源：<https://yq.aliyun.com/articles/722484?spm=a2c4e.11155435.0.0.1fcc2719mu67Vw>

**连接**

* **如何不需要密码的登录，直接mysql**

my.cnf 或者写在 ~/.my.cnf， 相对安全

[mysql] --表示： 只有mysql命令才能免密码

user=root

password=123

socket=/tmp/mysql.sock

[mysqladmin] --表示： 只有mysqladmin命令才能免密码

user=root

password=123

socket=/tmp/mysql.sock

[mysqldump] --表示： 只有mysqldump命令才能免密码

user=root

password=123

socket=/tmp/mysql.sock

[client] --表示：只要是客户端的命令，都是可以免密码的

user=root

password=123

socket=/tmp/mysql.sock

**MySQL如何查看用户名密码**

* MySQL5.7.6之前

1. show grants for $user;

2. select host,user,Password from user;

* MySQL5.7.6+

1. select host,user,authentication\_string,password\_lifetime,password\_expired,password\_last\_changed from mysql.user where user='lc\_rx';

**information\_schema相关**

**如何在线kill掉满足某种条件的session**

DB\_SYS: perl /home/Keithlan/scripts/outage/kill\_connection/kill\_sleepconn\_by\_opt.pl -opt $opt

**PROCESSLIST**

* **分析出当前连接过来的客户端ip的分布情况**

select substring\_index(host,':', 1) as appip ,count(\*) as count from information\_schema.PROCESSLIST group by appip order by count desc ;

* **分析处于Sleep状态的连接分布情况**

select substring\_index(host,':', 1) as appip ,count(\*) as count from information\_schema.PROCESSLIST where COMMAND='Sleep' group by appip order by count desc ;

* **分析哪些DB访问的比较多**

select DB ,count(\*) as count from information\_schema.PROCESSLIST where COMMAND='Sleep' group by DB order by count desc ;

* **分析哪些用户访问的比较多**

select user ,count(\*) as count from information\_schema.PROCESSLIST where COMMAND='Sleep' group by user order by count desc ;

**TABLES**

* **列出大于10G以上的表**

select TABLE\_SCHEMA,TABLE\_NAME,TABLE\_ROWS,ROUND((INDEX\_LENGTH+DATA\_FREE+DATA\_LENGTH)/1024/1024/1024) as size\_G from information\_schema.tables where ROUND((INDEX\_LENGTH+DATA\_FREE+DATA\_LENGTH)/1024/1024/1024) > 10 order by size\_G desc ;

**performance\_schema相关**

**performance\_schema占用多少内存**

[http://dev.mysql.com/doc/refman/5.7/en/show-engine.html](https://yq.aliyun.com/go/articleRenderRedirect?url=http%3A%2F%2Fdev.mysql.com%2Fdoc%2Frefman%2F5.7%2Fen%2Fshow-engine.html)  
SHOW ENGINE PERFORMANCE\_SCHEMA STATUS;  
For the Performance Schema as a whole, performance\_schema.memory is the sum of all the memory used (the sum of all other memory values).

**performance\_schema 瓶颈**

1) SHOW VARIABLES LIKE 'perf%';

2) SHOW STATUS LIKE 'perf%';

3) SHOW ENGINE PERFORMANCE\_SCHEMA STATUS\G

详细细节：http://keithlan.github.io/2015/07/17/22\_performance\_schema/

**如何查看每个threads当前session变量的值**

select \* from performance\_schema.variables\_by\_thread as a,(select THREAD\_ID,PROCESSLIST\_ID,PROCESSLIST\_USER,PROCESSLIST\_HOST,PROCESSLIST\_COMMAND,PROCESSLIST\_STATE from performance\_schema.threads where PROCESSLIST\_USER<>'NULL') as b where a.THREAD\_ID = b.THREAD\_ID and a.VARIABLE\_NAME = 'sql\_safe\_updates'

**TOP SQL 相关**

能够解决什么问题： 可以找到某个表是否还有业务访问？  
能够解决什么问题： 可以确定某个库，某个表的业务是否迁移干净？  
能够解决什么问题： 可以用于分析业务是否异常？  
能够解决什么问题： 根据TopN 可以分析压力？  
能够解决什么问题： 可以用于分析哪些表是热点数据，这些TopN的表才是值得优化的表。只要每一条语句快0.01ms，那么1亿条呢？

**实例中： 求SQL**

* **一个实例中查询最多的TopN SQL**

select SCHEMA\_NAME,DIGEST\_TEXT,COUNT\_STAR,FIRST\_SEEN,LAST\_SEEN from performance\_schema.events\_statements\_summary\_by\_digest where DIGEST\_TEXT like 'select%' and DIGEST\_TEXT not like '%SESSION%' order by COUNT\_STAR desc limit 10\G

* **一个实例中写入最多的TopN SQL**

select SCHEMA\_NAME,DIGEST\_TEXT,COUNT\_STAR,FIRST\_SEEN,LAST\_SEEN from performance\_schema.events\_statements\_summary\_by\_digest where DIGEST\_TEXT like 'insert%' or DIGEST\_TEXT like 'update%'or DIGEST\_TEXT like 'delete%' or DIGEST\_TEXT like 'replace%' order by COUNT\_STAR desc limit 10\G

**库中： 求SQL**

* **一个库中查询最多的TopN SQL**

同上 实例中： 求SQL

* **一个库中写入最多的TopN SQL**

同上 实例中： 求SQL

**实例中：求table**

* **使用说明**

usage: perl xx.pl -i 192.168.1.10 -p 3306 -e read|write|all 2>/dev/null ;

opt e:

read get select count

write get insert,update,delete count

all get all sql count

opt i:

192.xx.xx.xx ip address

opt p:

3306 db port

* **查看一个实例中，哪个表的SQL语句 访问最多?**

DB\_SYS: perl get\_table\_from\_sql.pl -i $ip -p $port -e all 2> /dev/null

* **查看一个实例中，哪个表的SQL语句 select【读】最多？**

DB\_SYS: perl get\_table\_from\_sql.pl -i $ip -p $port -e read 2> /dev/null

* **查看一个实例中，哪个表的SQL语句 insert+update+delete+replace【写】最多？**

DB\_SYS: perl get\_table\_from\_sql.pl -i $ip -p $port -e write 2> /dev/null

**Table IO 相关的监控**

**库级别**

* **如何查看一个MySQL实例中哪个库的all latency时间最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(SUM\_TIMER\_READ) as all\_read\_time,sum(SUM\_TIMER\_WRITE) as all\_write\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_time desc;

* **如何查看一个MySQL实例中哪个库的read latency时间最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(SUM\_TIMER\_READ) as all\_read\_time,sum(SUM\_TIMER\_WRITE) as all\_write\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_read\_time desc;

* **如何查看一个MySQL实例中哪个库的write latency时间最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(SUM\_TIMER\_READ) as all\_read\_time,sum(SUM\_TIMER\_WRITE) as all\_write\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_write\_time desc;

* **如何查看一个MySQL实例中哪个库的总访问量最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_star desc;

* **如何查看一个MySQL实例中哪个库的查询量(除了select中的fetchs外,还包括update，delete过程中的fetchs)最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_read desc;

* **如何查看一个MySQL实例中哪个库的写入量最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_write desc;

* **如何查看一个MySQL实例中哪个库的update量最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_update desc;

* **如何查看一个MySQL实例中哪个库的insert量最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_insert desc;

* **如何查看一个MySQL实例中哪个库的delete量最大**

select OBJECT\_SCHEMA,sum(SUM\_TIMER\_WAIT) as all\_time,sum(COUNT\_STAR) as all\_star,sum(COUNT\_read) as all\_read ,sum(COUNT\_WRITE) as all\_write,sum(COUNT\_FETCH) as all\_fetch,sum(COUNT\_INSERT) as all\_insert,sum(COUNT\_UPDATE) as all\_update,sum(COUNT\_DELETE) as all\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table group by OBJECT\_SCHEMA order by all\_delete desc;

**表级别**

* **表的all latency时间(read + write)最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,SUM\_TIMER\_READ,SUM\_TIMER\_WRITE,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by SUM\_TIMER\_WAIT desc limit 10

* **表的read latency(fetch)时间最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,SUM\_TIMER\_READ,SUM\_TIMER\_WRITE,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by SUM\_TIMER\_READ desc limit 10

* **表的write latency 时间最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,SUM\_TIMER\_READ,SUM\_TIMER\_WRITE,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by SUM\_TIMER\_WRITE desc limit 10

* **表的rows 总访问量最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by COUNT\_STAR desc limit 10

* **表的rows 查询量最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by COUNT\_read desc limit 10

* **表的rows 写入量最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by COUNT\_WRITE desc limit 10

* **表的rows update量最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by COUNT\_update desc limit 10

* **表的rows insert量最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by COUNT\_insert desc limit 10

* **表的rows delete量最大**

select OBJECT\_SCHEMA,OBJECT\_NAME,SUM\_TIMER\_WAIT,COUNT\_STAR,COUNT\_read,COUNT\_WRITE,COUNT\_UPDATE,COUNT\_insert,COUNT\_delete from performance\_schema.table\_io\_waits\_summary\_by\_table order by COUNT\_delete desc limit 10

**抓包**

\* tshark 中的 -e参数有哪些内容请参考

http://www.wireshark.org/docs/dfref/

https://www.wireshark.org/docs/dfref/m/mysql.html

https://www.wireshark.org/docs/dfref/t/tcp.html

https://www.wireshark.org/docs/dfref/m/memcache.html

https://www.wireshark.org/docs/dfref/h/http.html

\* tshark: 抓取mysql tcp包，以及大小

tshark -i any -R 'tcp.port == 3306 && mysql' -T fields -e tcp.port -e ip.addr -e mysql.query -e mysql.packet\_length -e tcp.len

tshark 高级版本将 -R 替换成了 -Y

\* tshark: 抓mysql包

tshark -i any dst host ${ip} and dst port 3306 -l -d tcp.port==3306,mysql -T fields -e frame.time -e 'ip.src' -e 'mysql.query' > yy.tshark --这种方式，会在/tmp/目录下创建很多临时文件，要小心，会产生磁盘报警。

\* tshark -i any dst host ${ip} and dst port 3306 -l -d tcp.port==3306,mysql -T fields -e 'ip.src' -e 'tcp.srcport' -e 'mysql.schema' -e 'mysql.query' -w yy.tshark --类似于tcpdump。

nohup tshark -i any dst host ${ip} and dst port 3306 -l -d tcp.port==3306,mysql -a duration:20 -T fields -e mysql.schema -e frame.time -e ip.src -e tcp.srcport -e mysql.query -w xx.sql & -- -a duration 当时间超过 20秒时，停止抓取。

nohup tshark -i any dst host ${ip} and dst port 3306 -l -d tcp.port==3306,mysql -a filesize:2000000 -T fields -e mysql.schema -e frame.time -e ip.src -e tcp.srcport -e mysql.query -w xx.sql & 注：当文件超过2G时，停止抓取。单位是Kilobyte。

\* 只抓取MySQL的包,不会有空格之类的了

tshark -i any dst host ${ip} and dst port 3306 -l -a duration:10 -R 'mysql.query' -T fields -e 'ip.src' -e 'mysql.query'

==from gitlab http://gitlab.corp.anjuke.com/\_incubator/knowledge/blob/master/tshark.md

\* thark：解tcpdump包

tshark -r xx.tcpdump -d tcp.port==3306,mysql -T fields -e mysql.schema -e frame.time -e ip.src -e mysql.query > test.tshark

\* 案例一、 memcache

# 需要使用 -d 让 tshark 认为 11213 是使用的 memcache 协议，否则 tshark 默认是将 11211 认为是 memcache 协议

~ tshark -i eth0 -d tcp.port==11213,memcache -R 'tcp.dstport == 11213 && memcache'

\* 案例二、 mysql

~ tshark -i eth0 -R 'tcp.port == 3306 && mysql.query' -T fields -e frame.time -e 'ip.src' -e 'mysql.query'

\* 案例三、http

~ tshark -i eth0 -R 'tcp.port == 80 && http'

# 这个命令非常有用，当我们的程序非常慢，但是有没有打印任何日志时，我们怀疑可能是某个 http 请求慢了，可以用这个命令检查

# http.time 表示整个 http请求 消耗的时间

# http.response.code 200、403、500 等

# tcp.analysis.initial\_rtt tcp 三次握手时间

# tcp.stream tshark 针对每一个5元组，都有一个编号，根据这个编号，可以方便的查到整个会话过程的所有请求 src.ip,src.port,tcp,dst.ip,dst.port，例如在这里，可以根据这个编号，找到请求所对应的 http.response.code ,因为在并发很高的时候，2个记录不一定紧挨着

~ tshark -i eth0 -R 'http && tcp.port == 80' -T fields -e tcp.analysis.initial\_rtt -e frame.time -e ip.addr -e tcp.port -e http.request.uri -e tcp.stream -e http.response.code -e http.time

\* 案例四、tcp

# 检查是否有tcp 包重传

~ tshark -i eth0 -R 'tcp.analysis.retransmission'

**slow query优化--切忌：不要在master进行分析和调优，在没有业务的机器上或者etl上分析诊断**

1) 先搞清楚时间到底花在哪里&&为什么时间会花在那 （show profile）

1.1 ) 主要工具和方法就是profiling

1.2 ) 整个性能优化，应该花90%的时间在测量上面，只有这样才能够对症下药

1.3 ) 通过show profile 可以知道，时间都花在哪里

1.4 ）通过session级别的status，可以知道为什么时间会花在那里

flush status;

select xx from tt where ff ;

show status where variable\_name like 'Handler%' or Variable\_name like 'Created%';

2) 完成一项任务的时间分两个部分 执行时间和等待时间

如何优化执行时间呢 --比较简单？

2.1) 降低子任务数量

2.2) 降低子任务的执行频率

2.3) 提升子任务的执行效率并且判断任务在什么时间执行最长

如何优化等待时间呢 --比较复杂？

2.4) 一般是由于资源竞争导致，要用合适的工具找到竞争点。

2.5) 判断任务在什么地方被阻塞的时间最长。

3) 通过slow，可以找到值得优化的SQL

awk '/^# Time:/{print $3, $4, c;c=0}/^# User/{c++}' dbbak10-001-slow.log --可以统计出每个时间点的slow 数量，精度比较细

3.1) 执行总时间最多的SQL

3.2) 单条SQL执行时间最多的SQL

4) 三种轻量级别的SQL抓取 show processlist & tcpdump & slow-query 解析工具可以用：pt-query-digest 解析tcpdump和slow query

msyql -e 'show proceslist\G' | grep State: | sort | uniq -c | sort -rn --轻量级 （show processlist && show status）

5) 找到最需要优化的SQL后，可以开始跟踪分析单条SQL来获得更加底层实际的东 西，目前最好的三种方法是a）show profile b）show status c）slow query条目

a）show profile

SQL> set profiling=1;

SQL> select \* from table;

SQL> show profiles;

SQL> show profile for query 1;

格式化输出：

SQL> set @query\_id = 1;

SQL> SELECT STATE,SUM(DURATION) AS Total\_R,

ROUND(

100\*SUM(DURATION) /

(SELECT SUM(DURATION)

FROM INFORMATION\_SCHEMA.PROFILING

WHERE QUERY\_ID = @query\_id

), 2) AS Pct\_R,

COUNT(\*) AS CallS,

SUM(DURATION) / COUNT(\*) AS "R/CALL"

FROM INFORMATION\_SCHEMA.PROFILING

WHERE QUERY\_ID = @query\_id

GROUP BY STATE

ORDER BY Total\_R DESC;

当然，通过show profile 可以知道时间主要花在什么地方，但是你不知道为什么 会花在那些地方？这是时候就必须要跟踪堆栈来找到进一步的原因了。

查看是否使用了磁盘临时表还是内存临时表：

flush status;

sql;

show status where variable\_name like 'Handler%' or variable\_name like 'Created%';

b) show status

SQL> 句柄计数器 handler counter,临时文件，表计数器

SQL> flush status ; 刷新绘画级别的状态值。

SQL> select \* from table;

SQL> show status where variable\_name like 'Handler%' or Variable\_name like 'Created%'; --可以看到是否利用了磁盘临时表，而explain是无法看到的。

6. 监控点 -- 通过监控状态数据可以发现哪些地方是异常的，然后再具体分析异 常时间点的日志。

a）show global status; --开销比较低

b）show processlist | grep state; 或者使用innotop --开销比较低

c）slow query + pt-query-digest

d）show innoDB status;

e) vmstat

f) iostat

7. 关于索引统计

发生过一件事情，show table status看到的大小100M，但是实际物理大小10G，通过这个发型索引统计有的时候非常不准确

这里简单介绍下：

innodb\_stats\_persistent=on , db重启后不会清空，不需要重新收集

innodb\_stats\_persistent=off， db重启后统计信息清空，需要重新收集统计

1、针对是否持久化统计信息mysql可以通过innodb\_stats\_persistent参数来控制

　　2、针对统计信息的时效性，mysql通过innodb\_stats\_auto\_recalc参数来控制是否自动更新

　　3、针对统计信息的准确性，mysql通过innodb\_stats\_persistent\_sample\_pages 参数来控制更新

4、mysql通过analyze table 语句来手动的更新统计信息

5、mysql> select \* from innodb\_table\_stats; last\_update可以查看索引统计的最后更新时间

6、当索引统计不准确的时候，可以通过analyze table来更新索引统计信息，让执行计划更加准确。

如果这样做后，执行计划还是不准确，那么可以试图调大innodb\_stats\_persistent\_sample\_pages，让索引页收集的更加多，让执行计划更准确

8. 关于索引选择性: 字段1 building\_id，字段2 status

单索引字段的索引选择性： select count(distinct building\_id)/count(\*) as selectivity from community\_units;

组合索引的索引选择性： select count(distinct (concat(building\_id,status)))/count(\*) as selectivity from community\_units;

组合前缀的索引选择性： select count(distinct (concat(building\_id,left(status,2))))/count(\*) as selectivity from community\_units;

得到的结果越接近1，效果越好

**cpu 模式调节**

[https://wiki.archlinux.org/index.php/CPU\_frequency\_scaling](https://yq.aliyun.com/go/articleRenderRedirect?url=https%3A%2F%2Fwiki.archlinux.org%2Findex.php%2FCPU_frequency_scaling)

* **有哪几种模式**

| **Governor** | **Description** |
| --- | --- |
| ondemand | Dynamically switch between CPU(s) available if at 95% cpu load |
| performance | Run the cpu at max frequency |
| conservative | Dynamically switch between CPU(s) available if at 75% load |
| powersave | Run the cpu at the minimum frequency |
| userspace | Run the cpu at user specified frequencies |

* **如何查看当前的cpu模式**

cat /sys/devices/system/cpu/cpu0/cpufreq/scaling\_governor

* **如何查看cpu支持哪几种模式**

cat /sys/devices/system/cpu/cpu2/cpufreq/scaling\_available\_governors

* **如何设置**

1. bios里面设置
2. os设置

**NC 传送**

传送文件

目的主机监听

nc -l 监听端口<未使用端口> > 要接收的文件名

nc -l 4444 > cache.tar.gz

源主机发起请求

nc 目的主机ip 目的端口 < 要发送的文件

nc 192.168.0.85 4444 < /root/cache.tar.gz

=============================================

==传送文件夹==

接收方的命令：

nc -l ${ip} 4444 | tar xf -

传送方的命令：

tar -cvf - ppc\_\* | nc ${ip} 4444

**rsync**

核心算法：http://www.oschina.net/question/28\_54213?fromerr=DHoiMICG

小bug：如果rsync一段时间，突然不传了，且流量中断，不妨加上这个参数试试 /usr/bin/rsync --sockopts=SO\_RCVBUF=10485760

配置：/etc/rsyncd.conf

uid = root

gid = root

use chroot = no

max connections = 64

pid file = /var/run/rsyncd.pid

lock file = /var/run/rsync.lock

log file = /var/log/rsyncd.log

[dbbak]

path = /data/dbbackup

use chroot = no

ignore errors

read only = no

list = no

[Binlog]

path = /data/BINLOG\_BACKUP

use chroot = no

ignore errors

read only = no

list = no

[fullbak]

path = /data/FULL\_BACKUP

use chroot = no

ignore errors

read only = no

list = no

启动： /usr/bin/rsync --daemon

限速100k/s传输 ：

/usr/bin/rsync -av --progress --update --bwlimit=100 --checksum --compress $file root@$ip::dbbak

正常传输：

/usr/bin/rsync -av --progress $file root@$ip::dbbak

**pigz使用**

* **常用知识普及**

错误的写法：nohup tar -cvf - xx\_20151129 | pigz -p 24 > xx\_20151129.tar.gz & --一定不能加nohup，因为中间有管道符，不能传递下去的

错误的代价：

tar: This does not look like a tar archive

tar: Skipping to next header

tar: Exiting with failure status due to previous errors

以上错误的案例中，为此付出过很大的代价，哭晕在厕所N次了...

正确的写法： tar -cvf - xx\_20151129 | pigz -p 24 > xx\_20151129.tar.gz &

* **用法**

\* 压缩

tar cvf - 目录名 | pigz -9 -p 24 > file.tgz

pigz：用法-9是压缩比率比较大，-p是指定cpu的核数。

\* 解压1

pigz -d file.tgz

tar -xf --format=posix file

\* 解压2

tar xf file.tgz

**axel & httpd 多线程数据传输**

\* axel 下载&安装

wget -c http://pkgs.repoforge.org/axel/axel-2.4-1.el5.rf.x86\_64.rpm

rpm -ivh axel-2.4-1.el5.rf.x86\_64.rpm

\* axel 核心参数

-n 指定线程数

-o 指定另存为目录

\* httpd服务搭建与配置

yum install httpd

\* httpd配置主目录

/etc/httpd/conf/httpd.conf

[xx html]# cat /etc/httpd/conf/httpd.conf | grep DocumentRoot

# DocumentRoot: The directory out of which you will serve your

#DocumentRoot "/var/www/html" --注释

DocumentRoot "/data/dbbackup/html" --配置成容量大的地址

\* 开启httpd服务

service httpd restart

\* 下载数据

目的地ip shell> nohup axel -n 10 -v -o /data/dbbackup/ http://$数据源ip/xx\_20151129.tar.gz &

**git 基本**

1. git add xx

2. git commit -m 'xx'

3. git pull

4. git push

**如何模拟网络延迟或丢包**

* 模拟网络eth0 timeout 1000ms

tc qdisc add dev eth0 root netem delay 1000ms

* 模拟网络eth0丢包率 10%

tc qdisc add dev eth0 root netem loss 10%

* 删除以上tc命令导致的网络延迟或者丢包规则

tc qdisc del dev eth0 root

**如何模拟网络故障**

* host1 网络断掉，只允许host2 访问

host1> iptables -A INPUT -p tcp -s host2 -j ACCEPT

host1> iptables -A INPUT -p tcp -s 0.0.0.0/0 -j DROP

* host1 网络断掉，只允许host2的22端口访问

host1> iptables -A INPUT -p tcp -s host2 --dport 22 -j ACCEPT

host1> iptables -A INPUT -p tcp -s 0.0.0.0/0 -j DROP

* 恢复host1 网络

host1> service iptables restart

**ansible 基础知识**

**文档**

官方： http://docs.ansible.com/

个人： http://sofar.blog.51cto.com/353572/1579894

**基础用法**

* ssh互信

1) 不需要加入key，也能登陆到所有机器

2）前提是：

ssh-add --mac本地，线下，将私钥加入到内存

ssh -A root@xx ； --会将私钥传送到远端机器

ssh-add -L 查看下。 --查看私钥是否传送过来

* yaml

- hosts: etl

remote\_user: root

tasks:

- shell: cat /home/mysql/xx.pl

- copy: src=files/rsync dest=/usr/bin/

- template: src=files/xx.pl dest=/home/mysql/

* hosts

[test1]

10.x.x.x bak\_dest\_ip=10.y.y.y bak\_source\_port=xx

[etl]

10.x.x.x bak\_dest\_ip=10.y.y.y bak\_source\_port=xx

* files

\*.pl

\*.file

* 常用语法

\* 命令中如果有管道等多种命令，需要用bash -c ，并且引号起来

\* -T：ping延迟时间 -f：线程数 -i：后面接hosts文件，xx标签 -m：command 命令模式 -a：命令内容

ansible -T 2 -f 1 -i ./hosts etl -m command -a "bash -c 'cat /home/mysql/xx.pl |grep bin/rsync'"

\* playbook方式跑ansible

ansible-playbook -i ./hosts rsync.yaml

**网络流量诊断**

* **tools**

\* ifstat

\* iftop

iftop -nNP -i tunl1 —看出口流量

iftop -nNP —看看整体的

\* 查看ip1 与 ip2 之间的流量

root@ip1> iftop -F $ip2/32 ============= iftop -F $P{ip}/32

\* 如何查看一个机器上哪个端口占用的流量最大

1> iftop 进入界面

2> 按 N

3> 按 S

**vim块操作**

* [选择] -> 在普通模式下按ctrl+v或者v进入块操作模式

v（小写）　　　　 按字符选择，选中按下V时光标所在的字符到当前光标所在字符间的内容

V（大写)　　　　　按行选择

[Ctrl]+V　　　　　选择矩形字符块

* [动作] -> 通过光标移动选中内容，可以进行ydp操作

y:复制选中内容到粘贴板

d:删除选中内容

p:用粘贴板里的内容替换选中的内容

=:对齐选中内容

对于矩阵字符块：[Shift] + i xxx [esc] :把xxx写到每一行的光标前面的位置

* [替换] -> 批量缩进或反缩进，类似于文本编辑器中的格式化

选中多行，按I(大写)进入插入模式，写入Tab，之后按ESC，即可完成批量缩进的功能

也可以写入内容，到选中的每一行的光标位置

**TGW 接口**

* TGW相关问题

\* 根据vip，vport，找到rsip(不需要固定key，因为不需要访问real-server)

wget -O- --post-data 'data={ "operator":"xx\_DEV", "rulelist":[ { "vip":"'"$vip"'", "vport":'"$vport"', "protocol":"TCP" } ] }' "http://10.126.70.51/cgi-bin/fun\_logic/bin/public\_api/getrs.cgi"

\* 将vip 从rsip下线（需要固定key，因为要访问real-server）

$del\_rs=`wget -O- --post-data 'data={ "client\_type" : "x'x\_DB", "ignore\_exist\_error" : false, "operator" : "xx\_DEV", "rs\_type" : "linux\_tunl", "need\_setup\_rs" : true, "op\_type" : "'del'", "rule\_list" : [ { "rule\_group":[ { "vip":"'$vip'", "vport":'$vport', "protocol":"TCP" } ], "rs\_os\_type":"linux", "rs\_list":[ { "rs\_ip":"'$source\_ip'", "rs\_port":'$source\_port', "rs\_weight":100 } ] } ], "sync" : true }' 'http://xx/cgi-bin/fun\_logic/bin/public\_api/op\_rs.cgi' 2>/dev/null`;

\* 将vip 从rsip上线（需要固定key，因为要访问real-server）

$add\_rs=`wget -O- --post-data 'data={ "client\_type" : "xx\_DB", "ignore\_exist\_error" : false, "operator" : "xx\_DEV", "rs\_type" : "linux\_tunl", "need\_setup\_rs" : true, "op\_type" : "'add'", "rule\_list" : [ { "rule\_group":[ { "vip":"'$vip'", "vport":'$vport', "protocol":"TCP" } ], "rs\_os\_type":"linux", "rs\_list":[ { "rs\_ip":"'$target\_ip'", "rs\_port":'$target\_port', "rs\_weight":100 } ] } ], "sync" : true }' 'http://xx/cgi-bin/fun\_logic/bin/public\_api/op\_rs.cgi' 2>/dev/null`;

\* 问题

其实TGW的接口会做两步操作：1，操作TGW server上的配置 2，操作real-server上的配置，这两步应该是原子操作。

> 假设：1 成功，2失败，那么就会导致tgw上的配置，请求均切换了，但是real-server却没做改变，导致两端出现问题。

临时解决方案：2失败了，那么手动执行2的操作。假设在TGW上执行的操作是del\_rs,那么可以在read-server上执行 /usr/local/realserver/RS\_TUNL0/etc/setup\_rs.sh -c (将本地的rsip和vip直接的配置关系清理掉)

> 假设：1 没有执行，2 执行了，那么就会导致tgw上的配置没变，但是real-server的配置改变了，导致从tgw来的请求均在real-server上找不到，出现问题。

临时解决方案：2执行了，那么手动让2还原到没有执行的状态。假设在read-server上误清理掉相关rs配置（/usr/local/realserver/RS\_TUNL0/etc/setup\_rs.sh -c），那么可以调用add\_rs 来恢复。

* vip漂移脚本

\* 位置： db\_sys: /data/online/tools/tgw\_vip\_shift

usage:

python vip\_shift.py view --vip=$vip --vip\_port=$vip\_port

python vip\_shift.py del --vip=$vip --vip\_port=$vip\_port --src\_ip=$src\_ip --src\_port=$src\_port

python vip\_shift.py add --vip=$vip --vip\_port=$vip\_port --target\_ip=$target\_ip --target\_port=$target\_port

python vip\_shift.py change --vip=$vip --vip\_port=$vip\_port --src\_ip=$src\_ip --src\_port=$src\_port --target\_ip=$target\_ip --target\_port=$target\_port

[-h] [--vip VIP] [--vip\_port VIP\_PORT] [--src\_ip SRC\_IP]

[--src\_port SRC\_PORT] [--target\_ip TARGET\_IP]

[--target\_port TARGET\_PORT] [-v VERBOSITY]

{del,add,change,view}

**SSH 如何跳过输入密码，只允许认证模式**

ssh -o BatchMode=yes -o PasswordAuthentication=no root@ip

**如何永久清空一台机器上的history**

\* 立即清空里的history当前历史命令的记录

history -c

\* 要求bash立即更新history文件

history -w

**nohup 失效的问题**

* 在secureCRT 或者 iterm2 等类似终端，使用nohup 执行命令，为啥退出后，后台执行的命令也就停止了？

\* 错误的做法

1. nohup xx\_cmd &

2. 点击左上角或者右上角的xx按钮退出

3. 然后发现，刚刚在后台的命令异常终止了

\* 正确的做法

1. nohup xx\_cmd &

2. 必须显示的 exit 退出shell，接下来，你想干嘛干嘛

3. 然后发现，刚刚在后台的命令，安然无恙，放心睡觉吧

**如何让iTerm2 tab页面显示从哪台机器上登陆过来的**

sudo vi /bin/go

#!/bin/sh

if [ "$1" = "" ]; then

echo "pleaes input ip"

else

echo "go ==> ssh -A root@$1"

echo "\033]0;$1\007"

ssh -A root@$1

# ssh -A Keithlan@$堡垒机 -t "ssh root@xx"

fi

**如何查看memcache/redis当前哪个链接数最多**

ss | grep '$ip:$port' | awk '{print $5}' | awk -F ':' '{print $1}' | sort -nr | uniq -c | sort -nr

**kibana简单语法**

\* 地址：http://opses.corp.anjuke.com/

\* 注意：选择搜索的时间段，右上角

\* filter:

语法： message:(+SQLSTATE +connection) 每个关键字用+号，不能有空格

\* 选择log name:

ops-user-userlog\*

ops-xinfang-userlog\*

ops-broker-userlog\*

\* 哪些关键字跟DB紧密相关

SQLSTATE

connection time out

too many connection

max\_user\_connections

**定位系统问题的工具和方法**

\* perf top -G : 当CPU性能出现问题的时候，使用最佳 --注意： 会卡住,导致linux宕机，小心 : http://blog.51cto.com/1152313/1767927

[ ] perf record -g --保留文件，稍后可以用 perf report分析

[ ] 如果需要分析某一个进程，可以加 -p ， perf record -g -p $pid

[ ] perf top -g 实时分析，不保留数据到文件

[ ] 如果需要分析某一个进程，可以加 -p ， perf top -g -p $pid

\* pidstat 1 5 ：分析cpu问题的好工具

\* dstat

\* pstack : 当进程卡住的时候，使用效果最佳

\* ss -tnlp

\* nstat

1. 检查back\_log 是否设置合理，如果不合理，那么就会看到很多如下信息,代表客户端的请求会connect timeout

linux> nstat -a | grep -i 'drops\|Overflow'

TcpExtListenOverflows 208539 0.0

TcpExtListenDrops 236999 0.0

\* top :

1. top -Hp $pid

2. top , 然后输入f，然后输入p和y ， 就可以看到top显示中对了2列， p对应的是swap(查看swap的进程)，y对应的是wchan(Sleeping in Function),很实用

\* gdb https://groups.google.com/forum/#!topic/mechanical-sympathy/QbmpZxp6C64

gdb -p $id

info thread

thread $id

bt

\* strace

第一种： strace -o /data/dbbackup/strace.log -fp $pid

第二种： 跟踪某些具体的操作 strace -o /data/dbbackup/strace.log -T -tt -f -e trace=read,open -p $pid

\* other

http://blog.donghao.org/2014/04/24/%E8%BF%BD%E8%B8%AAcpu%E8%B7%91%E6%BB%A1/

如果perf都用不了，可以尝试 echo t > /proc/sysrq-trigger ， 然后dmesg 或者查看kernel日志

如果上述方法还不行， 可以尝试 /proc/{pid}/wchan

[**atop的使用方法**](https://yq.aliyun.com/go/articleRenderRedirect?url=atop_doc.pdf)

查看历史的top

atop -r /var/log/atop/atop\_20180906 -b 4:00 -e 5:00 --查看某台机器凌晨4点~5点的top日志， t 下一页，T 上一页

**如何优化swap被占用的情况**

* 处理原则

1. 如果swap占用的内存比较小(500M以内)，那么通过 swapoff -a && swapon -a 可以快速释放掉(此操作有风险，谨慎)

2. 如果swap占用的内存比较大，则需要保证两点

2.1 必须保证linux的空闲内存 大于 swap占用空间

2.2 然后通过下面的方法找到占用swap最多的进程，优化处理进程，让其达到第一点后再释放swap

* 发现swap占用最多的进程

1. for i in $(ls /proc | grep "^[0-9]" | awk '$0>100'); do awk '/Swap:/{a=a+$2}END{print '"$i"',a/1024"M"}' /proc/$i/smaps;done| sort -k2nr | head

有些linux无法跑上面的程序，可参考下一条命令

2. for i in $(ll /proc | awk '{print $9}' | grep "^[0-9]" | awk '$0>100'); do awk '/Swap:/{a=a+$2}END{print '"$i"',a/1024"M"}' /proc/$i/smaps;done| sort -k2nr | head

* 查看机器有哪些服务

ss -tpnl

* 如何是否os的cache

cat /proc/sys/vm/drop\_caches

sync;sync;sync;

sync;sync;sync;

sync;sync;sync;

echo 3 > /proc/sys/vm/drop\_caches

sync;sync;sync;

sync;sync;sync;

echo 0 > /proc/sys/vm/drop\_caches

sync;sync;sync;

sync;sync;sync;

cat /proc/sys/vm/drop\_caches