|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 修订内容 | 修订者 | 修订时间 | 备注 |
| V1.0 | CPU |  | 2020/2/8 |  |
| V1.1 | 内存 |  | 2020/2/9 |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 目录

目录

[全局视野 4](#_Toc1506)

[系统分析 5](#_Toc26385)

[LINUX 60秒分析 5](#_Toc18379)

[BCC 60秒分析 5](#_Toc25211)

[系统状态 6](#_Toc16070)

[进程状态 9](#_Toc15361)

[系统分析图 11](#_Toc2155)

[CPU 12](#_Toc10121)

[CPU基准数据 12](#_Toc2830)

[分析策略 14](#_Toc29712)

[CPU分析 14](#_Toc15306)

[CPU相关工具 14](#_Toc25617)

[execsoonp 15](#_Toc5578)

[exitsnoop 17](#_Toc17952)

[runqlat 18](#_Toc21772)

[runqlen 18](#_Toc14132)

[runqslower 18](#_Toc22502)

[cpudist 20](#_Toc1150)

[profile 20](#_Toc26141)

[offcputime 20](#_Toc4286)

[syscount 22](#_Toc20815)

[argdist 23](#_Toc370)

[founccount 24](#_Toc16885)

[softirqs 26](#_Toc23128)

[hardirqs 27](#_Toc23895)

[smpcalls 27](#_Toc8724)

[llcstat 27](#_Toc5147)

[练习 27](#_Toc16639)

[上下文切换 28](#_Toc9819)

[模拟工具 28](#_Toc4962)

[分析方法 28](#_Toc4456)

[中断 29](#_Toc12980)

[软中断统计工具 29](#_Toc31907)

[内存 30](#_Toc21904)

[cache命中 31](#_Toc9742)

[内存泄漏 32](#_Toc25717)

[IO子系统 33](#_Toc5257)

[磁盘性能基准数据 33](#_Toc4511)

[跟踪进程、线程系统调用 34](#_Toc17625)

[FIO基准测试 34](#_Toc29805)

[回放IO 35](#_Toc28638)

[网络系统 35](#_Toc27953)

[端到端传说延时量级 36](#_Toc15284)

[测试网络性能工具 37](#_Toc31848)

[iperf 37](#_Toc30345)

[网卡 37](#_Toc20636)

[DNS 38](#_Toc12654)

[Dnsmsq 38](#_Toc10377)

[缓解DDos攻击 39](#_Toc625)

[模拟DDos 攻击 39](#_Toc21309)

[分析 39](#_Toc12478)

[缓解方案 40](#_Toc9791)

[网络延时 40](#_Toc25246)

[HTTP基准测试工具 40](#_Toc16911)

[分析过程 41](#_Toc25733)

[CPU 42](#_Toc32629)

[数据访问时间 42](#_Toc6555)

[数据访问指令周期 42](#_Toc29819)

[数据访问时间 44](#_Toc26964)

[Perf 45](#_Toc13833)

[perf生成火焰图 46](#_Toc23326)

[汇编到函数查看 47](#_Toc25458)

[常用性能数字 48](#_Toc22574)

# 全局视野

全局拓扑

数据流向

系统 -> 进程 -> 线程

**CPU（调度）**

**内存**

**文件系统**

**网络**

**内核**

**磁盘IO**

# 系统分析

## LINUX 60秒分析

uptime

dmesg | tail

vmstat 1

mpstat -P ALL 1

pidstat 1

iostat -xz 1

free -m

sar -n DEV 1

sar -n TCP,ETCP 1

top

## BCC 60秒分析

BCC工具监测清单

execsnoop

opensnoop

ext4slower

biolatency

biosnoop

cachestat

tcpconnect

tcpretrans

runqlat

profile

详见 《BCC学习笔记》

## 系统状态

1. 查看系统概况

top/htop

perf top

1. 查看系统负载变化情况

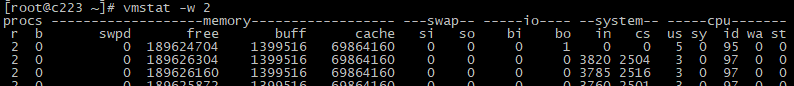
uptime

watch -d uptime

while ((1)); do uptime; sleep 3; done;

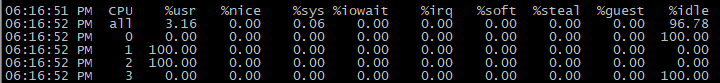
1. 查看总体cpu使用率、中断、上下文切换

vmstat -wtSM 2



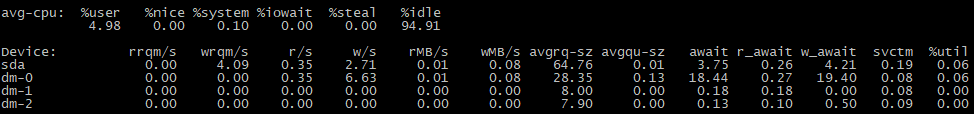
1. 查看各个CPU使用情况

mpstat -P ALL 1



1. 查看磁盘IO情况

iostat -kzx 2



1. 查看接口收发情况

sar -n DEV 1



1. perf record

perf record -g -- sleep 10

perf report --stdio

perf sched record -- sleep 5

perf sched latency

1. 查看进程、线程cpu使用率

pidstat 1

pidstat -t 1

1. 查看套接字状态

netstat -anp | more

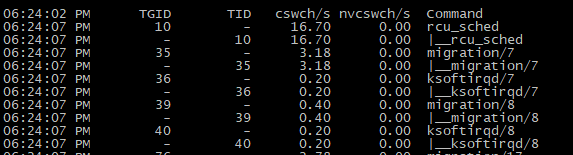
1. 查看系统上下文切换情况

# 如果上下切换比较高，可以监测下

pidstat -w 5

# 查看线程上下文切换

pidstat -wt 5



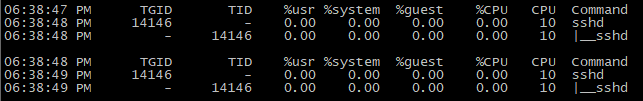
## 进程状态

1. 进程、线程CPU使用情况

pidstat -p PID 1



pidstat -p PID -t 1



1. 进程、线程上下文切换

pidstat -p PID -w 1

pidstat -p PID -wt 1



1. 进程、线程IO

pidstat -p PID -d 1

pidstat -p PID -dt 1



1. perf trace

perf trace -s -p PID

perf trace -p PID -e write

# 或者strace -p PID

lsof -p PID

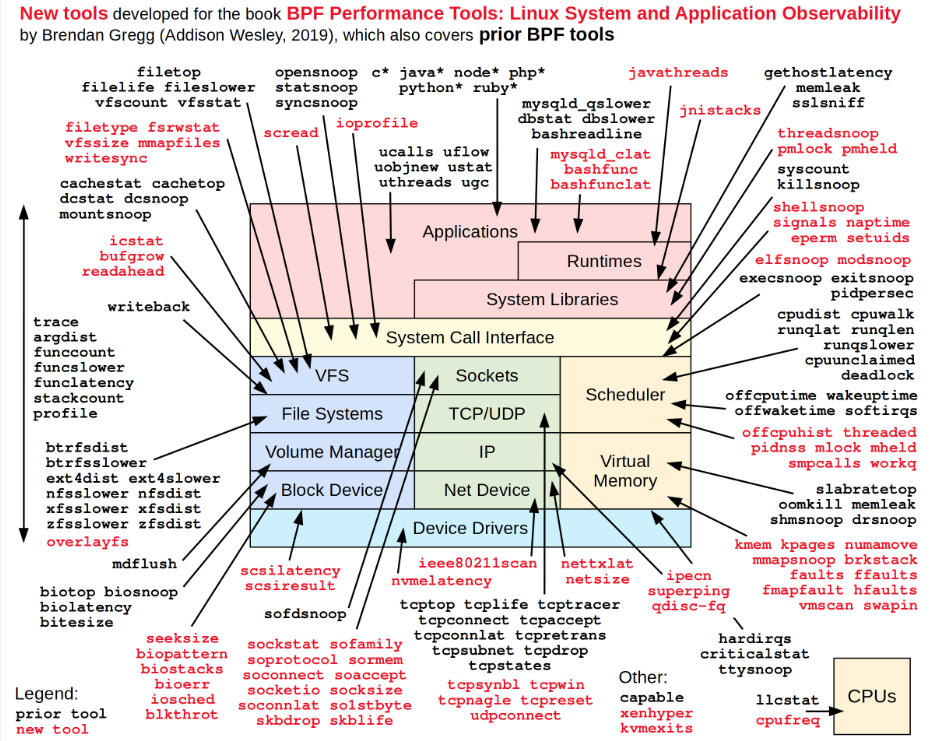
1. 进程打开的文件描述符

lsof -p PID



1. 火焰图

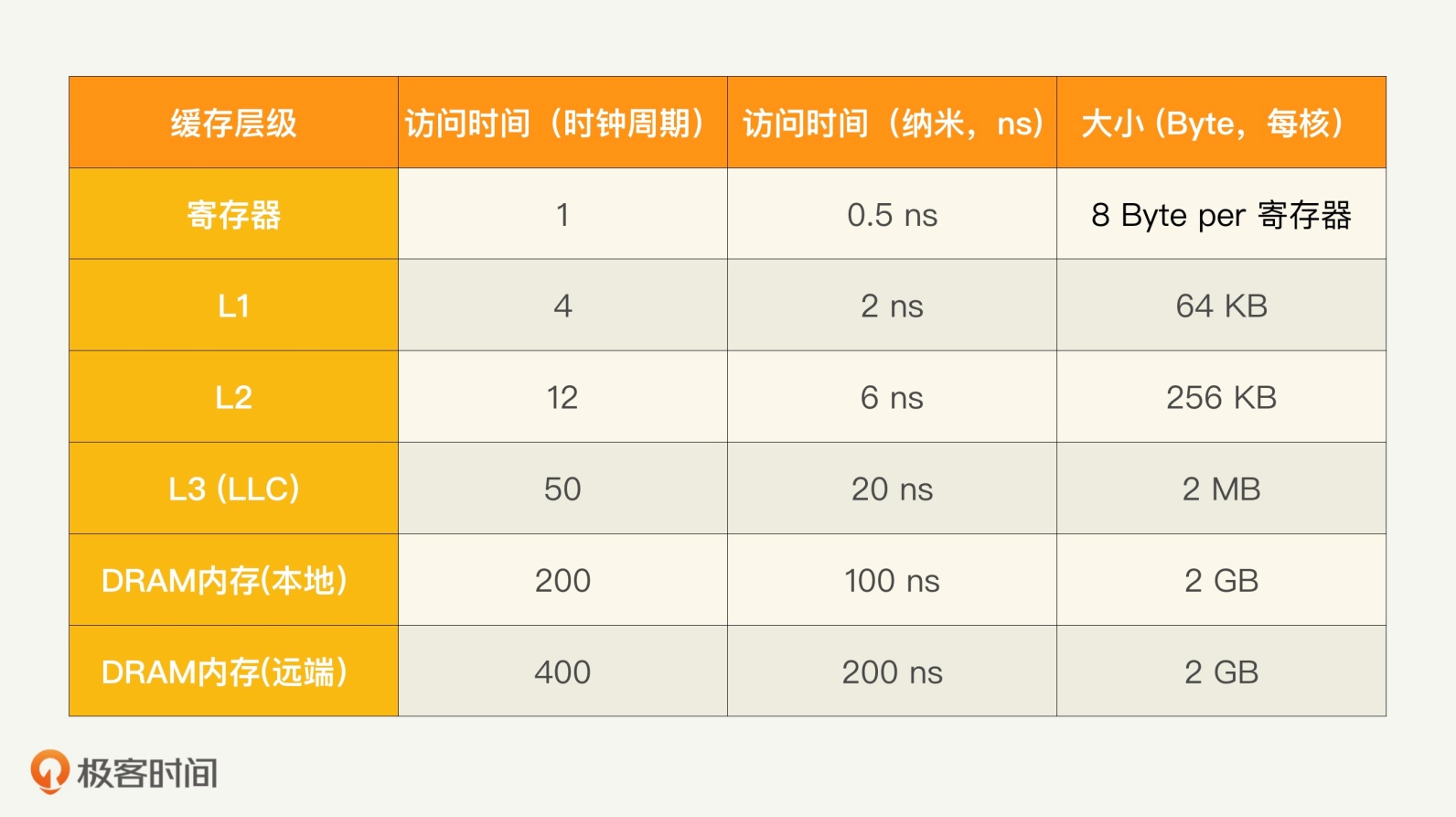
# 系统分析图



# CPU

## CPU基准数据

CPU访问时间，与cpu相关，下表方便了解量级



1. 指令分支延迟，约10ns

2. 互斥加锁和解锁，约10ns以上

3. **上下文切换**，约1us

## 分析策略

1. 查看整体CPU利用率

mpstat -P ALL 1

top

pidstat 2

perf top

bpftrace

vmstat -wtSM 2

1. 按进程分析

top

pidstat 2 -p PID

perf top -g

perf top -g -p PID

bpftrace

## 模拟工具

stress

## CPU分析

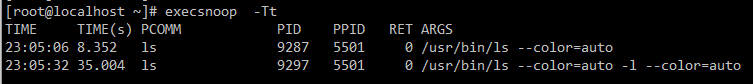
### CPU相关工具

|  |  |  |  |
| --- | --- | --- | --- |
| 工具 | 源 | 分析对象 | 描述 |
| execsnoop | BCC/BT | 调度 |  |
| exitsnoop | BCC | 调度 |  |
| runqlat | BCC/BT | 调度 |  |
| runqlen | BCC/BT | 调度 |  |
| runqslower | BCC | 调度 |  |
| cpudist | BCC | 调度 |  |
| profile | BCC | CPU |  |
| offcputime | BCC | 调度 |  |
| syscount | BCC | 系统调用 |  |
| argdist | BCC | 系统调用 |  |
| trace | BCC | 系统调用 |  |
| funccount | BCC | 软件 |  |
| softirqs | BCC | 中断 |  |
| hardirqs | BCC | 中断 |  |
| smpcalls |  | 内核 |  |
| llcstat | BCC | PMC |  |

<https://github.com/brendangregg/bpf-perf-tools-book>

### execsoonp

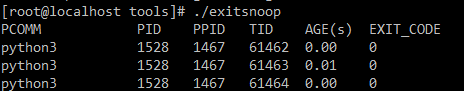
execsnoop -Tt



execsnoop.bt

|  |
| --- |
| #!/usr/bin/env bpftrace  BEGIN  {  printf("%-10s %-5s %s\n", "TIME(ms)", "PID", "ARGS");  }  tracepoint:syscalls:sys\_enter\_exec\*  {  printf("%-10u %-5d ", elapsed / 1000000, pid);  join(args->argv); |

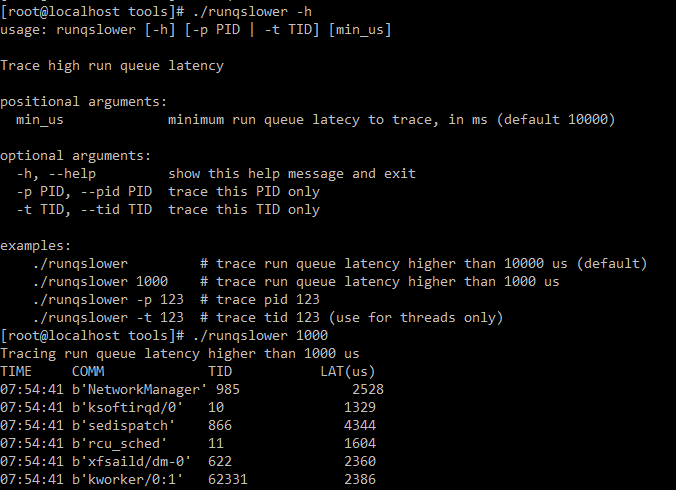
### exitsnoop



### runqlat

### runqlen

### runqslower



### cpudist

线程唤醒后在CPU执行的时长分布

### sar

sar -P ALL

sar -P ALL -f /var/log/sa/..

### profile

CPU上的运行时间。

可以知道那个函数占用cpu，不知道是慢还是调用的次数多， 使用funccount辅助确认调用次数。

profile -af 30 > out.stacks01

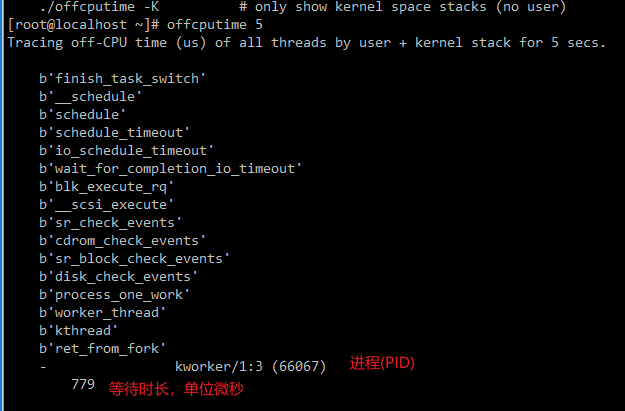
# 生成火焰图

git clone https://github.com/brendangregg/FlameGraph.git

./FlameGraph/flamegraph.pl --color=java < ./out.stacks01 > out.svg

### offcputime

统计线程阻塞和脱离CPU运行的时间，同时输出调用栈信息，以便理解阻塞原因。



#### 脱离CPU时间的火焰图

# 生成火焰图

git clone https://github.com/brendangregg/FlameGraph.git

offcputime -fKu 5 > out.offcputime01.txt

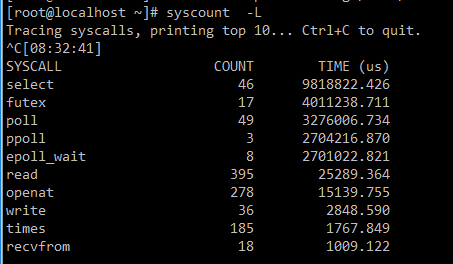
./FlameGraph/flamegraph.pl --hash --bgcolors=blue --title="OFF-CPU Time Flame Graph" < out.offcputime01.txt > out.offcputime01.svg

### syscount

syscount -i 1

syscount -Pi 1

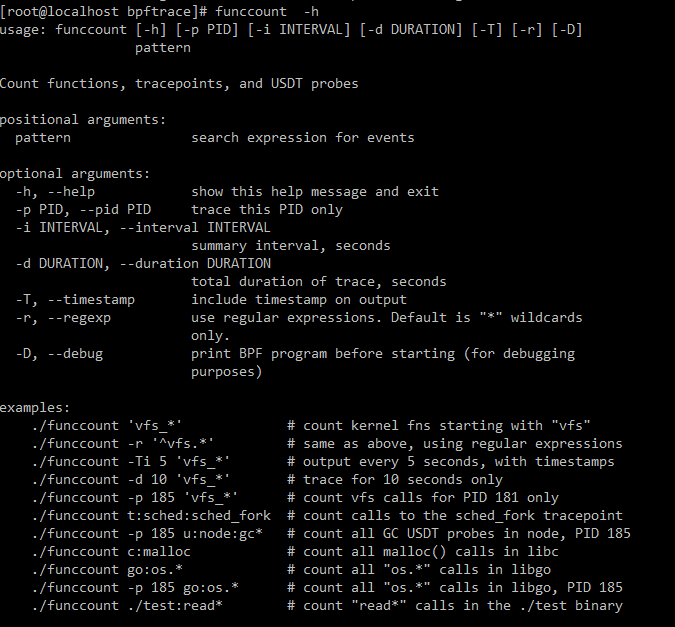
syscount -L



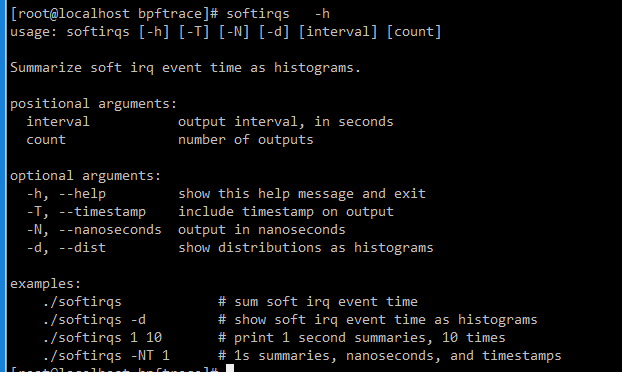
### argdist

# TODO

### founccount



### softirqs



### hardirqs

### smpcalls

### llcstat

利用PMC来按进程输出最后一级缓存的命中率

## 练习

创建两个繁忙进程，绑定CPU 0。 分析cpu

taskset -c 0 sh -c 'while :; do :; done' &

taskset -c 0 sh -c 'while :; do :; done' &

# 上下文切换

# 放到哪个章节？？？

中断上下文、进程上下文、线程上下文

pidstat -w 5

08:18:26 UID PID cswch/s nvcswch/s Command

08:18:31 0 1 0.20 0.00 systemd

cswch：每秒自愿上下文切换

**nvcswch：每秒非自愿上下文切换，cpu不够用的情况，这个指标会比较高**

## 模拟工具

# 以 10 个线程运行的基准测试，模拟多线程切换的问题

sysbench --num-threads=10 --thread-yields=100000 --max-time=300 --test=threads run

## 分析方法

简单来说,平均负载是指单位时间内,系统处于可运行状态和不可中断状态的平均进程数

while ((1)); do uptime; sleep 5; done;

# 中断

硬中断

cat /proc/interrupts

软中断

cat /proc/softirqs

# TODO 比较好的查看中断次数的工具

watch -d cat /proc/interrupts

watch -d cat /proc/softirqs

watch -d "/bin/cat /proc/softirqs | /usr/bin/awk 'NR == 1{printf \"%13s %s\n\",\" \",\$1}; NR > 1{printf \"%13s %s\n\",\$1,\$2}'"

## 软中断统计工具

# 内存

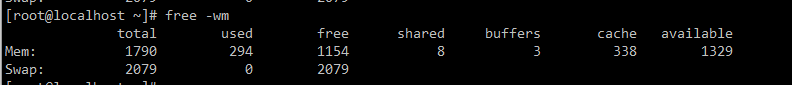
## 分析策略

剩余内存，换页，交换

## MEM分析

### free

free -wm



Buffer 是对磁盘数据的缓存，而 Cache 是文件数据的缓存，它们既会用在读请求中，也会用在写请求中。

### ps

# 按照进程排序

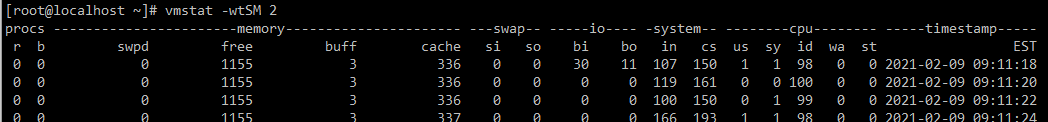
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem

# mem top20

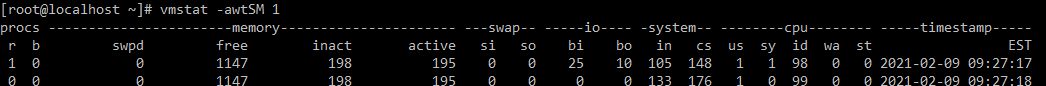
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head -21

### vmstat

vmstat -wtSM 2

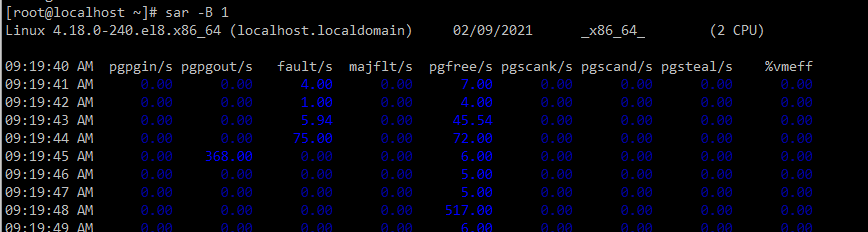


vmstat -awtSM 1



### sar

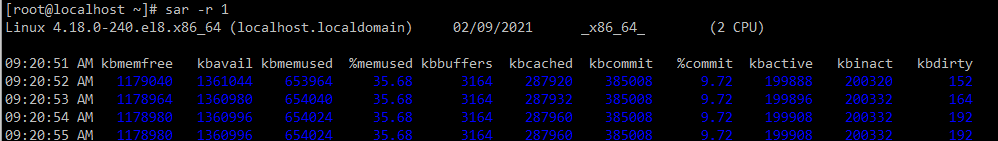
sar -B 1



# 历史内存情况

sar -r

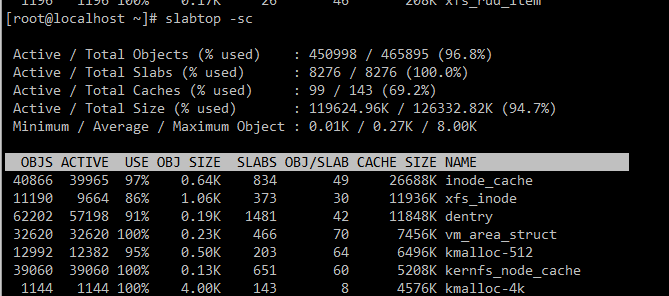
sar -r 1



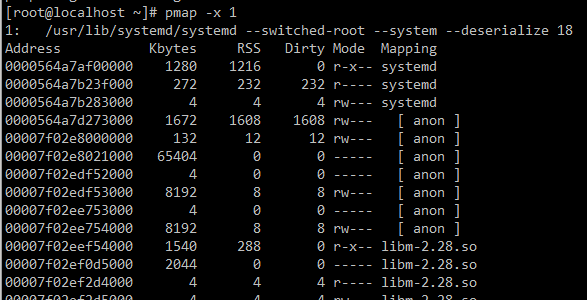
### slabtop

slabtop -sc

# -sc 按大小排序



### pmap



### brkstack.bt

stackcount -PU t:syscalls:sys\_enter\_brk

### shmsnoop

### 缺页异常

stackcount -U t:exceptions:page\_fault\_user

stackcount t:exceptions:page\_fault\_kernel

# TODO

### vmscan

# TODO

funccount 't:vmscan:\*'

vmscan.bt

### drsnoop

### swapin

识别哪些进程进行了换页操作

## 传统工具

dmesg

swapon

free

ps

pmap

sar

perf

ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu | head

cat /proc/meminfo

free -hw -c 3600 -s 2

-c： 次数

-s： 每隔2秒

Buffer 是对磁盘数据的缓存

Cache 是文件数据的缓存

numactl --hardware

available: 1 nodes (0)

node 0 cpus: 0 1

node 0 size: 7977 MB

node 0 free: 4416 MB

...

## cache命中

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 4052245BD4284CDD

echo "deb https://repo.iovisor.org/apt/xenial xenial main" | sudo tee /etc/apt/sources.list.d/iovisor.list

sudo apt-get update

sudo apt-get install -y bcc-tools libbcc-examples linux-headers-$(uname -r)

export PATH=$PATH:/usr/share/bcc/tools

cachestat 1 3

TOTAL MISSES HITS DIRTIES BUFFERS\_MB CACHED\_MB

2 0 2 1 17 279

2 0 2 1 17 279

2 0 2 1 17 279

cachetop

11:58:50 Buffers MB: 258 / Cached MB: 347 / Sort: HITS / Order: ascending

PID UID CMD HITS MISSES DIRTIES READ\_HIT% WRITE\_HIT%

13029 root python 1 0 0 100.0% 0.0%

cachestat 提供了整个系统缓存的读写命中情况

## 内存泄漏

# Install bcc

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 4052245BD4284CDD

echo "deb https://repo.iovisor.org/apt/bionic bionic main" | sudo tee /etc/apt/sources.list.d/iovisor.list

sudo apt-get update

sudo apt-get install -y bcc-tools libbcc-examples linux-headers-$(uname -r)

# -a 表示显示每个内存分配请求的大小以及地址

# -p 指定案例应用的 PID 号

/usr/share/bcc/tools/memleak -a -p $(pidof app)

Swap

# 开关swap

swapoff -a

swapon -a

# 按 VmSwap 使用量对进程排序，输出进程名称、进程 ID 以及 SWAP 用量

for file in /proc/\*/status ; do awk '/VmSwap|Name|^Pid/{printf $2 " " $3}END{ print ""}' $file; done | sort -k 3 -n -r | head

禁止swap

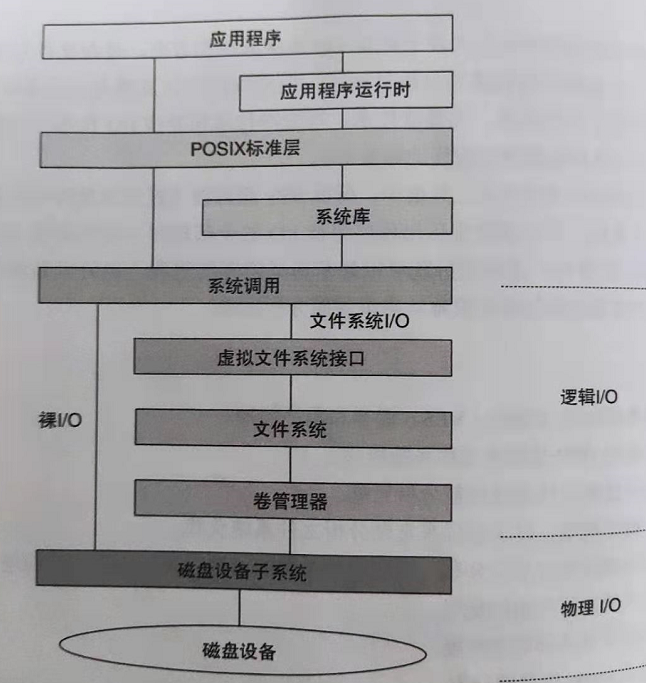
如果实在需要用到 Swap，可以尝试降低 swappiness 的值，减少内存回收时 Swap 的使用倾向

cat /proc/sys/vm/swappiness

响应延迟敏感的应用，如果它们可能在开启 Swap 的服务器中运行，你还可以用库函数 mlock() 或者 mlockall() 锁定内存，阻止它们的内存换出。

# 文件系统

## 背景知识



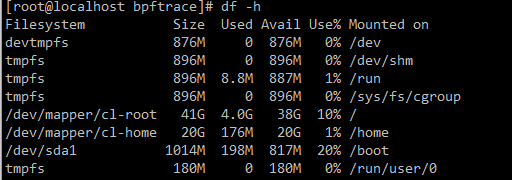
## 分析策略

1. 系统挂在文件系统df/mount
2. 历史容量，超过90%，可能有问题
3. opensnoop/filelife/ext4dist/ext4slower/cachestate/vfsstat

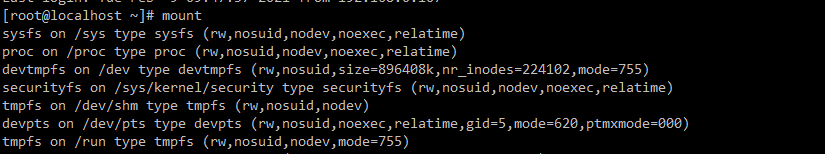
## 文件系统分析

### df

df -h



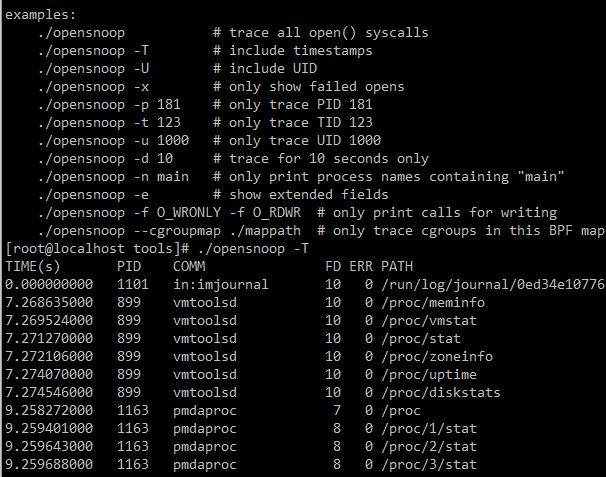
### mount



noatime

### perf trace

### opensnoop



### statsnoop

### syncsnoop

### mmapfiles

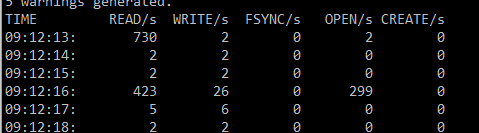
### scread

### fmapfault

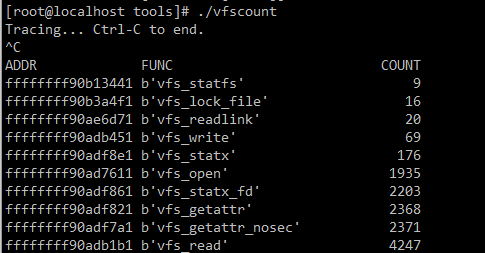
### filelife

短期文件

### vfsstat



### vfscount

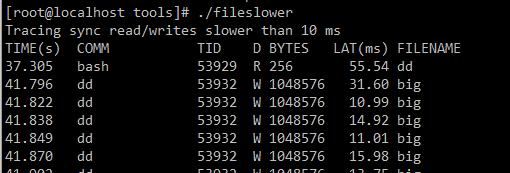


### vfssize

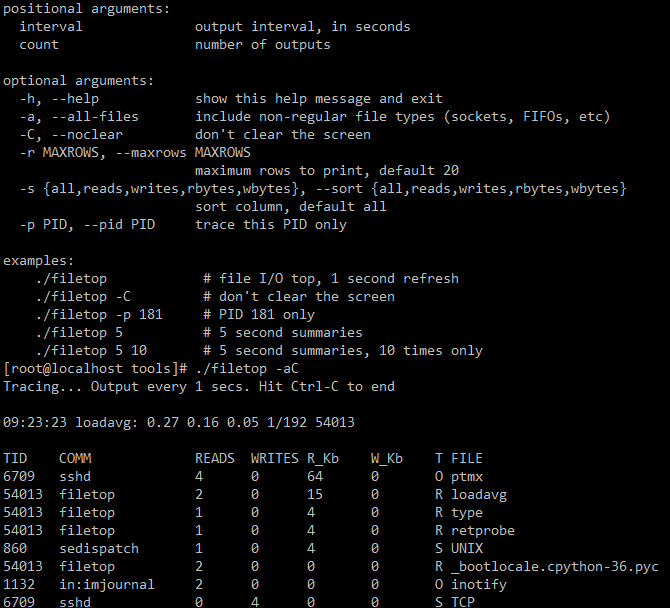
以直方图显示vfs读写大小

### fsrwstat

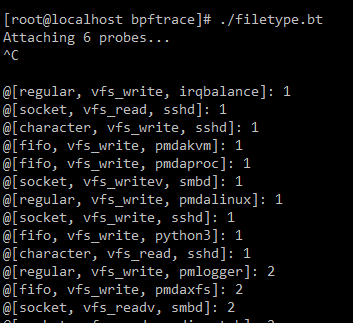
### fileslower



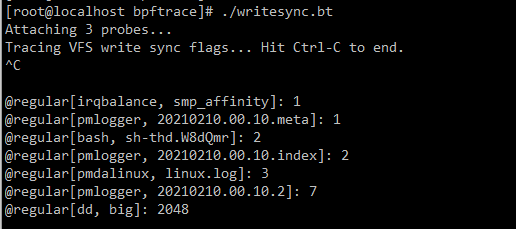
### filetop



### filetype

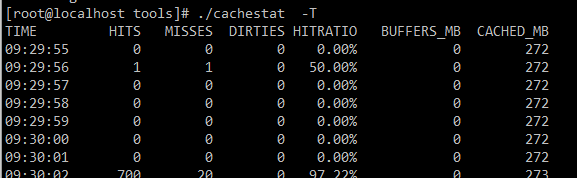


### writesync



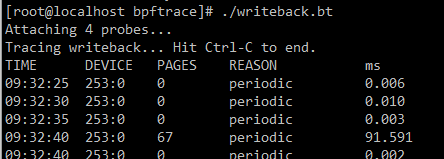
### cachestat

页缓存命中和有效程度



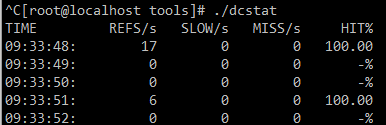
echo 3 > /proc/sys/vm/drop\_caches

### writeback

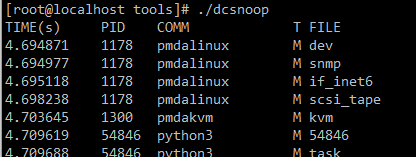


### dcstat

目录缓存统计信息



### dcsnoop



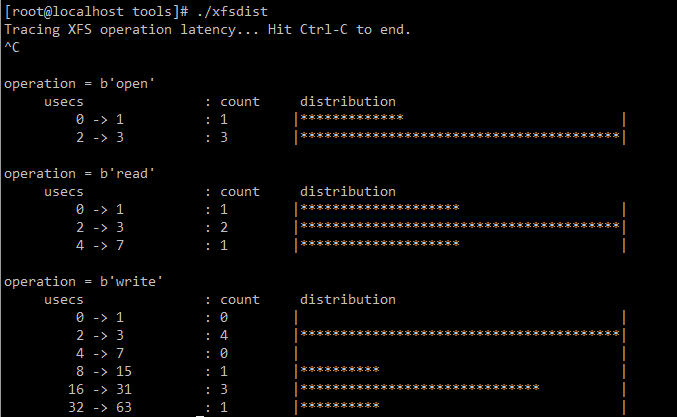
R: 查找

M: MISS

### mountsnoop

### xfsslower

### xfsdist



### icstat

inode缓存查找

### bufgrow

页缓存增长情况

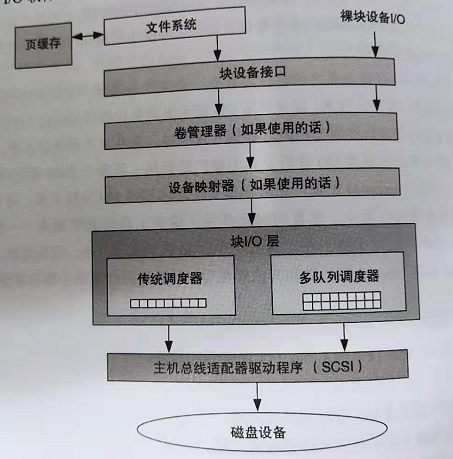
# IO子系统

## 背景知识

### 磁盘性能基准数据



### 块IO软件栈



## 分析策略

Iostat

基准测试

## 基准测试

sync && sudo echo 3 | sudo tee /proc/sys/vm/drop\_caches && sync

### FIO基准测试

# 随机读

fio -name=randread -direct=1 -iodepth=64 -rw=randread -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

# 随机写

fio -name=randwrite -direct=1 -iodepth=64 -rw=randwrite -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

# 顺序读

fio -name=read -direct=1 -iodepth=64 -rw=read -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

# 顺序写

fio -name=write -direct=1 -iodepth=64 -rw=write -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

### 回放IO

# 使用 blktrace 跟踪磁盘 I/O，注意指定应用程序正在操作的磁盘

$ blktrace /dev/sdb

# 查看 blktrace 记录的结果

# ls

sdb.blktrace.0 sdb.blktrace.1

# 将结果转化为二进制文件

$ blkparse sdb -d sdb.bin

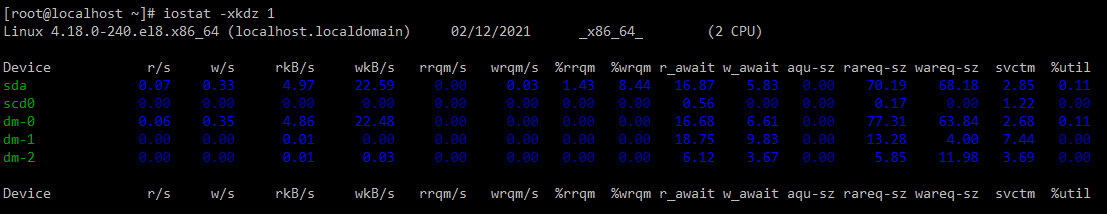
# 使用 fio 重放日志

$ fio --name=replay --filename=/dev/sdb --direct=1 --read\_iolog=sdb.bin

## IO分析

### iostat

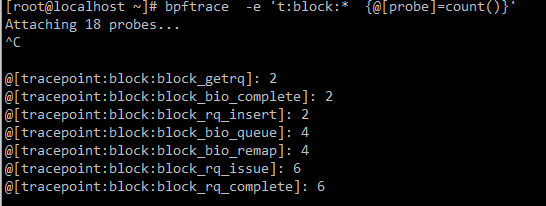
iostat -xkdz 1



### 单行程序

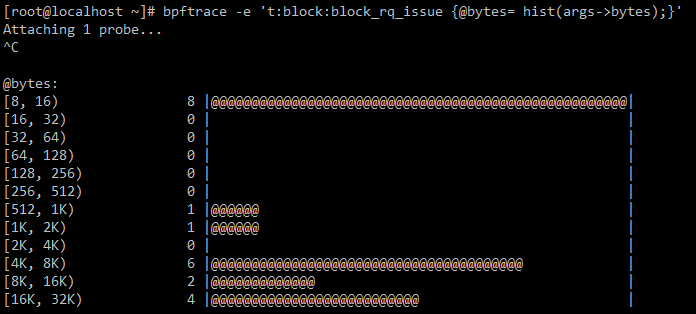
#### 块IO统计

bpftrace -e 't:block:\* {@[probe]=count()}'



#### 直方图统计块IO尺寸

bpftrace -e 't:block:block\_rq\_issue {@bytes= hist(args->bytes);}'

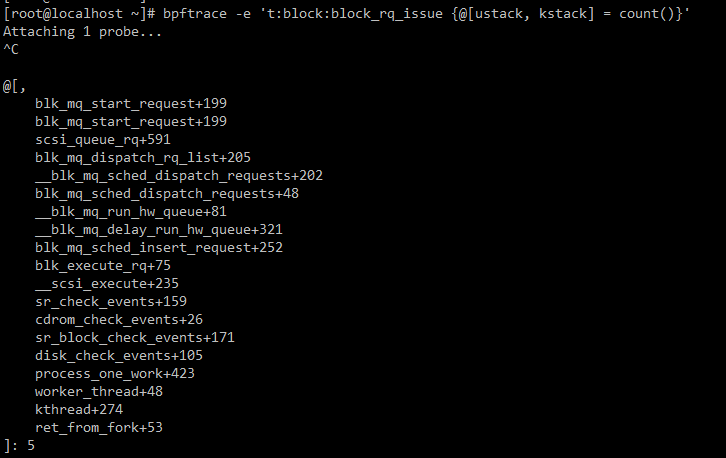


#### 块IO用户态调用栈

bpftrace -e 't:block:block\_rq\_issue {@[ustack] = count()}'

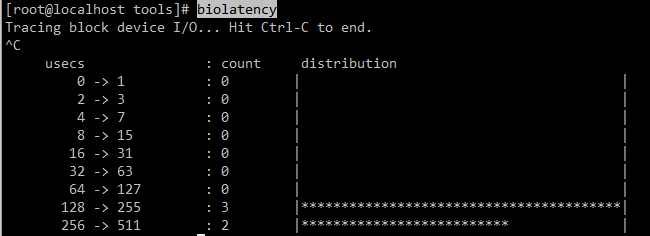
bpftrace -e 't:block:block\_rq\_issue {@[kstack] = count()}'

bpftrace -e 't:block:block\_rq\_issue {@[ustack, kstack] = count()}'



#### 直方图统计块IO尺寸

### biolatency



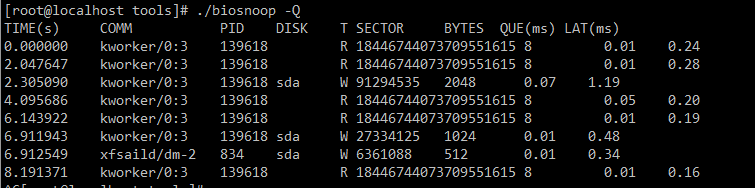
./biolatency -Fm

./biolatency -D

./biolatency -Q

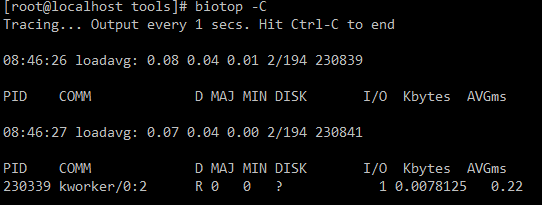
### biosnoop

biosnoop -Q



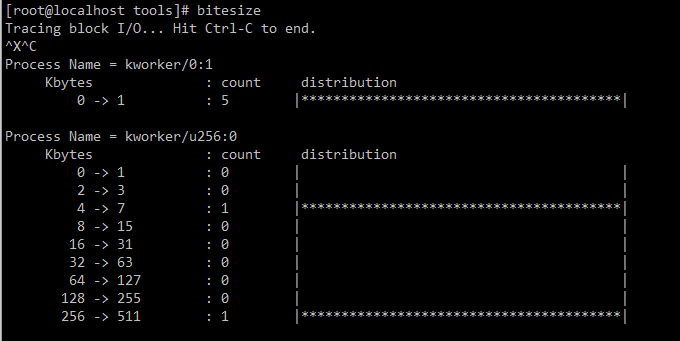
### biotop

biotop -C



### biteszie

顺序IO 128K性能可能会比较好。可以通过fio测试验证。

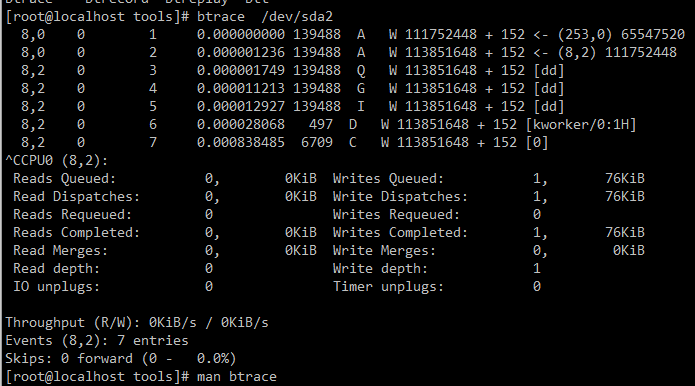


### biopattern

识别IO模型：随机或者顺序

### btrace

btrace - perform live tracing for block devices



### bioerr

bpftrace -e 't:block:block\_rq\_issue /args->dev==0/ {@[kstack]=count()}'

### mdflush

### iosched

# todo

## 跟踪进程、线程系统调用

strace -p 3387 -f 2>&1 | grep write

## FIO基准测试

# 随机读

fio -name=randread -direct=1 -iodepth=64 -rw=randread -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

# 随机写

fio -name=randwrite -direct=1 -iodepth=64 -rw=randwrite -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

# 顺序读

fio -name=read -direct=1 -iodepth=64 -rw=read -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

# 顺序写

fio -name=write -direct=1 -iodepth=64 -rw=write -ioengine=libaio -bs=4k -size=1G -numjobs=1 -runtime=1000 -group\_reporting -filename=/dev/sdb

## 回放IO

# 使用 blktrace 跟踪磁盘 I/O，注意指定应用程序正在操作的磁盘

$ blktrace /dev/sdb

# 查看 blktrace 记录的结果

# ls

sdb.blktrace.0 sdb.blktrace.1

# 将结果转化为二进制文件

$ blkparse sdb -d sdb.bin

# 使用 fio 重放日志

$ fio --name=replay --filename=/dev/sdb --direct=1 --read\_iolog=sdb.bin

# 网络系统

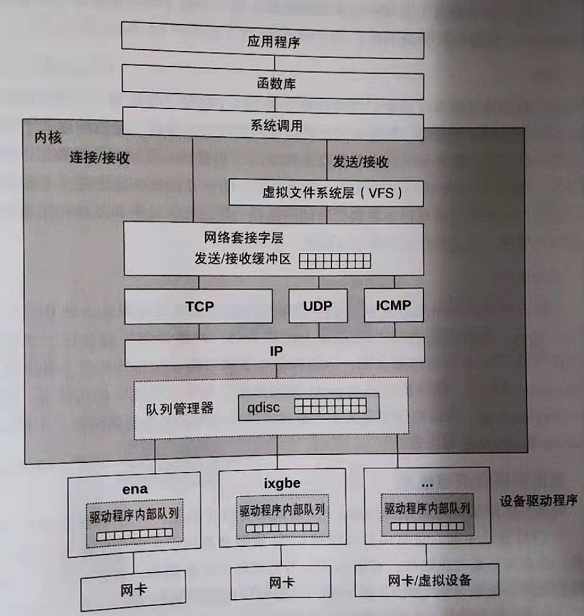
## 背景知识

### 端到端传说延时量级

光纤100公里约1毫秒



### 网络软件栈



## 测试网络性能工具

### iperf

# server

iperf -s

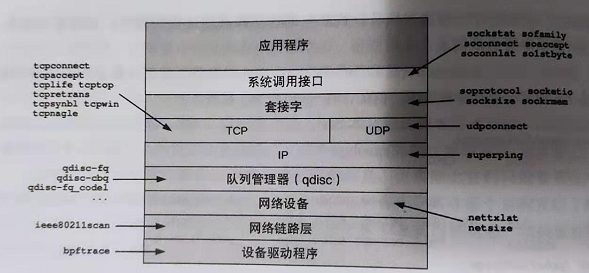
# client （2秒report，并发5线程）

iperf -c 192.168.1.61  -i 2 -P 5

iperf -c 192.168.1.61  -i 2 -t 60

## 分析策略

## 网络分析



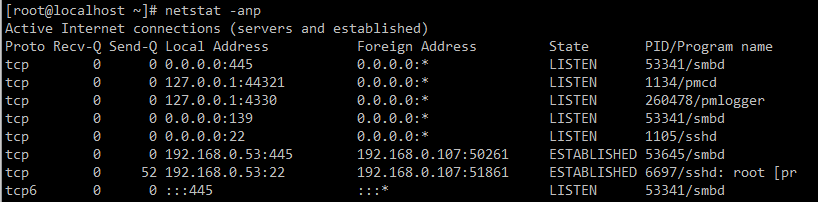
### sar

网卡收发包

sar -n DEV 1

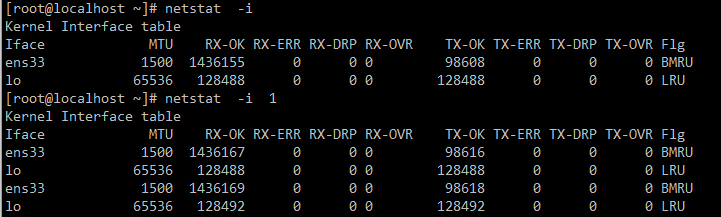
### netstat

netstat -anp



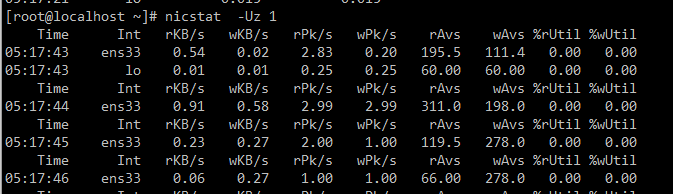
netstat -i

netstat -i 1



### nicstat

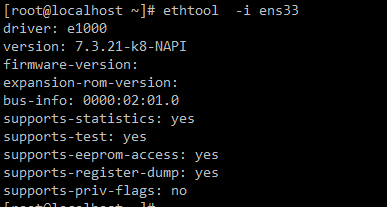
nicstat -Uz 1



<https://rpmfind.net/linux/fedora/linux/releases/33/Everything/x86_64/os/Packages/n/nicstat-1.95-14.fc33.x86_64.rpm>

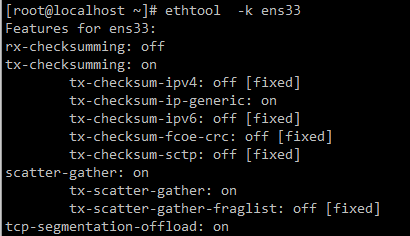
### ethtool

ethtool -i ens33



# 静态配置

ethtool -k ens33



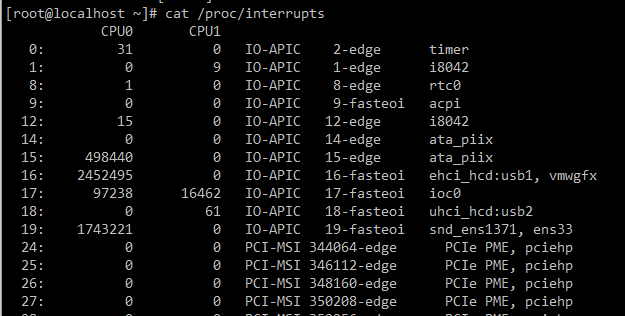
ethtool -K 调节开关

### tcpdump

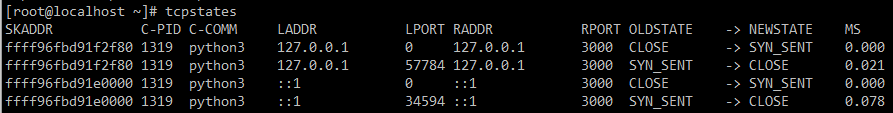
### 中断

观察中断分布情况

cat /proc/interrupts

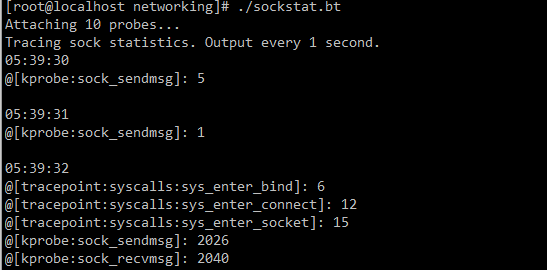


### tcpstates



### sockstat

每秒套接字调用次数， 进行负载画像分析，作为下一步分析的起点。

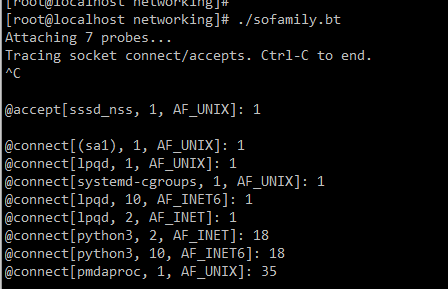


bpftrace -e 'kprobe:sock\_sendmsg { @[comm] = count(); }'

bpftrace -e 'kprobe:sock\_recvmsg { @[comm] = count(); }'

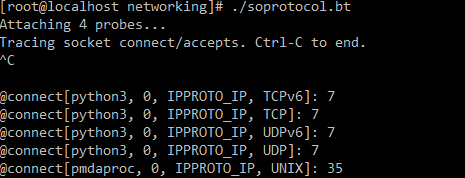
### sofamily

按照 进程和协议类型 跟踪accept和connect系统调用跟踪新的套接字连接，进行负载画像

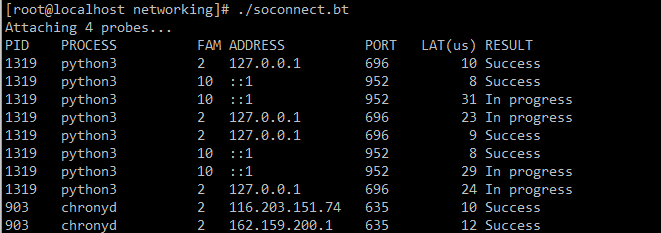


### soprotocol

按照 进程和传输协议号 跟踪accept和connect系统调用 跟踪 新的套接字连接，进行负载画像



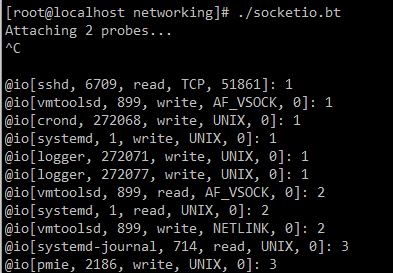
### soconnect



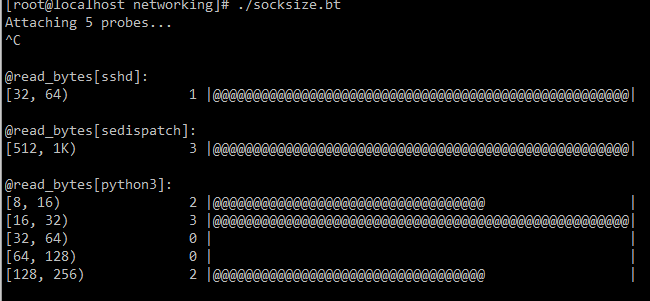
### soaccept

### socketio

socket IO次数



### socksize



### tcplife

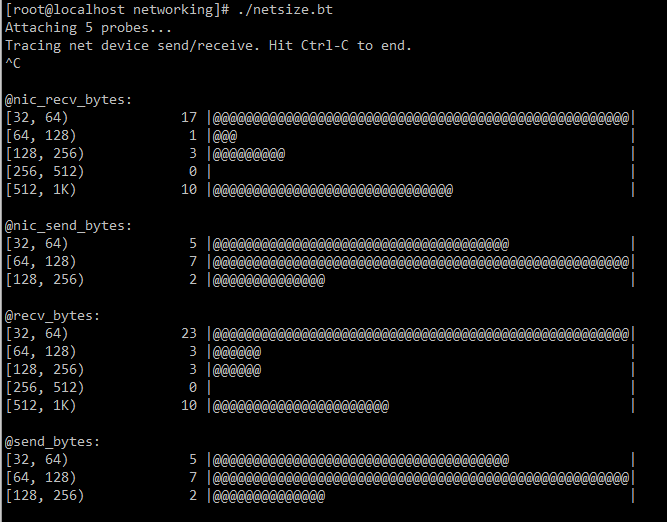
### tcptop

### tcpretrans

### tcpsynbl

### gethostlatency

### netsize



### 单行 bpftrace 程序

bpftrace -e 'kprobe:sock\_sendmsg { @[comm] = count(); }'

bpftrace -e 't:syscalls:sys\_enter\_accept\* { @[pid, comm] = count(); }'

bpftrace -e 't:syscalls:sys\_enter\_connect { @[pid, comm] = count(); }'

bpftrace -e 'k:sock\_sendmsg,k:sock\_recvmsg { @[probe, pid, comm] = count(); }'

bpftrace -e 'kr:sock\_sendmsg,kr:sock\_recvmsg /(int32)retval > 0/ { @[probe, pid, comm] = sum((int32)retval); }'

bpftrace -e 'k:sock\_sendmsg,k:sock\*recvmsg { @[probe, comm, pid] = count(); }'

bpftrace -e 'k:tcp\_\* { @[probe] = count(); }'

bpftrace -e 'k:tcp\_\* { @[probe, pid, comm] = count(); }'

bpftrace -e 'k:tcp\_sendmsg { @send\_bytes = hist(arg2); }'

bpftrace -e 'kr:tcp\_recvmsg /retval > 0/ { @recv\_bytes = hist(retval); }'

bpftrace -e 'k:udp\_sendmsg { @send\_bytes = hist(arg2); }'

bpftrace -e 'kr:udp\_recvmsg /retval > 0/ { @recv\_bytes = hist(retval); }'

## 网卡

# 查看接口收发包统计，是否丢包

ifconfig ethx

while ((1)) ; do ifconfig | grep ethx -A 10 | grep dropped ; sleep 3; done;

watch -d ifconfig

# 非常好的网卡优化

https://cromwell-intl.com/open-source/performance-tuning/ethernet.htm

## DNS

# 查看dns解析用时

time nslookup [www.baidu.com](http://www.baidu.com)

nslookup -debug time.geekbang.org

# +trace 表示开启跟踪查询

# +nodnssec 表示禁止 DNS 安全扩展

dig +trace +nodnssec time.geekbang.org

### Dnsmsq

echo nameserver 127.0.0.1 > /etc/resolv.conf

apt-get install dnsmasq

systemctl restart dnsmasq

# -w 表示只输出 HTTP 状态码及总时间，-o 表示将响应重定向到 /dev/null

curl -s -w 'Http code: %{http\_code}\nTotal time:%{time\_total}s\n' -o /dev/null http://192.168.0.30/

...

Http code: 200

Total time:0.002s

## 缓解DDos攻击

### 模拟DDos 攻击

hpnig3 发SYNC flood

# -S 参数表示设置 TCP 协议的 SYN（同步序列号），-p 表示目的端口为 80

# -i u10 表示每隔 10 微秒发送一个网络帧

hping3 -S -p 80 -i u10 192.168.0.30

### 分析

# 查看套接字状态

netstat -anp | more

# 抓包分析

tcpdump -i eth0 -n tcp port 80

### 缓解方案

# 缓解 syn flood 攻击

cat /etc/sysctl.conf

net.ipv4.tcp\_syncookies = 1

net.ipv4.tcp\_synack\_retries = 1

net.ipv4.tcp\_max\_syn\_backlog = 1024

## 网络延时

### HTTP基准测试工具

git clone https://github.com/wg/wrk

cd wrk

apt-get install build-essential -y

make

sudo cp wrk /usr/local/bin/

### 分析过程

#### 测试网络RTT

# 测试网络RTT

hping3 -c 3 -S -p 80 baidu.com

# --tcp 表示使用 TCP 协议，-p 表示端口号，-n 表示不对结果中的 IP 地址执行反向域名解析

traceroute --tcp -p 80 -n baidu.com

#### **测试 80 端口性能**

wrk --latency -c 100 -t 2 --timeout 2 <http://192.168.0.30/>

#### 抓包分析

tcpdump -nn tcp port 80 -w nginx.pcap

#### 理论知识

TCP\_QUICKACK：禁止延迟确认，延迟确认最小时间间隔40ms。

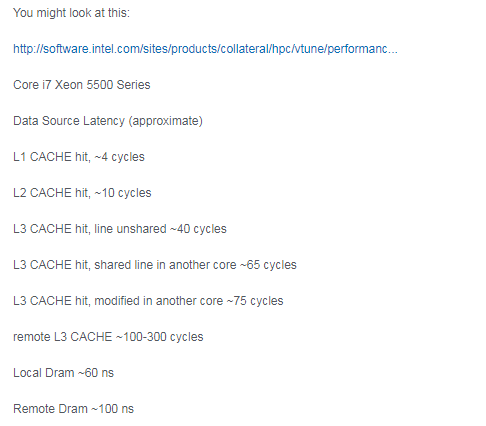
设置TCP\_NODELAY选项，禁止Nagle算法。Nagle 算法通过合并 TCP 小包，提高网络带宽的利用率。

# CPU

## 数据访问时间

### 数据访问指令周期

L3 CACHE命中指令周期



上图出处：

https://software.intel.com/en-us/forums/intel-manycore-testing-lab/topic/287236

### 数据访问时间

0.5 ns - CPU L1 dCACHE reference

1 ns - speed-of-light (a photon) travel a 1 ft (30.5cm) distance

5 ns - CPU L1 iCACHE Branch mispredict

7 ns - CPU L2 CACHE reference

71 ns - CPU cross-QPI/NUMA best case on XEON E5-46\*

100 ns - MUTEX lock/unlock

100 ns - own DDR MEMORY reference

135 ns - CPU cross-QPI/NUMA best case on XEON E7-\*

202 ns - CPU cross-QPI/NUMA worst case on XEON E7-\*

325 ns - CPU cross-QPI/NUMA worst case on XEON E5-46\*

10,000 ns - Compress 1K bytes with Zippy PROCESS

20,000 ns - Send 2K bytes over 1 Gbps NETWORK

250,000 ns - Read 1 MB sequentially from MEMORY

500,000 ns - Round trip within a same DataCenter

10,000,000 ns - DISK seek

10,000,000 ns - Read 1 MB sequentially from NETWORK

30,000,000 ns - Read 1 MB sequentially from DISK

150,000,000 ns - Send a NETWORK packet CA -> Netherlands

| | | |

| | | ns|

| | us|

| ms|

出处：

<https://stackoverflow.com/questions/4087280/approximate-cost-to-access-various-caches-and-main-memory>

# Perf

<http://www.brendangregg.com/perf.html>

<https://www.ibm.com/developerworks/cn/linux/l-cn-perf1/>

<https://www.cnblogs.com/arnoldlu/p/6241297.html>

查看系统概况

perf top

perf stat -p PID sleep 10

指定进程

perf top -g -p PID

指定事件perf

# -e 指定perf事件

perf top -g -e cache-misses -p PID

# perf record默认将统计信息写入perf.data文件

perf record -g -p PID sleep 10

perf report

perf report -i perf.data

### perf生成火焰图

git clone https://github.com/brendangregg/FlameGraph.git

perf record -e cpu-clock -g -p 1841717 sleep 10

perf script -i perf.data &> perf.unfold

./FlameGraph/stackcollapse-perf.pl perf.unfold > perf.fold

./FlameGraph/flamegraph.pl perf.fold > perf.svg

参考：

<https://www.cnblogs.com/happyliu/p/6142929.html>

### 汇编到函数查看

o 显示汇编地址

可以通过地址查到函数行号

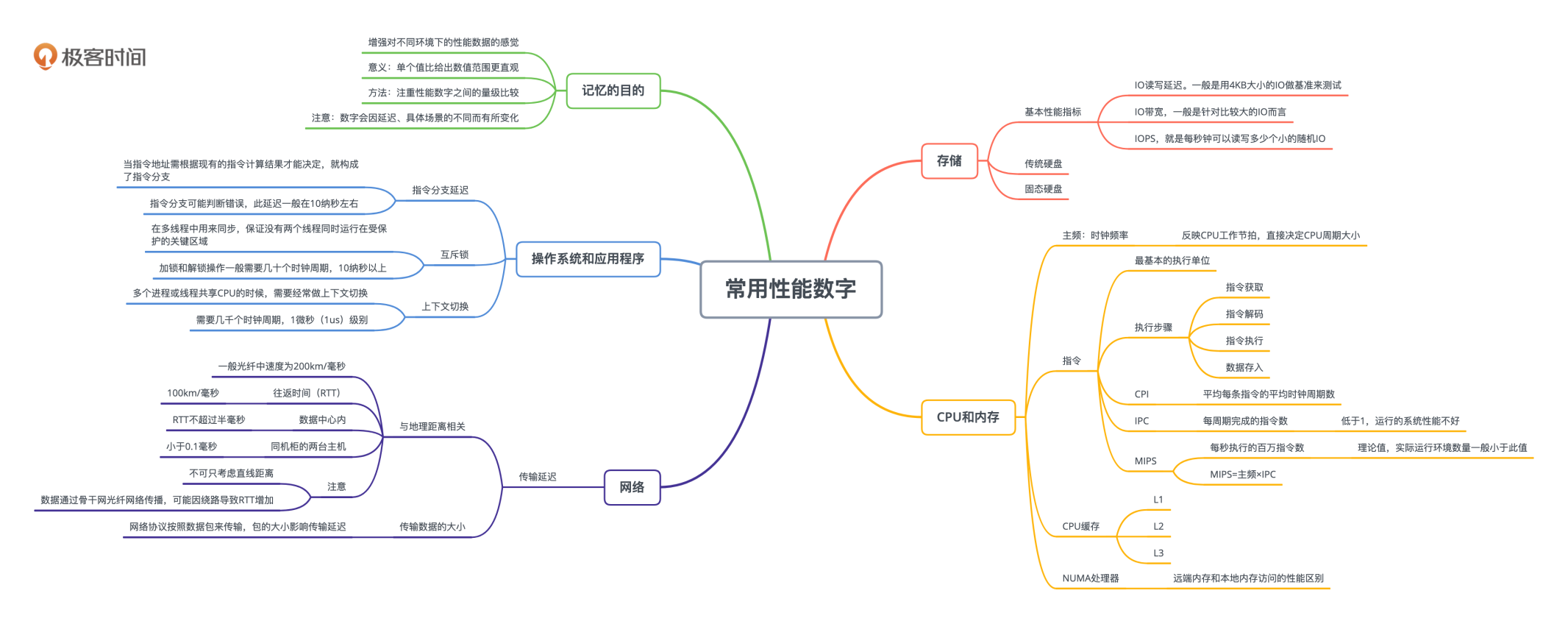
# -e 可执行文件

# -a 地址

addr2line -e a.out -a XXXXX

addr2line -e a.out -a 51a8b8

# 常用性能数字



下面数字的来源

https://colin-scott.github.io/personal\_website/research/interactive\_latency.html

