系列操作系统。并且在同一组集群架构的服务器系统都保持系统

和内核版本一致。

2.2 标准化数据库稳定系统

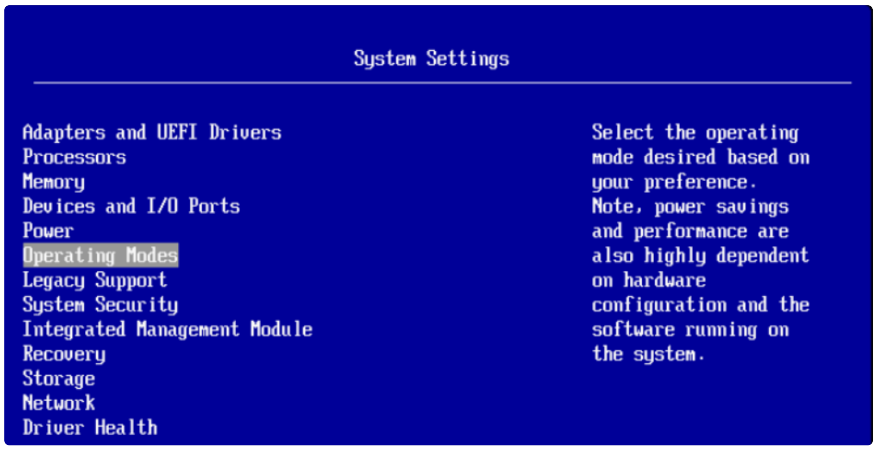
> 目前采用Centos7.2以上双数版。并且安装同版本光盘稳定兼容较好的软件包。

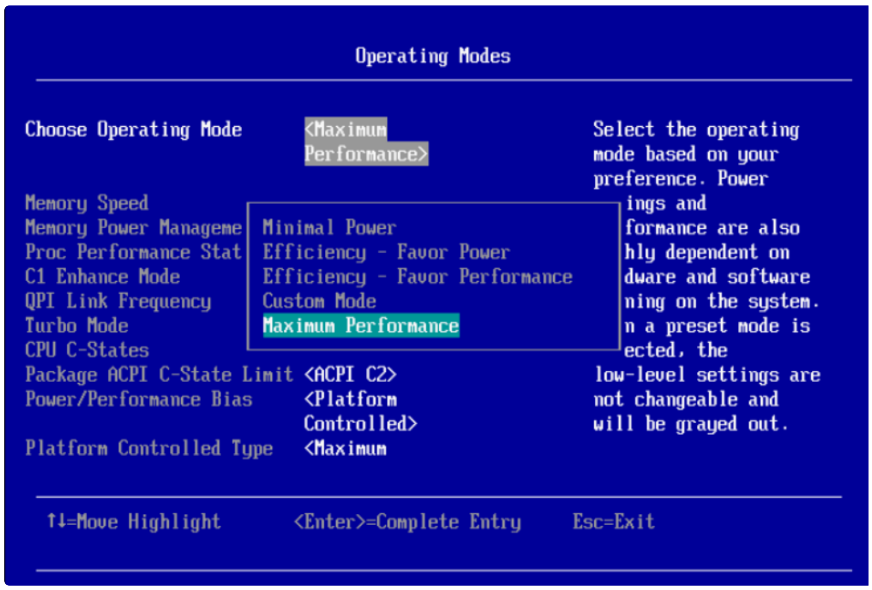
2.3. 标准化操作系统及硬件参数

2.3.1 关闭NUMA

> a. bios级别:
> 在bios层面numa关闭时，无论os层面的numa是否打开，都不会影响性能。
> # numactl --hardware
> available: 1 nodes (0) #如果是2或多个nodes就说明numa没关掉
> b. OS grub级别:
> vi /boot/grub2/grub.cfg
> #/* Copyright 2010, Oracle. All rights reserved. */
> default=0
> timeout=5
> hiddenmenu
> foreground=000000
> background=ffffff
> splashimage=(hd0,0)/boot/grub/oracle.xpm.gz
> title Trying_C0D0_as_HD0
> root (hd0,0)
> kernel /boot/vmlinuz-2.6.18-128.1.16.0.1.el5 root=LABEL=DBSYS ro
> bootarea=dbsys rhgb quiet console=ttyS0,115200n8 console=tty1
> crashkernel=128M@16M numa=off
> initrd /boot/initrd-2.6.18-128.1.16.0.1.el5.img
> 在os层numa关闭时,打开bios层的numa会影响性能，QPS会下降15-30%;
> c. 数据库级别:
> mysql> show variables like '%numa%';
> +------------------------+-------+
> | Variable_name | Value |
> +------------------------+-------+
> | innodb_numa_interleave | OFF |
> +------------------------+-------+
> 或者:
> vi /etc/init.d/mysqld
> 找到如下行
> # Give extra arguments to mysqld with the my.cnf file. This script
> # may be overwritten at next upgrade.
> $bindir/mysqld_safe --datadir="$datadir" --pid-file="$mysqld_pid_file_path"
> $other_args >/dev/null &
> wait_for_pid created "$!" "$mysqld_pid_file_path"; return_value=$?
> 将$bindir/mysqld_safe --datadir="$datadir"这一行修改为:
> /usr/bin/numactl --interleave all $bindir/mysqld_safe --datadir="$datadir"
> --pid-file="$mysqld_pid_file_path" $other_args >/dev/null &
> wait_for_pid created "$!" "$mysqld_pid_file_path"; return_value=$?

2.3.2开启CPU高性能模式





2.3.3阵列卡RAID配置

> raid10(推荐)
> SSD、PCI-E、Flash

2.3.4 关闭THP(和mongodb有关)

未关闭会导致内存泄漏swap的使用或内存的碎片化比较严重

> vi /etc/rc.local 在文件末尾添加如下指令：
> if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
> echo never > /sys/kernel/mm/transparent_hugepage/enabled
> fi
> if test -f /sys/kernel/mm/transparent_hugepage/defrag; then
> echo never > /sys/kernel/mm/transparent_hugepage/defrag
> fi
> [root@master ~]# cat /sys/kernel/mm/transparent_hugepage/enabled
> always madvise [never]
> [root@master ~]# cat /sys/kernel/mm/transparent_hugepage/defrag
> always madvise [never]

2.3.5 网卡绑定

> bonding技术，业务数据库服务器都要配置bonding继续。建议是主备模式。
> 交换机一定要堆叠。

2.3.6存储多路径

> 使用独立存储设备的话，需要配置多路径:
> linux 自带 multipath
> 厂商提供

2.3.7 系统层面参数优化

a. 更改文件句柄和进程数
内核优化 /etc/sysctl.conf
vm.swappiness = 5（也可以设置为0）　当物理内存剩余百分之几，开启swap分区，对于数据库尽量不适用swap.
vm.dirty_ratio = 20
vm.dirty_background_ratio = 10
net.ipv4.tcp_max_syn_backlog = 819200
net.core.netdev_max_backlog = 400000
net.core.somaxconn = 4096
net.ipv4.tcp_tw_reuse=1
net.ipv4.tcp_tw_recycle=0

b. 防火墙
禁用selinux： /etc/sysconfig/selinux 更改SELINUX=disabled.
iptables如果不使用可以关闭。可是需要打开MySQL需要的端口号

c. 文件系统优化
推荐使用XFS文件系统
MySQL数据分区独立，例如挂载点为: /data
mount参数 defaults, noatime, nodiratime, nobarrier 如/etc/fstab:
/dev/sdb /data xfs
defaults,noatime,nodiratime,nobarrier 1 2

d. 不使用LVM

e. io调度 （io调度将磁盘中的数据读取到内存中，再由cpu处理，如果io调度差，会使cpu等待时间过长 ）
SAS：　　io调度算法： deadline 最后期限
SSD&PCI-E： io调度算法： noop 电梯

centos 7 默认是deadline
cat /sys/block/sda/queue/scheduler
#临时修改为deadline(centos6)
echo deadline >/sys/block/sda/queue/scheduler
vi /boot/grub/grub.conf
更改到如下内容:
kernel /boot/vmlinuz-2.6.18-8.el5 ro root=LABEL=/ elevator=deadline rhgb
quiet

f. nofile 最大句柄数 (操作系统对磁盘进行操作动作时会分配一个文件句柄)
要设置的足够大。
设置方法
临时修改
[root@db01 ~]# ulimit -HSn 65535
到 **/etc/rc.local** 每次启动启用。
永久修改
终极解除 Linux 系统的最大进程数和最大文件打开数限制:
vim /etc/security/limits.conf
添加如下的行
* soft nproc 11000
* hard nproc 11000
* soft nofile 655350
* hard nofile 655350
查看数据库使用的文件句柄
1.查看数据库进程号 1451
ps -ef | grep mysqld
mysql　1451　1 2 22:53 ?　00:00:05 /usr/local/mysql/bin/mysqld --defaults-file=/data/3307/my.cnf

```
root      1532   1414  0 22:58 pts/0    00:00:00 grep --color=auto mysqld
```

2.切换目录查看

ll /proc/1451/fd

数据库端的参数是 open_files_limit=5000(默认)

3.MySQL软件及版本选择优化

Bash | Copy

```
1、稳定版：选择开源的社区版的稳定版GA版本。
2、选择mysql数据库GA版本发布后6个月-12个月的GA双数版本，大约在15-20个小版本左右。
3、要选择前后几个月没有大的BUG修复的版本，而不是大量修复BUG的集中版本。
4、要考虑开发人员开发程序使用的版本是否兼容你选的版本。
5、作为内部开发测试数据库环境，跑大概3-6个月的时间。
6、优先企业非核心业务采用新版本的数据库GA版本软件。
7、向DBA高手请教，或者在技术氛围好的群里和大家一起交流，使用真正的高手们用过的好用的G

最终建议：  8.0.20是一个不错的版本选择。向后可以选择双数版。
```

4.MySQL三层结构及主从复制参数优化

show variables like '%%' 和 show status like '%%'的关系

参数（variables）的设置是依据stauts（状态）设置的

4.1 连接层参数优化

4.1.1 连接层参数优化

Bash | Copy

```
♥max_connections=1000  (单节点)                    最大并发连接数上限
max_connect_errors=999999                          最大错误连接上限 数
♥wait_timeout=600s  (10分钟) (默认8个小时)          非交互式连接的等待时间
interactive_wait_timeout=3600                      交互式连接等待时间
net_read_timeout   = 120s
net_write_timeout = 120s
♥max_allowed_packet= 32M                           最大传输数据包大小

本地登陆数据库参数
如果不设置，则每次本地登陆数据库会先扫描mysql所有的文件，会导致本地登陆慢
 【mysql】
no-auto-rehash
```

4.1.2 连接参数有相对应的计数器

```
mysql> show status like '%connect%';
+---------------------------------------------------------+--------------
| Variable_name                                           | Value
+---------------------------------------------------------+--------------
| Aborted_connects                                        | 0
| Connection_errors_accept                                | 0
| Connection_errors_internal                              | 0
| Connection_errors_max_connections                       | 0
| Connection_errors_peer_address                          | 0
| Connection_errors_select                                | 0
| Connection_errors_tcpwrap                               | 0
| Connections                                             | 11
| Locked_connects                                         | 0
| Max_used_connections                                    | 1
| Max_used_connections_time                               | 2021-05-23 22
| Mysqlx_connection_accept_errors                         | 0
| Mysqlx_connection_errors                                | 0
| Mysqlx_connections_accepted                             | 0
| Mysqlx_connections_closed                               | 0
| Mysqlx_connections_rejected                             | 0
| Performance_schema_session_connect_attrs_longest_seen   | 114
| Performance_schema_session_connect_attrs_lost           | 0
| Ssl_client_connects                                     | 0
| Ssl_connect_renegotiates                                | 0
| Ssl_finished_connects                                   | 0
| Threads_connected                                       | 1
+---------------------------------------------------------+--------------

```

4.1.3 案例

```
案例1.409报错
案例2.连接无法及时释放，kill进程不能生效
背景环境 mysql5.6版本 架构 keepalived+lvs+双主
分析原因：keepalived心跳检测时间大于wait_timeout
案例3.max_connections连接也和文件句柄数 和数据库参数open_files_limit数有关
```

4.2 server层参数优化

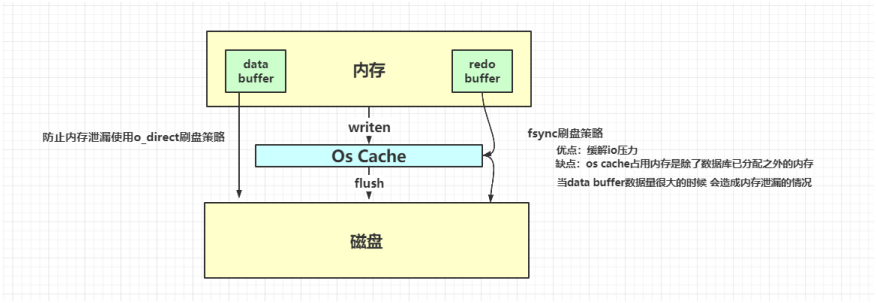
Bash | Copy

```
♥sql_safe_updates=1    安全更新模式（safe update mode），作用是做update  d

慢日志方面
slow_query_log=ON                          slow log的开关
♥slow_query_log_file=/data/3307/slow.log    文件位置及名字
♥long_query_time=0.01~0.1                   设定慢查询的时间限定
♥log_queries_not_using_indexes=ON            没走索引的语句也记录
♥log_throttle_queries_not_using_indexes = 10  重复性没走索引问题只记录的行数

会话级别的参数，每个连接会话都会独立分配，独立占用。
sort_buffer= 1M
join_buffer= 1M
read_buffer= 1M
read_rnd_buffer= 1M   随机读
tmp_table= 16M
heap_table= 16M

max_execution_time=28800               语句最大执行时间 ，设置太小可能会导致语
♥lock_wait_timeout= 60s （默认一年）     MDL(元数据)锁等待时间（发生DDL DCL语句
♥lower_case_table_names=1               自动把大写转化为小写
thread_cache_size=64                   线程缓存个数
♥log_timestamps=SYSTEM                  以操作系统的时间戳定义
♥init_connect="set names utf8"          登陆客户端会话默认字符集
event_scheduler=OFF                    关闭事件调度（类似linux中的定时任务）
♥secure-file-priv=/tmp                  导出数据库日志的安全路径
binlog日志相关（sql层日志）
♥binlog_expire_logs_seconds=2592000（30天）     自动过期时间
♥sync_binlog=1                           如果等于1 每次事务提交都会进
log-bin=/data/3307/mysql-bin             binlog文件名的前缀
log-bin-index=/data/3307/mysql-bin.index      binlog的索引文件
max_binlog_size=500M（默认1G）            binlog最大文件大小
binlog_format=ROW                        binlog记录格式
```

4.3 存储引擎层参数优化 🧠

Bash | Copy

```
♥transaction-isolation="READ-COMMITTED"    隔离级别RC。优点：避免next_lock

在事务被提交并写入到表空间磁盘文件上之前，事务数据存储在InnoDB的redo日志文件里
这些日志位于innodb_log_group_home_dir定义的目录中，通常我们把这个目录设置与inno
为了获得最佳性能，建议分离innodb_data_home_dir和innodb_log_group_home_dir到
innodb_data_home_dir=/xxx                  共享表空间ibdata目录
innodb_log_group_home_dir=/xxx             日志存放目录


redo log
innodb_log_file_size=2048M（默认5M）             redo log磁盘设置大小512M-4
innodb_log_files_in_group=3                       redo log组数设置2~4组

早期数据库单节点的 "双1" 概念
♥sync_binlog=1参数选项
0：关闭binlog立即同步到磁盘的功能，由操作系统决定binlog刷盘操作。当断电或者操作系
1：binlog立即刷新到磁盘
N：binlog组提交刷新磁盘，组提交等待中可能也有binlog数据的丢失
♥innodb_flush_log_at_trx_commit=2   参数选项
0：每秒gc刷新日志到os缓存，再gc刷新到磁盘
1：每次事务提交立即gc刷新日志到磁盘
2：每次事务提交gc刷新日志到缓存，再每秒将os缓存gc刷新到磁盘

刷新策略
主要负责日志（redo buffer）,数据（data buffer）刷盘
刷新策略的参数默认值是fsync
可选择项：
fsync
♥innodb_flush_method=O_DIRECT

iops（每秒的读写次数）
不同磁盘设置参数范围
sas单盘:200~500
sas raid10 500~1000
pci-e ssd 1000-1500~2000-3000
pci-e ssd raid10 2000-3000~5000-6000
flash 10000~20000
♥innodb_io_capacity=1000
innodb_io_capacity_max=4000
IBP内存
♥innodb_buffer_pool_size=64G    一般设置为物理内存的50%~75%

将IBP内存分成多少份（4~16）共同使用，减少latch征用
♥innodb_buffer_pool_instances=4

redo buffer
♥innodb_log_buffer_size=64M            是单个日志文件的0.5~1倍大小

脏页数量的百分比
当内存中的脏页百分比达到我们设置的值就会触发checkpoint,将脏页写入磁盘
如果百分比设置的过小，那么就意味着刷盘频率增大，io频率变多，io压力也就大了
♥innodb_max_dirty_pages_pct=85

行锁等待超时时间（默认50s）
♥innodb_lock_wait_timeout=10

文件句柄
♥innodb_open_files=63000

刷脏页线程的数量（默认值1个）
innodb_page_cleaners=4

innodb排序缓冲区，不常用，可以通过索引优化替代它
innodb_sort_buffer_size=64M

死锁相关
♥innodb_print_all_deadlocks=1   是否记录所有的死锁信息，会记录在日志中
innodb_rollback_on_timeout=ON
innodb_deadlock_detect=ON
```



4.4 主从复制参数优化

Bash | Copy

```
relay_log有关
relay_log=/opt/log/mysql/blog/relay
relay_log_index=/opt/log/mysql/blog/relay.index
max_relay_log_size=500M
relay_log_recovery=ON

增强半同步有关
rpl_semi_sync_master_enabled=ON
♥rpl_semi_sync_master_timeout=1000      等待ack信息的超时时间，如果超时转化为
rpl_semi_sync_master_trace_level=32
♥rpl_semi_sync_master_wait_for_slave_count=1   至少有一台从库放回ack信息，半
rpl_semi_sync_master_wait_no_slave=ON
♥rpl_semi_sync_master_wait_point=AFTER_SYNC   ack的等待点
rpl_semi_sync_slave_enabled=ON
rpl_semi_sync_slave_trace_level=32

组提交相关
binlog_group_commit_sync_delay=1 (s)
binlog_group_commit_sync_no_delay_count=1000(个)

gtid
gtid_mode=ON
enforce_gtid_consistency=ON

跳过自动启动主从，确认主从无误手动启动从库
skip-slave-start=1

从库只读模式设定，防止脑裂出现主从不一致
#read_only=ON
#super_read_only=ON

强制从库也记录binlog，并且同步GTID信息
log_slave_updates=ON

server_id=2330602
report_host=xxxx
```

5.MySQL开发规范

5.1 字段规范

```
1. 每个表建议在30个字段以内(了解三大范式)。
2. 需要存储emoji字符的，则选择utf8mb4字符集。
3. 机密数据，加密后存储。
4. 整型数据，默认加上UNSIGNED。
5. 存储IPV4地址建议用bigINT UNSIGNE，查询时再利用INET_ATON()、INET_NTOA()函数
6. 如果遇到BLOB、TEXT大字段单独存储表或者附件形式存储。
7. 选择尽可能小的数据类型，用于节省磁盘和内存空间。
8. 存储浮点数，可以放大倍数存储。
9. 每个表必须有主键，INT/BIGINT并且自增做为主键，分布式架构使用sequence序列生成器
10. 每个列使用not null，或增加默认值。
```

5.2 SQL语句规范

```
1. 去掉不必要的括号
如:       ((a AND b) AND c OR (((a AND b) AND (c AND d))))
修改成    (a AND b AND c) OR (a AND b AND c AND d)

2.  去掉重叠条件
如:       (a<b AND b=c) AND a=5
修改成    b>5 AND b=c AND a=5
如:       (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
修改成    B=5 OR B=6

3. 避免使用not in、not exists 、<>、like %%
4. 多表连接，小表驱动大表
5. 减少临时表应用，优化order by 、group by、union、distinct、join等
6. 减少语句查询范围，精确查询条件
7. 多条件，符合联合索引最左原则
8. 查询条件减少使用函数、拼接字符等条件、条件隐式转换
9. union all 替代 union(因为有隐式的排序)
10.减少having子句使用
11.如非必须不使用 for update语句
12.update和delete，开启安全更新参数
13.减少inset  ... select语句应用
14.使用load 替代insert录入大数据
15.导入大量数据时，可以禁用索引、增大缓冲区、增大redo文件和buffer、关闭autocommi
16.优化limit，最好业务逻辑中先获取主键ID，再基于ID进行查询
  limit 5000000,10     limit 10 , 200
17. DDL执行前要审核
18. 多表连接语句执行前要看执行计划
```

6.MySQL的索引优化

```
1. 非唯一索引按照“i_字段名称_字段名称[_字段名]”进行命名。
2. 唯一索引按照“u_字段名称_字段名称[_字段名]”进行命名。
3. 索引名称使用小写。
4. 索引中的字段数不超过5个。（联合索引的列条件不超过5个）
5. 唯一键由3个以下字段组成，并且字段都是整形时，使用唯一键作为主键。
6. 没有唯一键或者唯一键不符合5中的条件时，使用自增id作为主键。
7. 唯一键不和主键重复。
8. 索引选择度高的列作为联合索引最左条件
9. ORDER BY，GROUP BY，DISTINCT的字段需要添加在索引的后面。
10. 单张表的索引数量控制在5个以内，若单张表多个字段在查询需求上都要单独用到索引，需要
    查询性能问题无法解决的，应从产品设计上进行重构。

11. 使用EXPLAIN判断SQL语句是否合理使用索引，尽量避免extra列出现：Using File So

12. UPDATE、DELETE语句需要根据WHERE条件添加索引。

13. 对长度大于50的VARCHAR字段建立索引时，按需求恰当的使用前缀索引，或使用其他方法。

14. 下面的表增加一列url_crc32，然后对url_crc32建立索引，减少索引字段的长度，提高效

CREATE TABLE all_url(ID INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMEN
url VARCHAR(255) NOT NULL DEFAULT 0,
url_crc32 INT UNSIGNED NOT NULL DEFAULT 0,
index idx_url(url_crc32));

15. 合理创建联合索引（避免冗余），(a,b,c) 相当于 (a) 、(a,b) 、(a,b,c)。

16. 合理利用覆盖索引，减少回表。

17. 减少冗余索引和使用率较低的索引（查看方法如下）
mysql> select * from sys.schema_unused_indexes;
mysql> select * from sys.schema_redundant_indexes\G
```

7.MySQL的事务及锁优化 🧠

7.1 MDL锁-全局（读）锁 Global Read lock

简介

```
加锁方法：  FTWRL, flush tables with read lock.
解锁方法：  unlock tables;

出现的场景
1.MDP备份 --master-data
2.xtrabackup（pxb）
版本不同主要是加锁方式不同
mysql5.6 5.7版本 ---> xbk2.4  出现全局读锁
mysql 8.0 版本 ---> xbk8.0

加锁期间的影响：阻塞所有事务的写入，阻塞没有提交的事务不能再commit.
阻塞时间：  lock_wait_timeout=1年
```

检测方法

```bash
## 8.0之前需要手工配置开启.
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME = 'wait/lock/metadata/sql/mdl';

1.查看MDL锁情况
mysql> select * from performance_schema.metadata_locks\G
查看MDL锁 详细情况
select OBJECT_SCHEMA ,OBJECT_NAME ,LOCK_TYPE,LOCK_DURATION,LOCK_STATUS
2.查看有没有MDL锁阻塞情况
mysql> show processlist; 或者使用 select * from information_schema.proce
3.查看锁等待
mysql> select * from sys.schema_table_lock_waits;
4.进行沟通，分析 杀死进程，解除锁等待问题
```

经典案例

Bash | Copy

```
案例一：
mysql5.7 xtrabackup/mysqldump备份时数据库出现夯住的状态，所有修改查询都不能

一.模拟操作
session1: 模拟一个大的查询或事务
mysql> select id,sleep(100)  from city where id<100  for update ;


session2: 模拟备份时的FTWRL
mysql> flush tables with read lock;
-- 此时发现命令被阻塞

session3: 发起查询，发现被阻塞
mysql> select * from world.city where id=1 for update;

二.查看与分析

1.查看连接线程的状态,查看有没有MDL锁情况
mysql> show processlist;
+----+-----------------+-----------------+--------+---------+-----
| Id | User            | Host            | db     | Command | Time
+----+-----------------+-----------------+--------+---------+-----
|  5 | event_scheduler | localhost       | NULL   | Daemon  | 2443
|  6 | system user     | connecting host | NULL   | Connect | 2443
|  7 | system user     |                 | NULL   | Query   | 2443
| 17 | root            | localhost       | world  | Query   |    3
| 18 | root            | localhost       | world  | Query   |    3
| 19 | root            | localhost       | world  | Query   |    2
| 20 | root            | localhost       | taobao | Query   |    1
| 21 | root            | localhost       | NULL   | Query   |
+----+-----------------+-----------------+--------+---------+-----

2.查看MDL锁情况，获取被阻塞者，和阻塞者的sql线程ID
2.1查询语句：
use performance_schema
select * from metadata_locks\G ;
关注的地方就是PENDING（被阻塞的） GRANTED（获得锁者，阻塞者）的OWNER_THREA
2.2开始分析
(将pending和granted的线程id取出分析)
PENDING的sql线程id：
 OBJECT_TYPE: GLOBAL                       --->61
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
          COLUMN_NAME: NULL
OBJECT_INSTANCE_BEGIN: 120449904
            LOCK_TYPE: INTENTION_EXCLUSIVE
        LOCK_DURATION: STATEMENT
          LOCK_STATUS: PENDING
               SOURCE: sql_base.cc:3006
      OWNER_THREAD_ID: 61
       OWNER_EVENT_ID: 21

 OBJECT_TYPE: GLOBAL                        --->59
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
          COLUMN_NAME: NULL
OBJECT_INSTANCE_BEGIN: 140385167227072
            LOCK_TYPE: SHARED
        LOCK_DURATION: EXPLICIT
          LOCK_STATUS: PENDING
               SOURCE: lock.cc:1033
      OWNER_THREAD_ID: 59
       OWNER_EVENT_ID: 34

 OBJECT_TYPE: GLOBAL                         --->60
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
          COLUMN_NAME: NULL
OBJECT_INSTANCE_BEGIN: 140385680678224
            LOCK_TYPE: INTENTION_EXCLUSIVE
        LOCK_DURATION: STATEMENT
          LOCK_STATUS: PENDING
               SOURCE: sql_base.cc:3006
      OWNER_THREAD_ID: 60
```

```
        OWNER_EVENT_ID: 12

GRANTED的sql线程id:                          --->58
 OBJECT_TYPE: TABLE
       OBJECT_SCHEMA: world
         OBJECT_NAME: city
         COLUMN_NAME: NULL
OBJECT_INSTANCE_BEGIN: 140384899894000
            LOCK_TYPE: SHARED_WRITE
        LOCK_DURATION: TRANSACTION
         LOCK_STATUS: GRANTED
              SOURCE: sql_parse.cc:6057
      OWNER_THREAD_ID: 58
       OWNER_EVENT_ID: 15

3.获取阻塞者，被阻塞者的详细语句进行分析（通过sql线程id的方式）
3.1查询语句
use performance_schema
events_statements_history  用来查看历史操作过的sql语句信息
♥events_statements_current  我们分析锁，应该是查看当前执行的sql语句信息
select * from  events_statements_current  where THREAD_ID=线程id\G;

3.2开始分析
PENDING的sql线程id对应的具体sql语句:
THREAD_ID:59--->  SQL_TEXT: flush tables with read lock
THREAD_ID:60--->  SQL_TEXT: select * from city where id=500 for up
THREAD_ID:61--->  SQL_TEXT: select * from a where id=1 for update

granted的sql线程id对应的具体sql语句:
THREAD_ID:58--->  SQL_TEXT: select id,sleep(100)  from city where

4.分析阻塞者与被阻塞者之间的关系
首先我们知道58是阻塞者，所以是阻塞的源头，执行的语句是select id,sleep(100)
通过每个被阻塞者的具体语句与源头语句进行对比分析
THREAD_ID:61--->  SQL_TEXT: select * from a where id=1 for update
THREAD_ID:60--->  SQL_TEXT: select * from city where id=500 for up
所以推断出结果
源头58 阻塞了59 flush tables with read lock，59阻塞了60和61

5.解决问题 分析阻塞者执行sql的线程id对应的连接线程processlist_id的执行用户
5.1查询语句
use performance_schema
mysql> select * from threads where THREAD_ID=58 \G;
*************************** 1. row ***************************
          THREAD_ID: 58
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 17
   PROCESSLIST_USER: root
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: world
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 4274
  PROCESSLIST_STATE: User sleep
   PROCESSLIST_INFO: select id,sleep(100)  from city where id<100
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
            HISTORY: YES
    CONNECTION_TYPE: Socket
       THREAD_OS_ID: 1509
     RESOURCE_GROUP: USR_default
5.2 找到对应的执行用户，与人进行沟通，了解情况，判断是否能够终止操作，释放锁。
为了保证正常语句可以执行，必须选择舍弃，在这里大事务停止回滚也会造成锁问题，可能

5.3 终止备份操作
mysql> select * from threads where THREAD_ID=59 \G;
*************************** 1. row ***************************
          THREAD_ID: 59
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 18
   PROCESSLIST_USER: root
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: world
PROCESSLIST_COMMAND: Query
```

```
     PROCESSLIST_TIME: 4540
    PROCESSLIST_STATE: Waiting for global read lock
     PROCESSLIST_INFO: flush tables with read lock
     PARENT_THREAD_ID: NULL
                 ROLE: NULL
        INSTRUMENTED: YES
             HISTORY: YES
      CONNECTION_TYPE: Socket
         THREAD_OS_ID: 4030
       RESOURCE_GROUP: USR_default

5.4 将对应的processlist_id kill掉
kill 18

6.结论：我们备份操作要选择在业务低谷时期完成。
```

7.2.1 分析锁等待方法

Bash | Copy

```
第一步 查询锁等待详细信息
select * from sys.innodb_lock_waits;   ----> blocking_pid(锁源的连接线程)

第二步 通过连接线程找SQL线程
select * from performance_schema.threads;

第三步 通过SQL线程找到 SQL语句
select * from performance_schema.events_statements_history;

第四步 分析优化SQL语句
```

7.2.3 锁等待产生的原因和影响

Bash | Copy

```
原因：
record lock 、gap、next lock
都是基于索引加锁,与事务隔离级别有关。

影响：
导致操作系统cpu压力比较大 （压力过大原因：1.索引查询比较慢，低效率 2.大事务  3.锁问
```

7.2.4 锁等待的优化方向

Bash | Copy

```
1. 优化索引
2. 减少事务的更新范围
3. RC
4. 拆分语句：
例如：   update t1 set num=num+10 where k1 <100;  k1 是辅助索引,record lock
     改为：
     select id from t1 where  k1 <100; ---> id: 20,30,50
     update t1 set num=num+10   where id in (20,30,50);
```

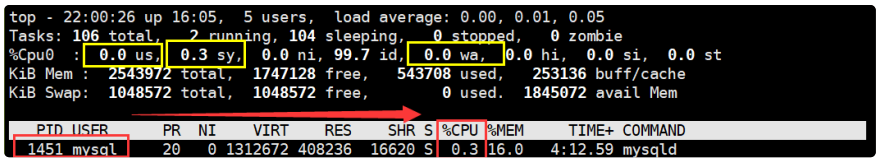
7.2.5 案例–mysql服务器cpu爆满–从操作系统层面追溯到数据库层面排查思路

案例背景：数据库服务器硬件配置 16核（c）cpu,通过top方式观察cpu爆满（使用总量达到1200%~1300%，平均时间比例达到80%）

排查思路：

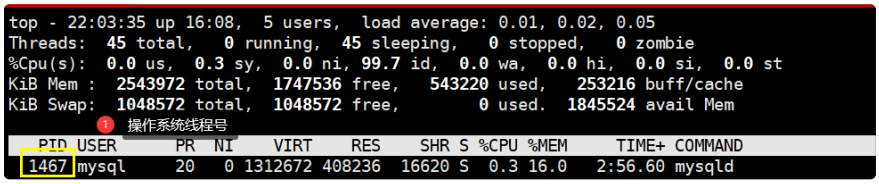
从操作系统层面追溯到数据库层面

1.在操作系统层面通过top命令查看到 mysql进程的cpu爆满



2.查看mysql进行的线程（因为mysql是线程工作）

通过top -Hp 1451(数据库进程号)，查看哪些mysql线程占用比例高



3.通过操作系统中mysql线程占用比例最高的操作系统线程号（os_id），找到对应数据库线程号

mysql在操作系统层面的线程和数据库中的线程号是映射关系，假设我们要找的os_id是1467。

Bash | Copy

```
mysql> use performance_shcema
mysql> select * from threads where THREAD_OS_ID=1467\G;
*************************** 1. row ***************************
          THREAD_ID: 15
               NAME: thread/innodb/log_checkpointer_thread
               TYPE: BACKGROUND
     PROCESSLIST_ID: NULL
   PROCESSLIST_USER: NULL
   PROCESSLIST_HOST: NULL
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
   PROCESSLIST_TIME: NULL
  PROCESSLIST_STATE: NULL
   PROCESSLIST_INFO: NULL
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
            HISTORY: YES
    CONNECTION_TYPE: NULL
       THREAD_OS_ID: 1467
     RESOURCE_GROUP: SYS_default
```

4.通过对应的数据库线程所对应的线程名，进行不同问题的分析处理

4.1 IO问题（考虑缓存方面的问题）

1.批量更改数据

2.操作大事务

3.内存小

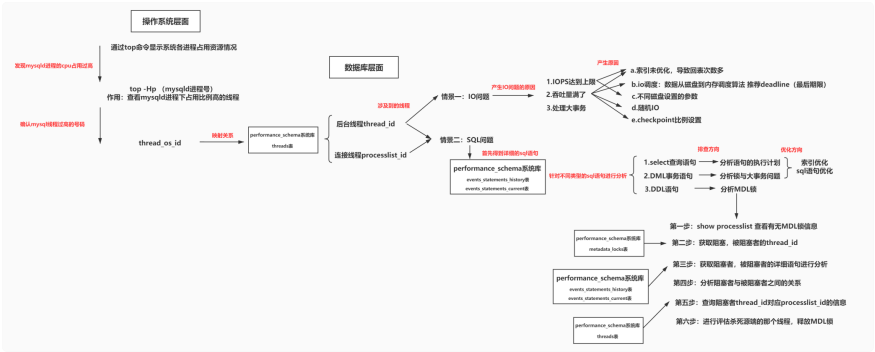
4..2 SQL问题

我们首先要知道sql语句是什么

通过数据库线程id（thread_id）找对相对应的sql语句

Bash | Copy

```
使用到的是系统库performance_schema下的两张表
events_statements_history  查看历史操作过的sql语句信息（cpu负载有所降低）
events_statements_current 查看当前执行的sql语句信息  （cpu负载还是很高情况下）

情况一：select查询语句--->索引方面问题，可以查看索引的执行计划
情况二：事务型语句---> 行锁方面的问题
情况三：DDL语句---> MDL锁问题
```



8.MySQL架构优化

Bash | Copy

```
高可用架构的选择
第一套：MHA+GTID+proxySQL+增强半同步+(延时从库)+（binlog_server） 999安全级别
第二套：MGR\innodb cluster 极其追求数据一致性，完整性
第三套：PXC  极其追求数据一致性，完整性

读写分离中间件选择
proxySQL 使用第一套和第三套高可用架构
Mysql-router 是第二套架构自动的读写分离功能


MySQL
高可用架构（+读写分离）：
分布式架构：
NoSQL-Redis
高可用架构（+读写分离）：Redis+Sentinel（哨兵）
分布式架构：Redis+ Cluster
NoSQL-MongoDB
高可用架构（+读写分离）：MongoDB RS
分布式架构：MongoDB SHARDING Cluster

ES 使用搜索类场景
```

9.MySQL的安全优化

Bash | Copy

```
1、 使用普通nologin用户管理MySQL
2、 合理授权用户、密码复杂度及最小权限、系统表保证只有管理员用户可访问。
3、 删除数据库匿名用户
4、 锁定非活动用户
5、 MySQL尽量不暴露互联网,需要暴露互联网用户需要设置明确白名单、替换MySQL默认端口号
6、 优化业务代码，防止SQL注入。
```

mysql%E6%95%B0%E6%8D%AE%E5%BA%93%E5%81%9A%E8%BF%87%E5%93%AA%E4%BA%9B%E4%BC%98%E5%8C%96%EF%BC%9F