Linux性能优化的工具集和方法论

邓钫元-大数据架构

2018.4.19

关于我

- 13年浙大软件工程毕业
- 13-14年 百度商业平台部-风控平台研发
- 15年至今 链家大数据集群及基础引擎建设

- 专注于hadoop生态组件， 热爱开源，为社区贡献多个patch
- 丰富的性能调优经验

引言

“为啥我的程序卡住了，它在做啥？”

“磁盘卡死了，啥程序占用的？”

“该不该换成ssd？”

我们常遇到各种性能问题, 该如何定位问题，解决问题。

本课程主要介绍系统性能的度量指标，分析思路，及相关工具集。

目录

- 系统性能指标和观测方法
- 常用命令及平台-分析系统负载
- 动态追踪-了解程序在做什么
- 实战案例
- Q&A

性能问题是充满挑战的

性能是主观的

磁盘平均io响应时间是10ms, 好或坏？

取决于业务需求及程序热点

系统是复杂的

子系统相互关联，甚至有连锁故障

运行环境不一（硬件/软件）

可能多问题并存

量化数据

控制变量法

观测视角

资源分析（自下而上）

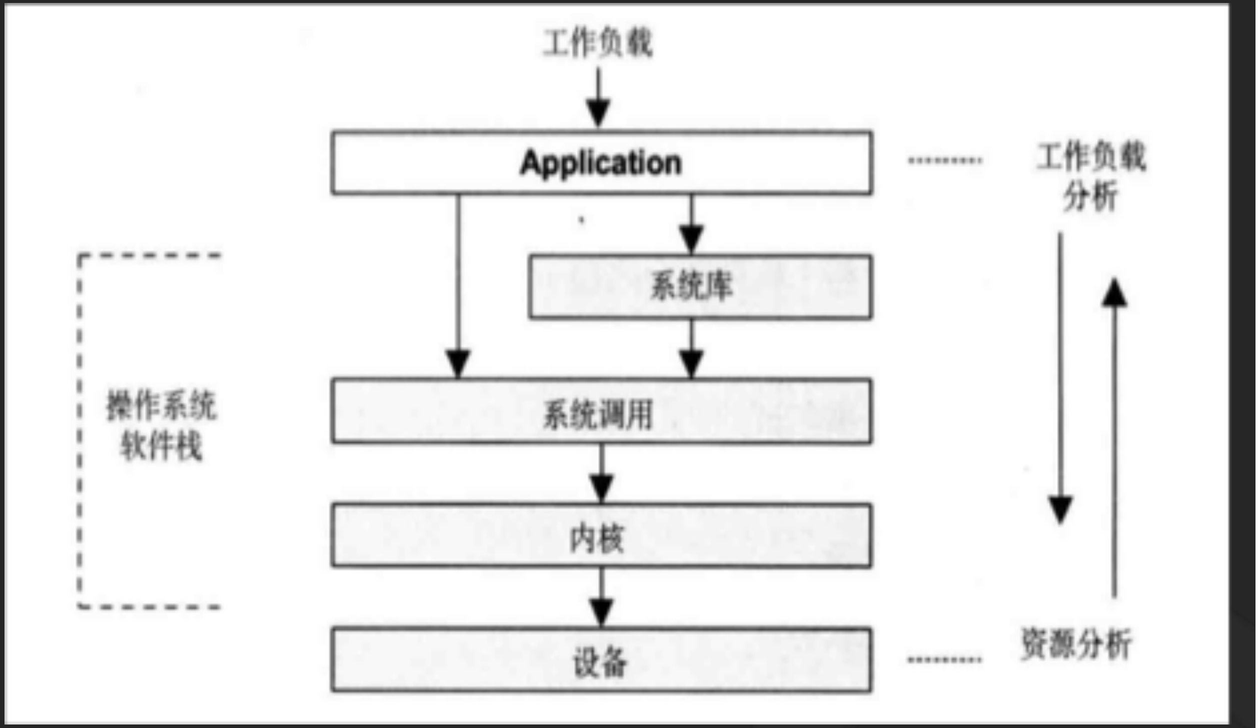
从系统的资源指标开始

更通用，适合资源被打满的情况

工作负载分析（自上而下）

从应用的metrics和stack开始

贴近代码逻辑，适合并发锁问题



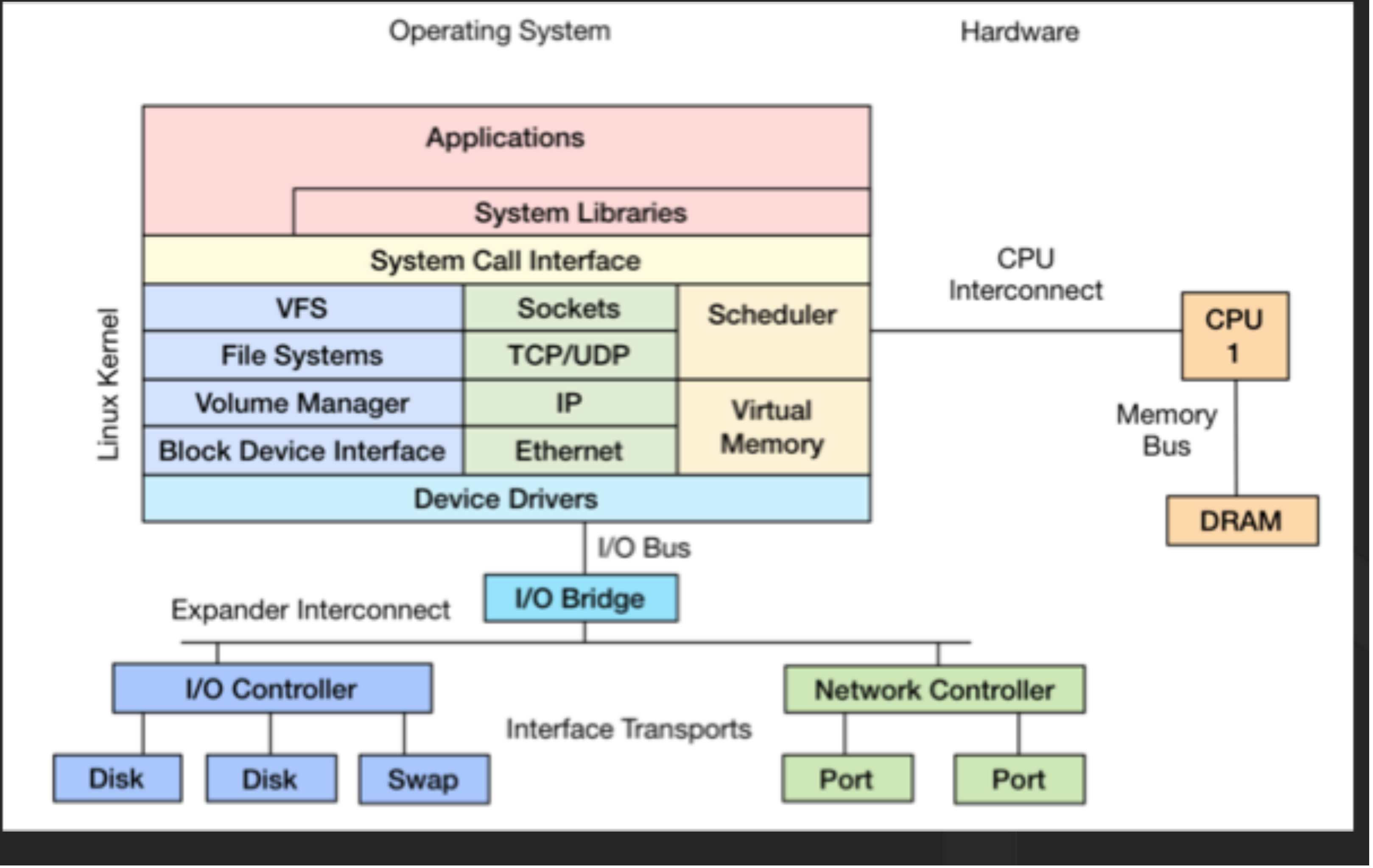
系统资源有哪些

硬件资源

- CPU
- 内存
- 磁盘
- 网络

软件资源

- 软件锁
- 线程池/连接池



观测方法-USE方法

使用率（ Utilization ）

设备繁忙程度

工作时间/观测时间

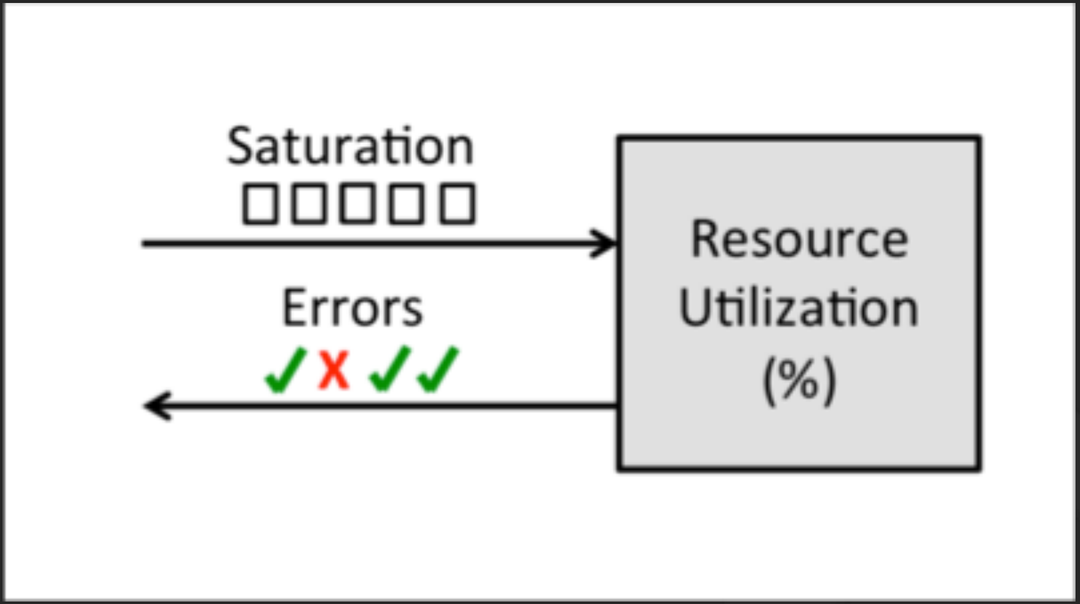
饱和度（ Saturation ）

队列长度，排队时间

超出设备处理能力的程度

错误率（ Errors ）

设备出错率



分析方法

问题

为什么A主机到B主机网络延迟很大？（事实：A和B在不同机架）

假设

A和B机架的交换机有故障

预测

A/B机架内互联通畅，A到另一机架的主机C通畅，A/B各机架各换一台机器互联延迟

实验

分别测试以上场景

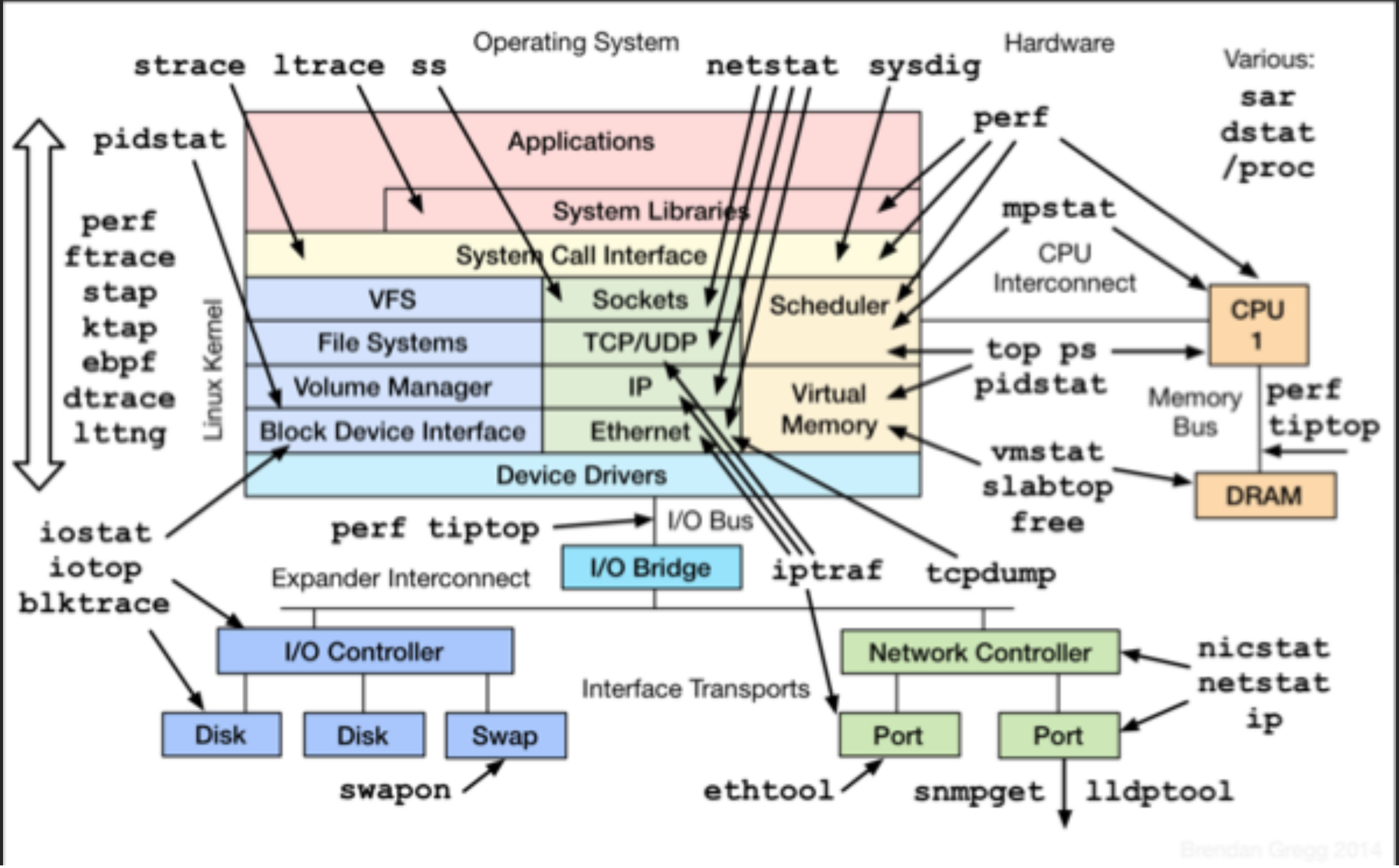
分析

结果与预测相同，说明问题出在A和B的交换机有故障

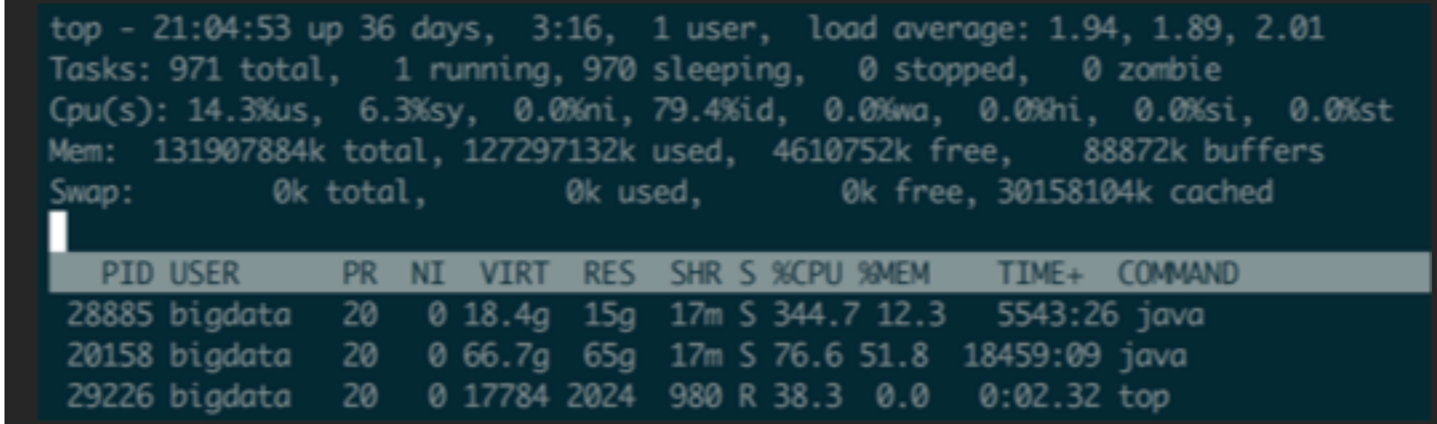
目录

- 系统性能指标和观测方法
- 常用命令及平台-分析系统负载
- 动态追踪-了解程序在做什么
- 实战案例
- Q&A

眼花缭乱的命令


Operating System
Hardware
Various:
sar
dstat
/proc
strace
ltrace
ss
netstat
sysdig
pidstat
perf
perf
ftrace
stap
ktap
ebpf
dtrace
lttng
Linux Kernel
Applications
System Libraries
System Call Interface
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Volume Manager
IP
Virtual Memory
Block Device Interface
Ethernet
Device Drivers
mpstat
CPU Interconnect
CPU 1
top ps
pidstat
Memory Bus
perf
tiptop
vmstat
slabtop
free
DRAM
iostat
iotop
blktrace
perf tiptop
I/O Bus
I/O Bridge
iptraf
tcpdump
Expander Interconnect
I/O Controller
Network Controller
nicstat
netstat
ip
Interface Transports
Disk
Disk
Swap
Port
Port
swapon
ethtool
snmpget
lldptool
Brendan Gregg 2014

cpu分析



```
top - 21:04:53 up 36 days,  3:16,  1 user,  load average: 1.94, 1.89, 2.01
Tasks: 971 total,   1 running, 970 sleeping,   0 stopped,   0 zombie
Cpu(s): 14.3%us,  6.3%sy,  0.0%ni, 79.4%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  131907884k total, 127297132k used,  4610752k free,    88872k buffers
Swap:        0k total,        0k used,        0k free, 30158104k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
28885 bigdata   20   0 18.4g  15g  17m S 344.7 12.3   5543:26 java
20158 bigdata   20   0 66.7g  65g  17m S 76.6 51.8  18459:09 java
29226 bigdata   20   0 17784 2024  980 R 38.3  0.0   0:02.32 top
```

user/sys/iowait-占用

Load-饱和度

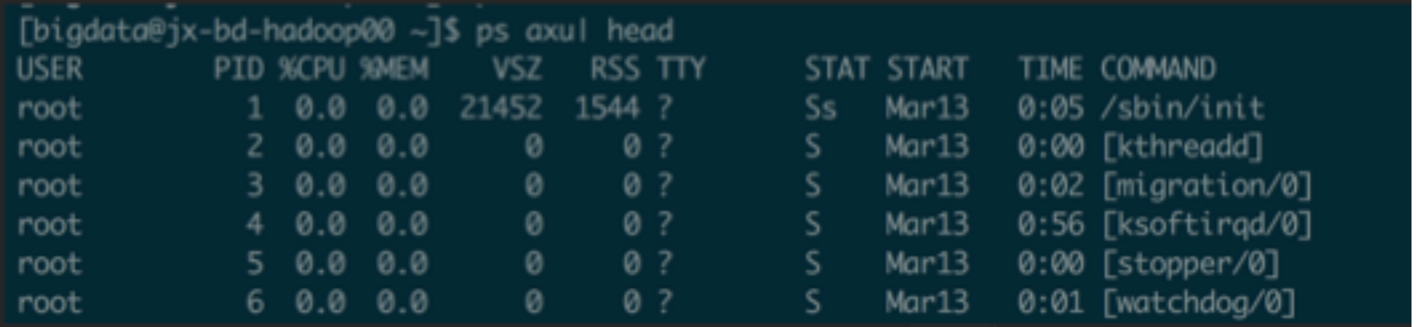
%CPU>100%?

Top -p 进程号 –H 查询线程占用

进程状态R、S、D、T、Z

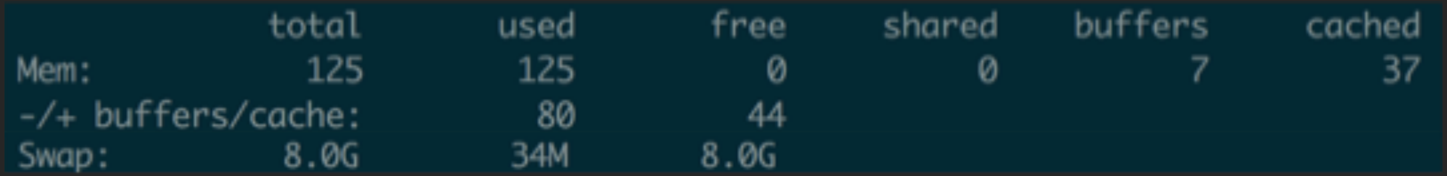
ps aux H(线程) f(进程数)

借助sort找到占用最大



```
[bigdata@jx-bd-hadoop00 ~]$ ps axu| head
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  21452  1544 ?        Ss   Mar13   0:05 /sbin/init
root         2  0.0  0.0      0     0 ?        S    Mar13   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Mar13   0:02 [migration/0]
root         4  0.0  0.0      0     0 ?        S    Mar13   0:56 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S    Mar13   0:00 [stopper/0]
root         6  0.0  0.0      0     0 ?        S    Mar13   0:01 [watchdog/0]
```

内存分析



```
                   total       used       free     shared    buffers     cached
Mem:                 125        125          0          0          7         37
-/+ buffers/cache:               80         44
Swap:               8.0G        34M       8.0G
```

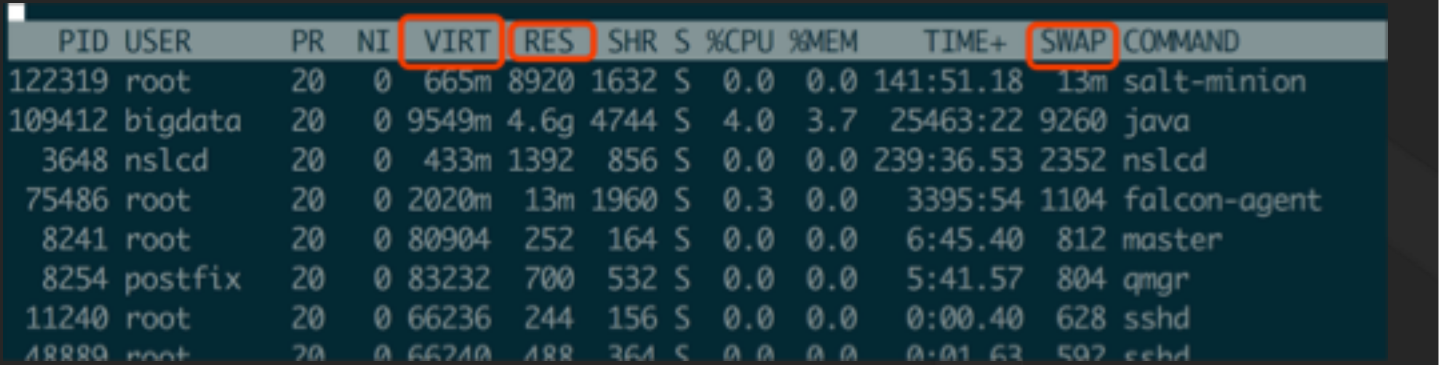
刚学车：free=0G, 内存不够

实习期：free=44G,内存很充足

老司机：cache被谁占用，命中率如何

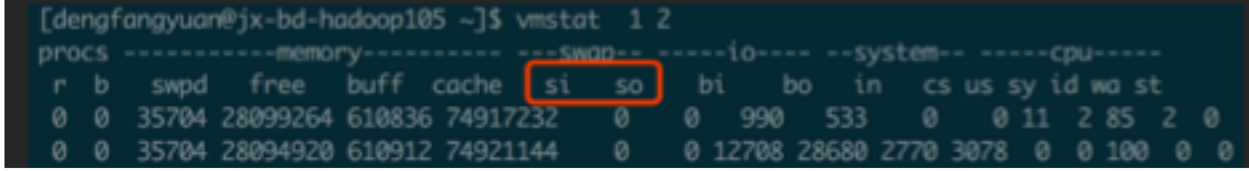
VIRT RES分别是啥

Top->按f选择swap->按O选择swap排序



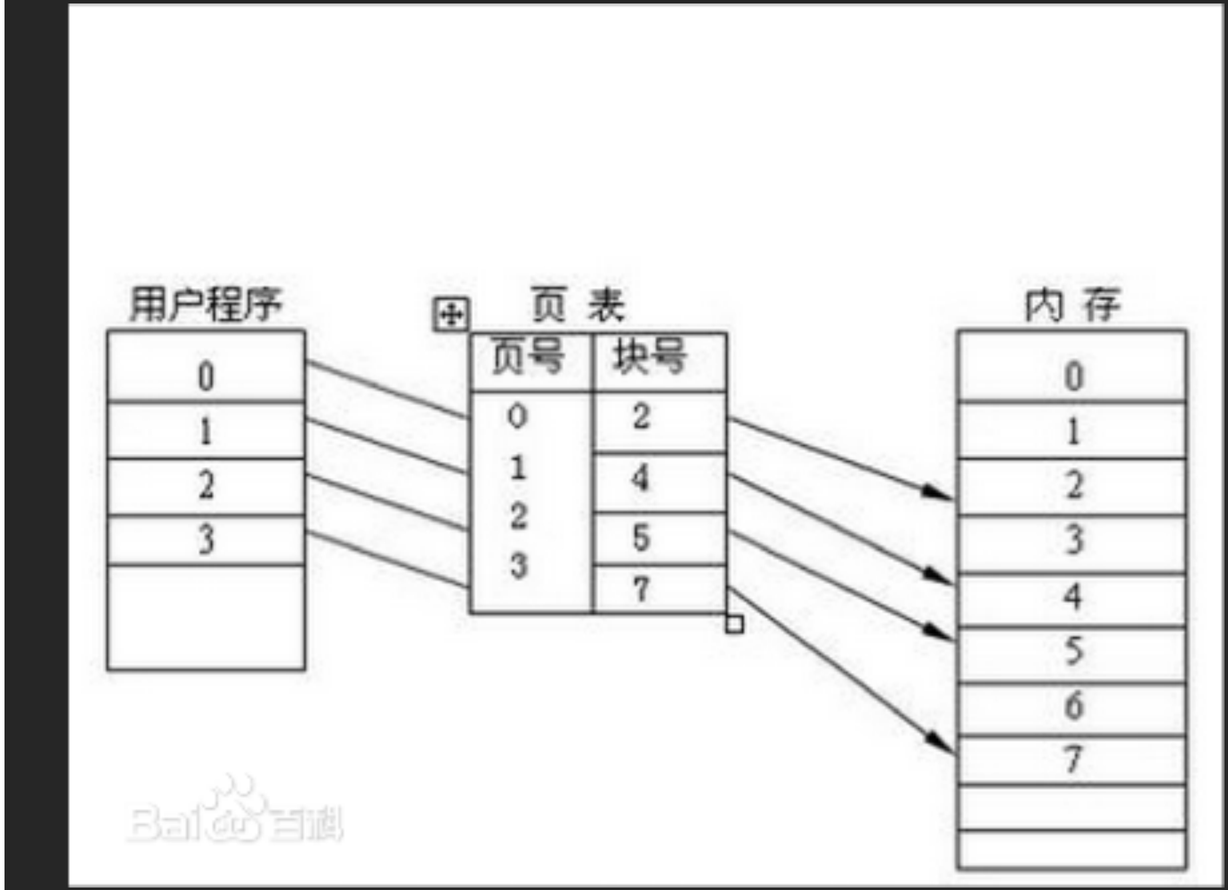
```
   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  SWAP COMMAND
122319 root      20   0  665m 8920 1632 S  0.0  0.0 141:51.18  13m salt-minion
109412 bigdata   20   0 9549m 4.6g 4744 S  4.0  3.7  25463:22 9260 java
  3648 nslcd     20   0  433m 1392  856 S  0.0  0.0 239:36.53 2352 nslcd
 75486 root      20   0 2020m  13m 1960 S  0.3  0.0   3395:54 1104 falcon-agent
  8241 root      20   0 80904  252  164 S  0.0  0.0   6:45.40  812 master
  8254 postfix   20   0 83232  700  532 S  0.0  0.0   5:41.57  804 qmgr
 11240 root      20   0 66236  244  156 S  0.0  0.0   0:00.40  628 sshd
 48889 root      20   0 66240  488  364 S  0.0  0.0   0:01.63  592 sshd
```

Vmstat查看swap速率



```
[dengfangyuan@jx-bd-hadoop105 ~]$ vmstat  1 2
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0  35704 28099264 610836 74917232    0    0   990   533    0    0 11  2 85  2  0
 0  0  35704 28094920 610912 74921144    0    0 12708 28680 2770 3078  0  0 100  0  0
```

简化的页表/虚拟内存



vm.overcommit_memory：

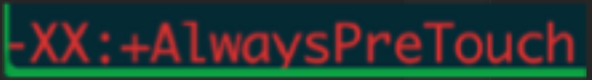
0： 不能超过剩余物理内存

1： 不做检查，直到发生OOM

2： RAM * vm.overcommit_ratio + Swap

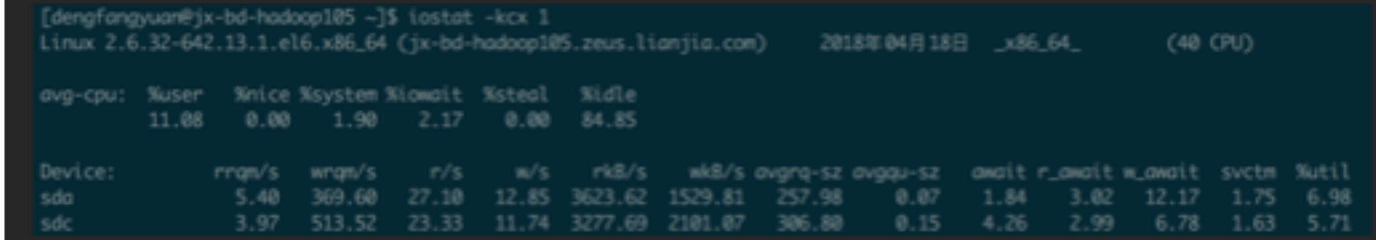
允许overcommit可以提高内存利用率

高实时场景下，避免缺页中断（及cache淘汰），jvm可设置



```
-XX:+AlwaysPreTouch
```

磁盘IO分析

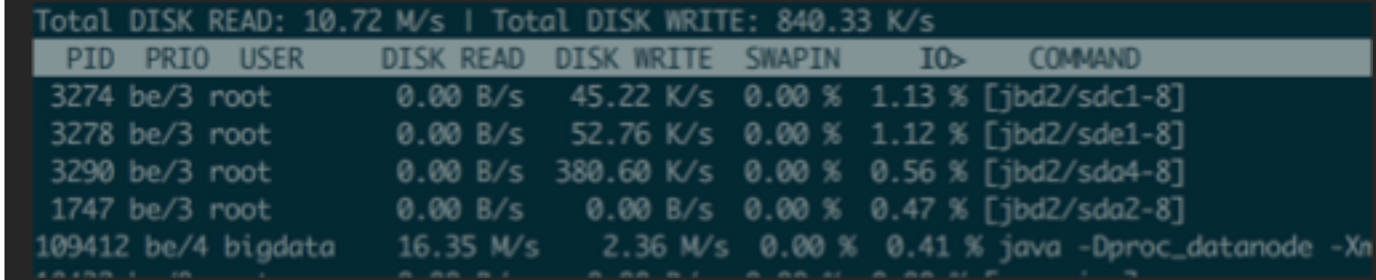


```
[dengfangyuan@jx-bd-hadoop105 ~]$ iostat -kcx 1
Linux 2.6.32-642.13.1.el6.x86_64 (jx-bd-hadoop105.zeus.lianjia.com)    2018年04月18日  _x86_64_        (40 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          11.08    0.00    1.90    2.17    0.00   84.85

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               5.40   369.60   27.10   12.85  3623.62  1529.81   257.98     0.07    1.84    3.02   12.17   1.75   6.98
sdc               3.97   513.52   23.33   11.74  3277.69  2101.07   306.80     0.15    4.26    2.99    6.78   1.63   5.71
```

Util使用率

Await/avgqu-sz 排队时间/队列长度

r/s rKB/s 读次数 读速度

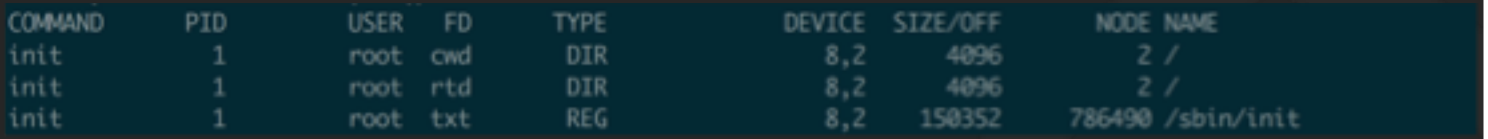


```
Total DISK READ: 10.72 M/s | Total DISK WRITE: 840.33 K/s
  PID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 3274 be/3 root        0.00 B/s   45.22 K/s  0.00 %  1.13 % [jbd2/sdc1-8]
 3278 be/3 root        0.00 B/s   52.76 K/s  0.00 %  1.12 % [jbd2/sde1-8]
 3290 be/3 root        0.00 B/s  380.60 K/s  0.00 %  0.56 % [jbd2/sda4-8]
 1747 be/3 root        0.00 B/s    0.00 B/s  0.00 %  0.47 % [jbd2/sda2-8]
109412 be/4 bigdata   16.35 M/s    2.36 M/s  0.00 %  0.41 % java -Dproc_datanode -Xm
```

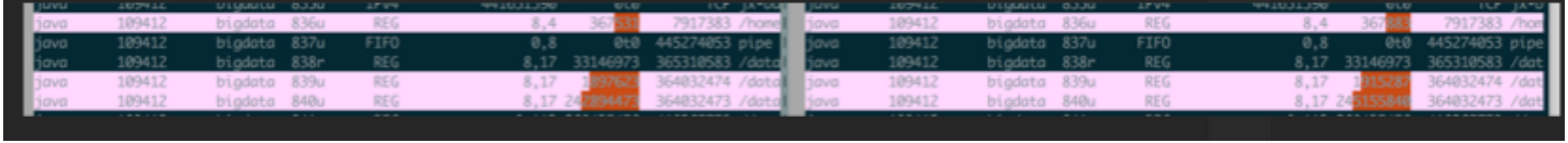
Iotop找出谁在占用磁盘

默认显示线程 –P显示进程

Lsof+diff 找出哪个文件被读写

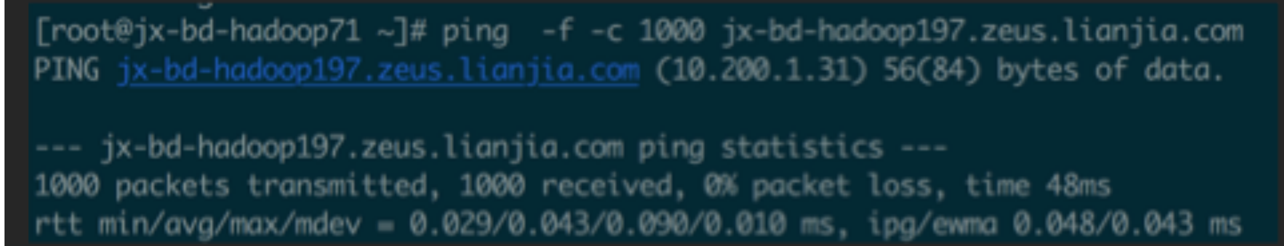


```
COMMAND     PID      USER   FD      TYPE             DEVICE  SIZE/OFF       NODE NAME
init          1      root  cwd       DIR                8,2      4096          2 /
init          1      root  rtd       DIR                8,2      4096          2 /
init          1      root  txt       REG                8,2    150352     786490 /sbin/init
```



```
java    109412   bigdata  836u   REG      8,4     367531   7917383 /home | java    109412   bigdata  836u   REG      8,4     367883   7917383 /hom
java    109412   bigdata  837u  FIFO      0,8        0t0 445274053 pipe  | java    109412   bigdata  837u  FIFO      0,8        0t0 445274053 pipe
java    109412   bigdata  838r   REG     8,17   33146973 365310583 /data | java    109412   bigdata  838r   REG     8,17   33146973 365310583 /dat
java    109412   bigdata  839u   REG     8,17    1897623 364032474 /data | java    109412   bigdata  839u   REG     8,17    1915287 364032474 /dat
java    109412   bigdata  840u   REG     8,17  242894473 364032473 /data | java    109412   bigdata  840u   REG     8,17  245155840 364032473 /dat
```

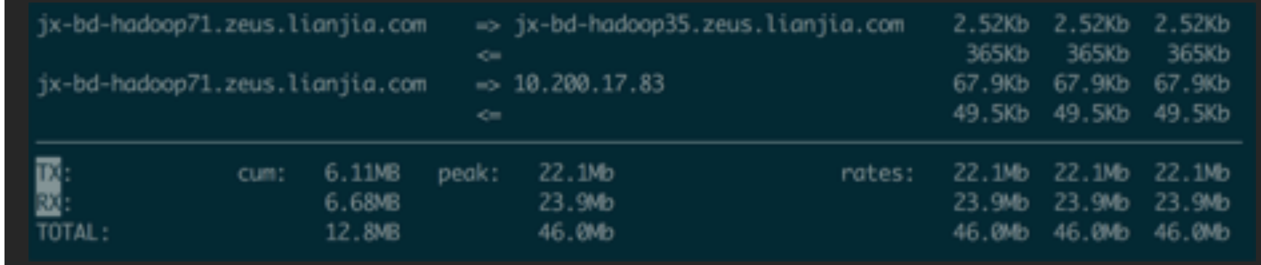
网络IO分析



```
[root@jx-bd-hadoop71 ~]# ping  -f -c 1000 jx-bd-hadoop197.zeus.lianjia.com
PING jx-bd-hadoop197.zeus.lianjia.com (10.200.1.31) 56(84) bytes of data.

--- jx-bd-hadoop197.zeus.lianjia.com ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time 48ms
rtt min/avg/max/mdev = 0.029/0.043/0.090/0.010 ms, ipg/ewma 0.048/0.043 ms
```

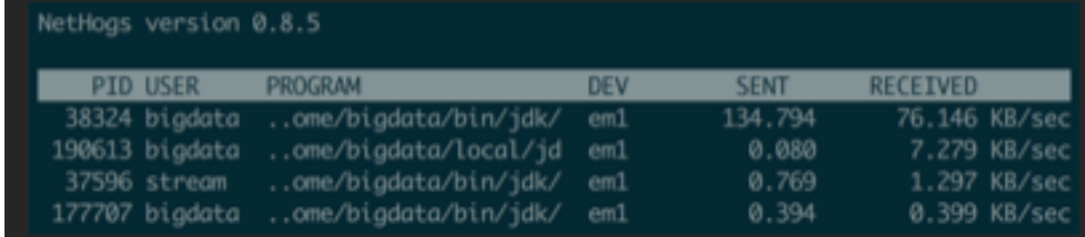
Ping查看网络延时

-f –c 10000 查看丢包率



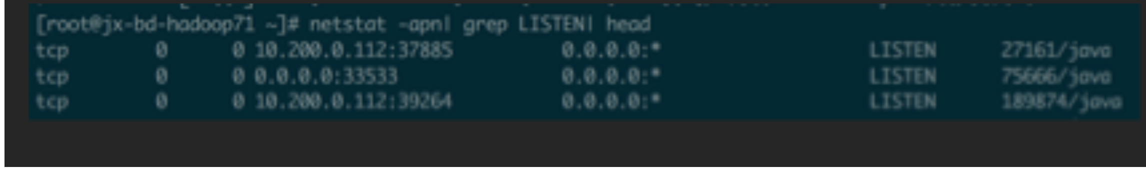
```
jx-bd-hadoop71.zeus.lianjia.com     => jx-bd-hadoop35.zeus.lianjia.com     2.52Kb  2.52Kb  2.52Kb
                                    <=                                      365Kb   365Kb   365Kb
jx-bd-hadoop71.zeus.lianjia.com     => 10.200.17.83                        67.9Kb  67.9Kb  67.9Kb
                                    <=                                     49.5Kb  49.5Kb  49.5Kb
------------------------------------------------------------------------------------------------
TX:             cum:   6.11MB   peak:   22.1Mb                  rates:     22.1Mb  22.1Mb  22.1Mb
RX:                    6.68MB           23.9Mb                             23.9Mb  23.9Mb  23.9Mb
TOTAL:                 12.8MB           46.0Mb                             46.0Mb  46.0Mb  46.0Mb
```

Iftop 显示整体网速/每个连接的网速



```
NetHogs version 0.8.5

    PID USER     PROGRAM                   DEV        SENT      RECEIVED
  38324 bigdata  ..ome/bigdata/bin/jdk/    em1     134.794    76.146 KB/sec
 190613 bigdata  ..ome/bigdata/local/jd    em1       0.080     7.279 KB/sec
  37596 stream   ..ome/bigdata/bin/jdk/    em1       0.769     1.297 KB/sec
 177707 bigdata  ..ome/bigdata/bin/jdk/    em1       0.394     0.399 KB/sec
```

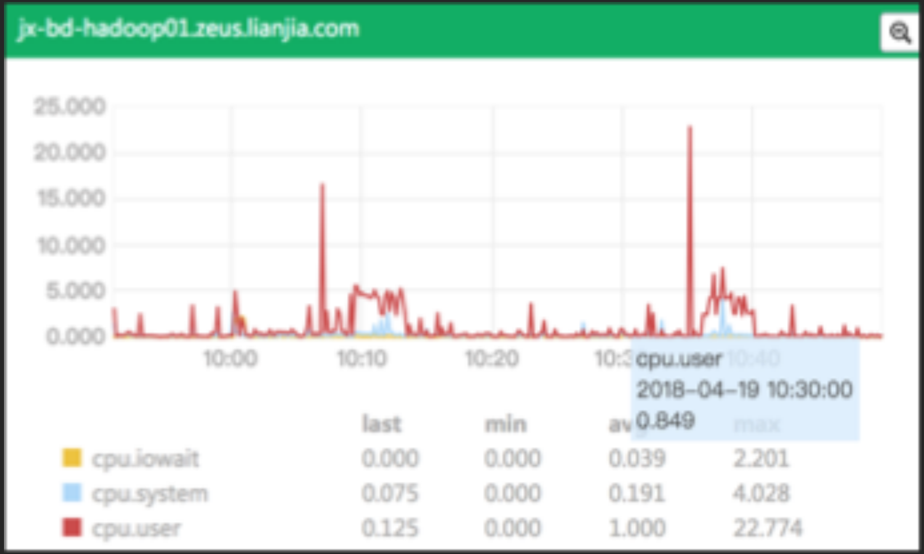
Nethogs找出占用网速的程序



```
[root@jx-bd-hadoop71 ~]# netstat -apn| grep LISTEN| head
tcp        0      0 10.200.0.112:37885          0.0.0.0:*                   LISTEN      27161/java
tcp        0      0 0.0.0.0:33533               0.0.0.0:*                   LISTEN      75666/java
tcp        0      0 10.200.0.112:39264          0.0.0.0:*                   LISTEN      189874/java
```

Netstat –apn 查看tcp/udp连接，及端口占用情况

历史数据查询-falcon



	last	min	avg	max
🟨 cpu.iowait	0.000	0.000	0.039	2.201
🟦 cpu.system	0.075	0.000	0.191	4.028
🟥 cpu.user	0.125	0.000	1.000	22.774

http://uic.lianjia.com/
可查看监控图，设置报警

Wiki说明：
http://wiki.lianjia.com/pages/viewpage.action?pageId=8146307

目录

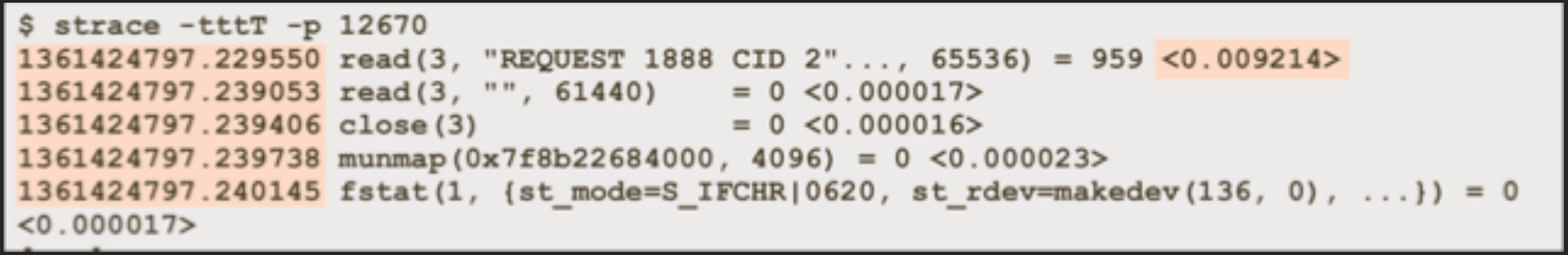
- 系统性能指标和观测方法
- 常用命令及平台-分析系统负载
- 动态追踪-了解程序在做什么
- 实战案例
- Q&A

动态追踪定义

无需修改程序源代码，活体分析
可实时跟踪程序状态，用于统计及分析
可以动态注入探针

本章主要介绍通用工具，及java相关工具

Strace-系统调用跟踪

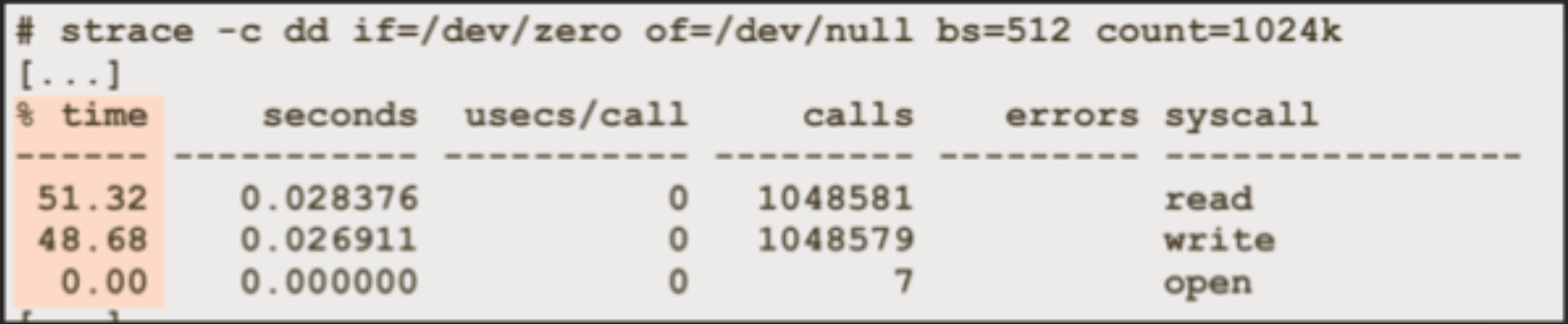


```
$ strace -tttT -p 12670
1361424797.229550 read(3, "REQUEST 1888 CID 2"..., 65536) = 959 <0.009214>
1361424797.239053 read(3, "", 61440)     = 0 <0.000017>
1361424797.239406 close(3)               = 0 <0.000016>
1361424797.239738 munmap(0x7f8b22684000, 4096) = 0 <0.000023>
1361424797.240145 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
<0.000017>
```

-p 进程号

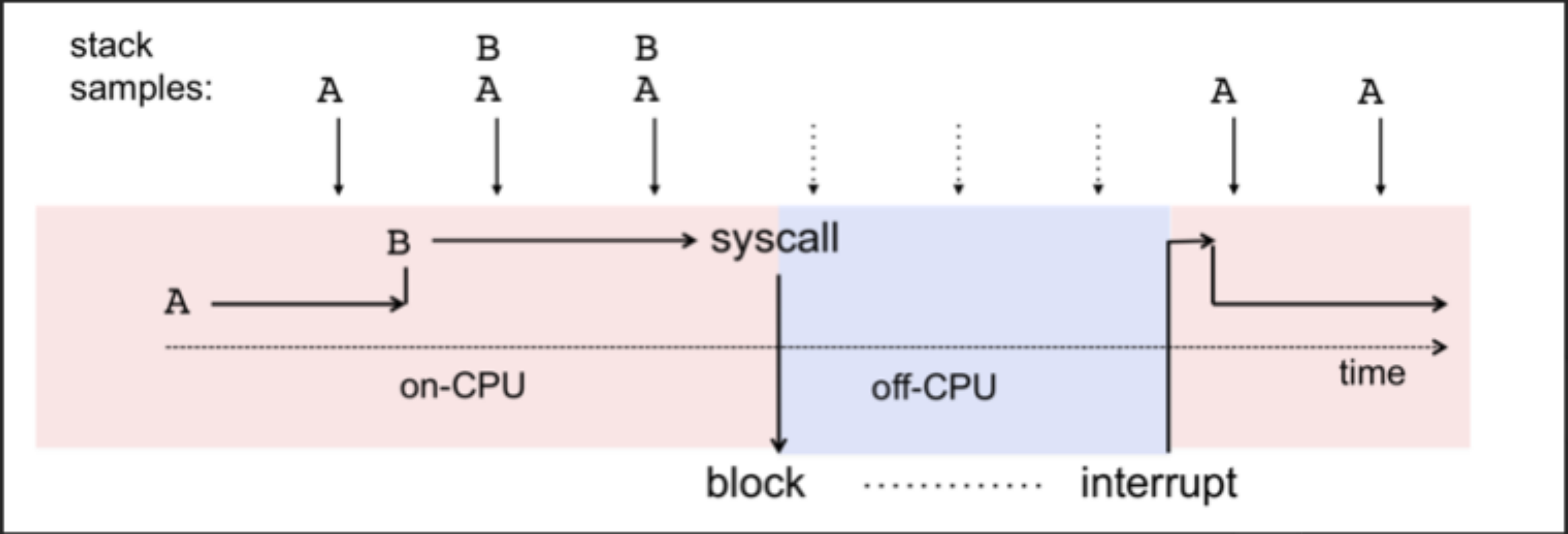
–c 统计耗时

相关命令：ltrace
跟踪glibc调用

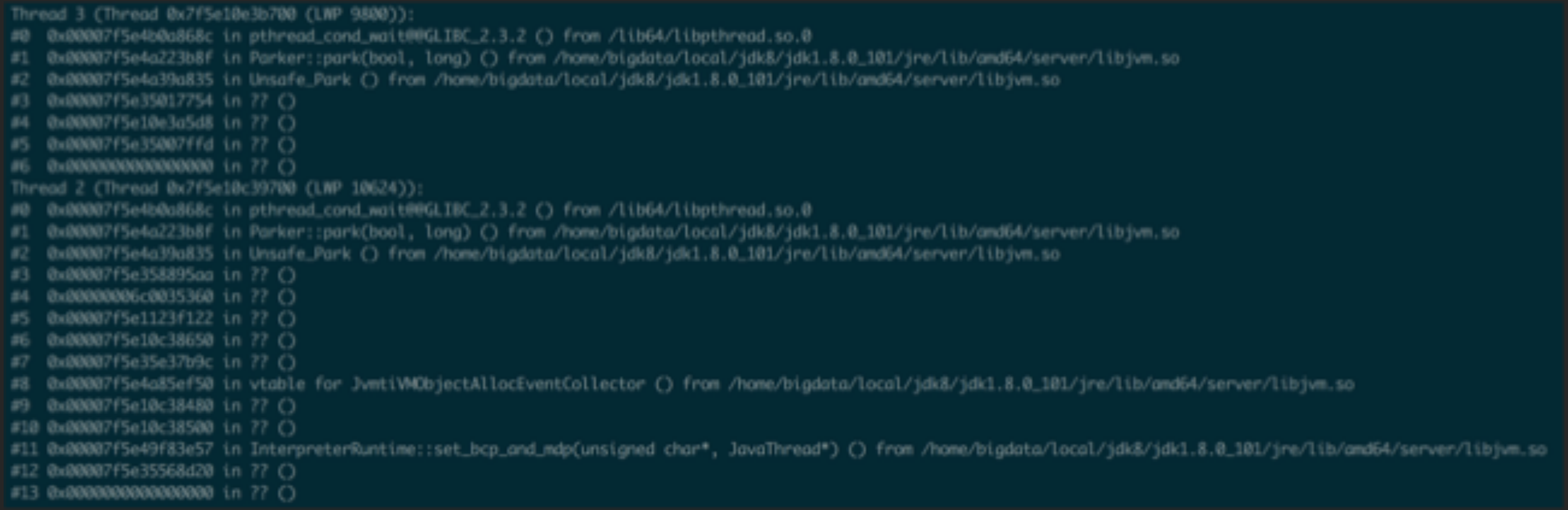


```
# strace -c dd if=/dev/zero of=/dev/null bs=512 count=1024k
[...]
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 51.32    0.028376           0   1048581           read
 48.68    0.026911           0   1048579           write
  0.00    0.000000           0         7           open
```

函数调用/CPU执行过程



Pstack-跟踪线程堆栈（瞬态）



```
Thread 3 (Thread 0x7f5e10e3b700 (LWP 9800)):
#0  0x00007f5e4b0a868c in pthread_cond_wait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
#1  0x00007f5e4a223b8f in Parker::park(bool, long) () from /home/bigdata/local/jdk8/jdk1.8.0_101/jre/lib/amd64/server/libjvm.so
#2  0x00007f5e4a39a835 in Unsafe_Park () from /home/bigdata/local/jdk8/jdk1.8.0_101/jre/lib/amd64/server/libjvm.so
#3  0x00007f5e35017754 in ?? ()
#4  0x00007f5e10e3a5d8 in ?? ()
#5  0x00007f5e35007ffd in ?? ()
#6  0x0000000000000000 in ?? ()
Thread 2 (Thread 0x7f5e10c39700 (LWP 10624)):
#0  0x00007f5e4b0a868c in pthread_cond_wait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
#1  0x00007f5e4a223b8f in Parker::park(bool, long) () from /home/bigdata/local/jdk8/jdk1.8.0_101/jre/lib/amd64/server/libjvm.so
#2  0x00007f5e4a39a835 in Unsafe_Park () from /home/bigdata/local/jdk8/jdk1.8.0_101/jre/lib/amd64/server/libjvm.so
#3  0x00007f5e358895aa in ?? ()
#4  0x00000006c0035360 in ?? ()
#5  0x00007f5e1123f122 in ?? ()
#6  0x00007f5e10c38650 in ?? ()
#7  0x00007f5e35e37b9c in ?? ()
#8  0x00007f5e4a85ef50 in vtable for JvmtiVMObjectAllocEventCollector () from /home/bigdata/local/jdk8/jdk1.8.0_101/jre/lib/amd64/server/libjvm.so
#9  0x00007f5e10c38480 in ?? ()
#10 0x00007f5e10c38500 in ?? ()
#11 0x00007f5e49f83e57 in InterpreterRuntime::set_bcp_and_mdp(unsigned char*, JavaThread*) () from /home/bigdata/local/jdk8/jdk1.8.0_101/jre/lib/amd64/server/libjvm.so
#12 0x00007f5e35568d20 in ?? ()
#13 0x0000000000000000 in ?? ()
```

只能显示c函数栈

- 各语言内部栈有各种工具
- 可排查jvm本身的异常

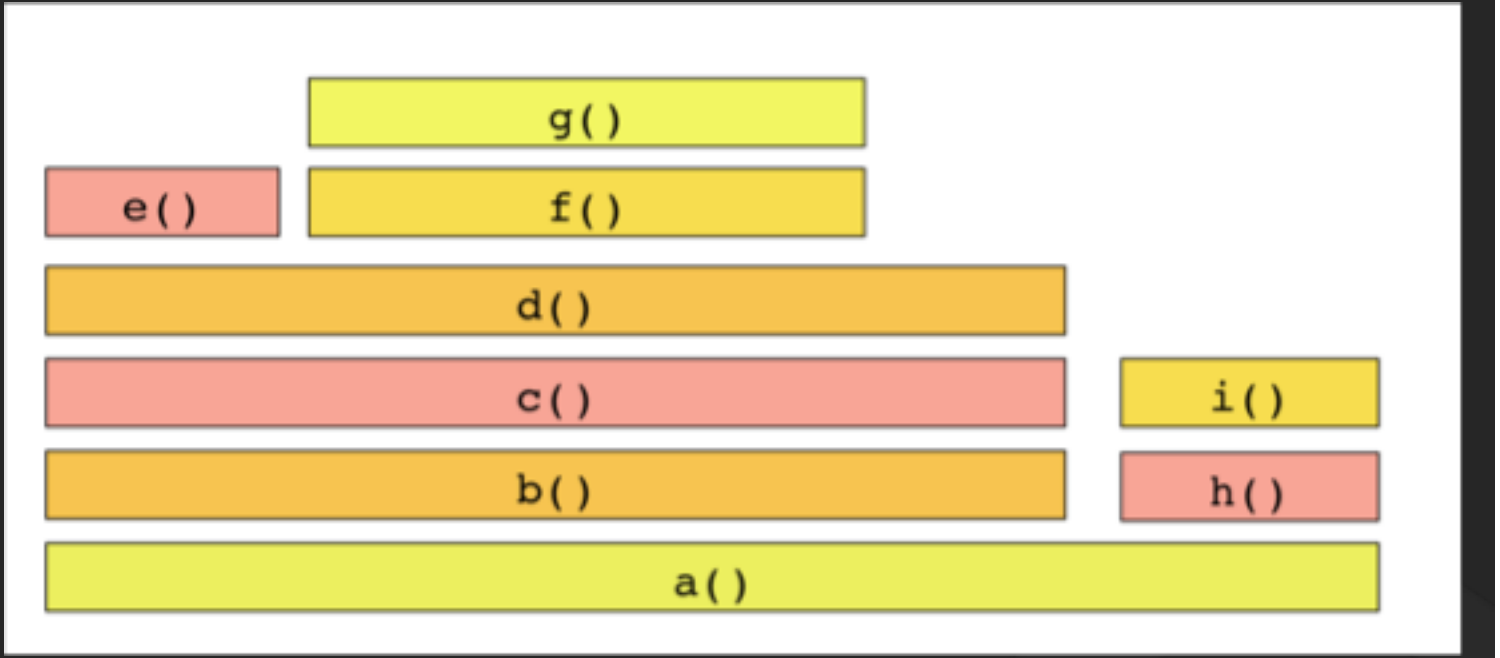
需要编译有符号信息

火焰图-函数cpu占用可视化（统计）

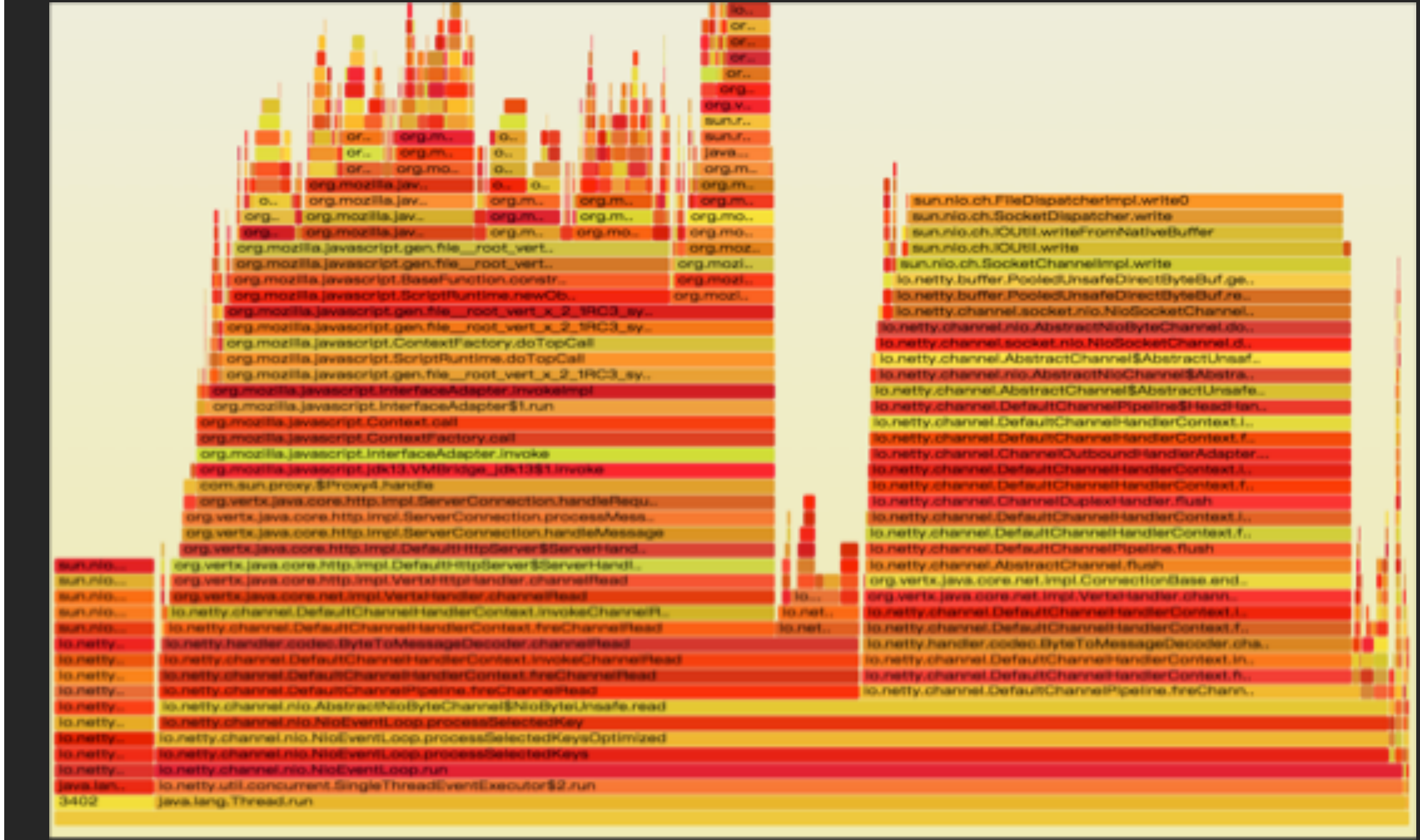
一定时间内采样堆栈调用生成

函数宽度表示cpu时间占比

垂直方向表示函数调用栈



真实火焰图样例

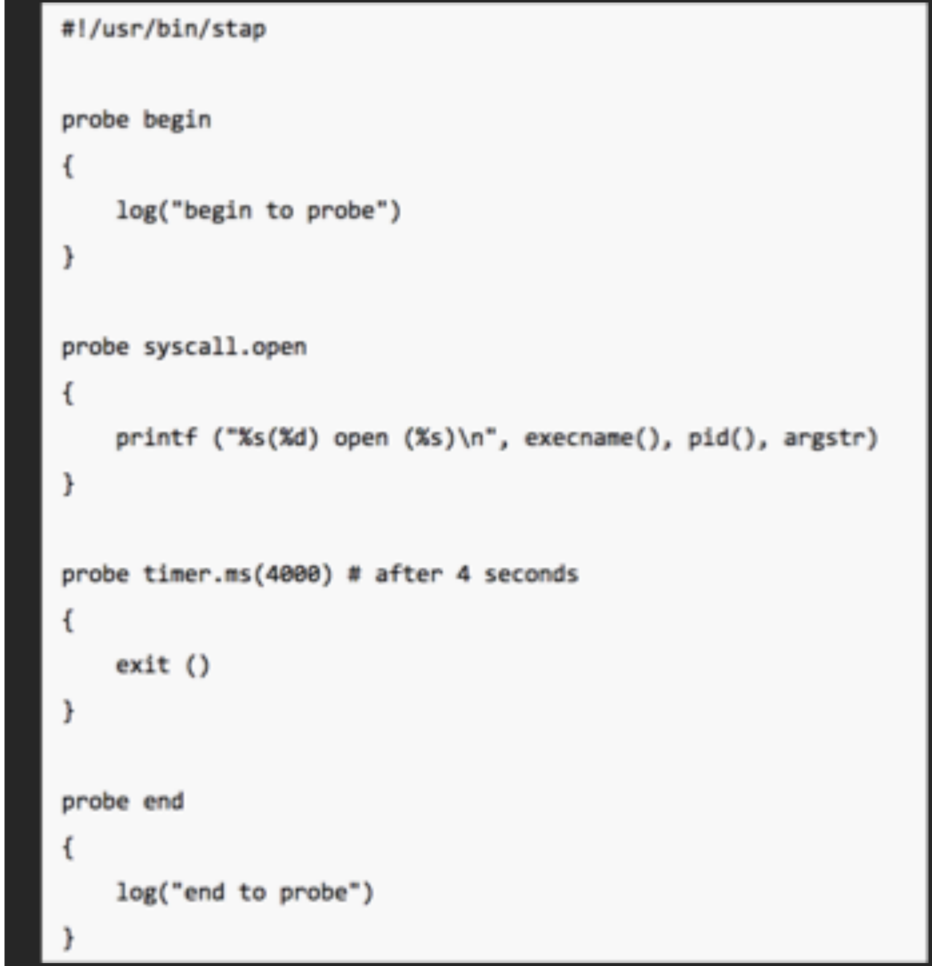


通用方法只能生成C调用

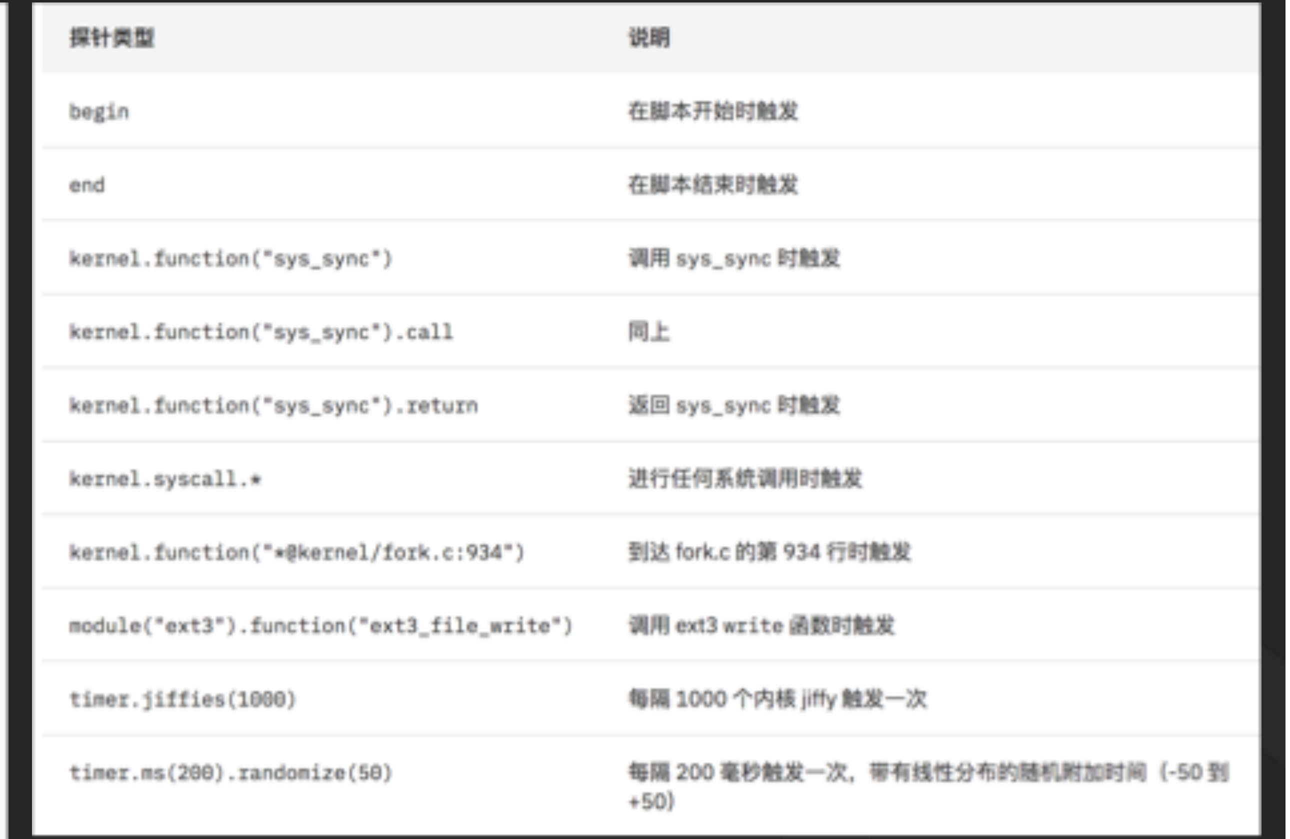
不同语言有各种工具

http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

Systemtap-内核动态探针



```
#!/usr/bin/stap

probe begin
{
    log("begin to probe")
}

probe syscall.open
{
    printf ("%s(%d) open (%s)\n", execname(), pid(), argstr)
}

probe timer.ms(4000) # after 4 seconds
{
    exit ()
}

probe end
{
    log("end to probe")
}
```

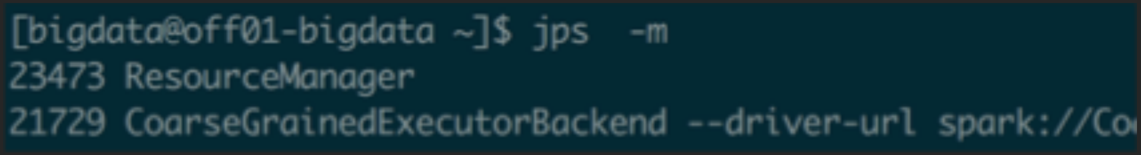


探针类型	说明
begin	在脚本开始时触发
end	在脚本结束时触发
kernel.function("sys_sync")	调用 sys_sync 时触发
kernel.function("sys_sync").call	同上
kernel.function("sys_sync").return	返回 sys_sync 时触发
kernel.syscall.*	进行任何系统调用时触发
kernel.function("*@kernel/fork.c:934")	到达 fork.c 的第 934 行时触发
module("ext3").function("ext3_file_write")	调用 ext3 write 函数时触发
timer.jiffies(1000)	每隔 1000 个内核 jiffy 触发一次
timer.ms(200).randomize(50)	每隔 200 毫秒触发一次，带有线性分布的随机附加时间（-50 到 +50）

支持系统调用，内核函数，模块函数

Java专属工具集

jps

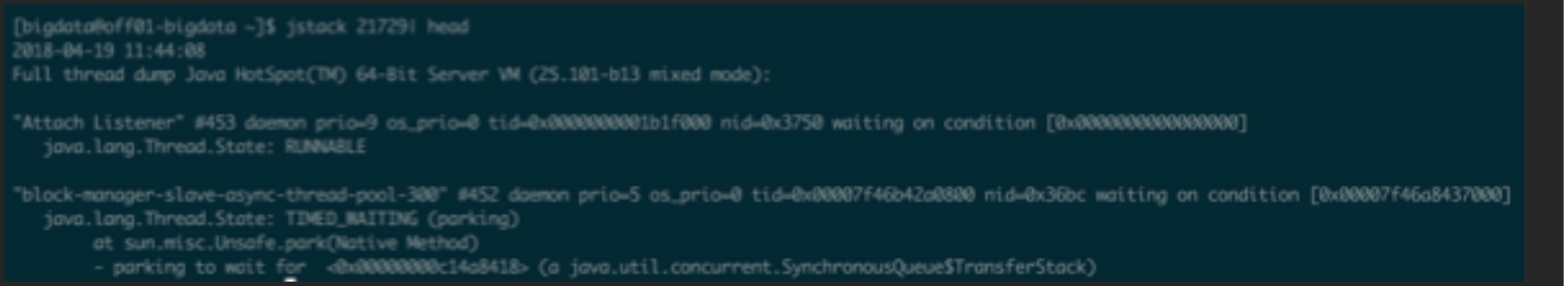


```
[bigdata@off01-bigdata ~]$ jps  -m
23473 ResourceManager
21729 CoarseGrainedExecutorBackend --driver-url spark://Co
```

Jstack

查看java堆栈，nid是系统线程号

死锁检测



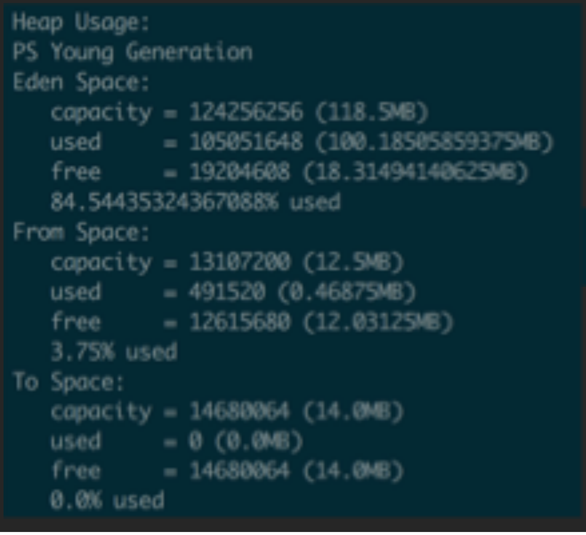
```
[bigdata@off01-bigdata ~]$ jstack 21729| head
2018-04-19 11:44:08
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.101-b13 mixed mode):

"Attach Listener" #453 daemon prio=9 os_prio=0 tid=0x0000000001b1f000 nid=0x3750 waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"block-manager-slave-async-thread-pool-300" #452 daemon prio=5 os_prio=0 tid=0x00007f46b42a0800 nid=0x36bc waiting on condition [0x00007f46a8437000]
   java.lang.Thread.State: TIMED_WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00000000c14a8418> (a java.util.concurrent.SynchronousQueue$TransferStack)
```

jmap

-heap 查看分代情况

-dump 导出内存镜像

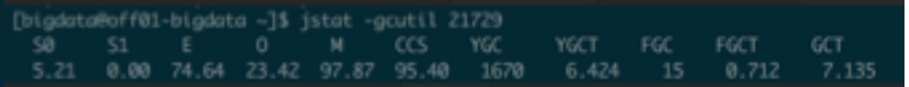
可用mat等分析对象个数，内存占用



```
Heap Usage:
PS Young Generation
Eden Space:
   capacity = 124256256 (118.5MB)
   used     = 105051648 (100.18505859375MB)
   free     = 19204608 (18.31494140625MB)
   84.54435324367088% used
From Space:
   capacity = 13107200 (12.5MB)
   used     = 491520 (0.46875MB)
   free     = 12615680 (12.03125MB)
   3.75% used
To Space:
   capacity = 14680064 (14.0MB)
   used     = 0 (0.0MB)
   free     = 14680064 (14.0MB)
   0.0% used
```

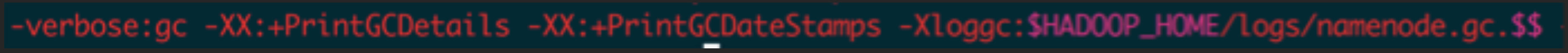
jstat

gc情况分析



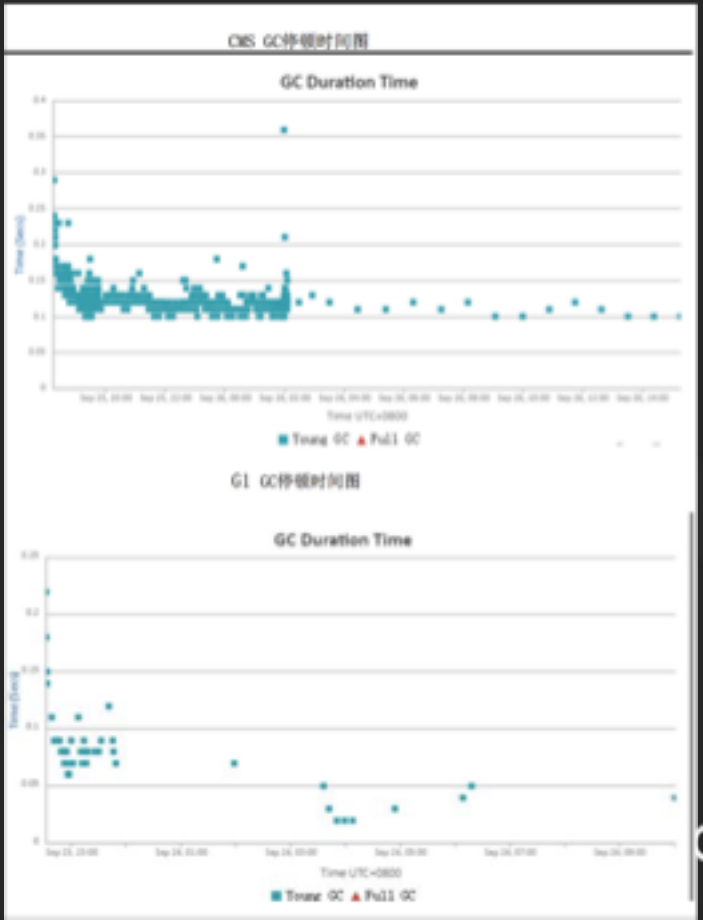
```
[bigdata@off01-bigdata ~]$ jstat -gcutil 21729
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT
  5.21   0.00  74.64  23.42  97.87  95.40   1670    6.424    15    0.712    7.135
```

Gc可视化分析



```
-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:$HADOOP_HOME/logs/namenode.gc.$$
```

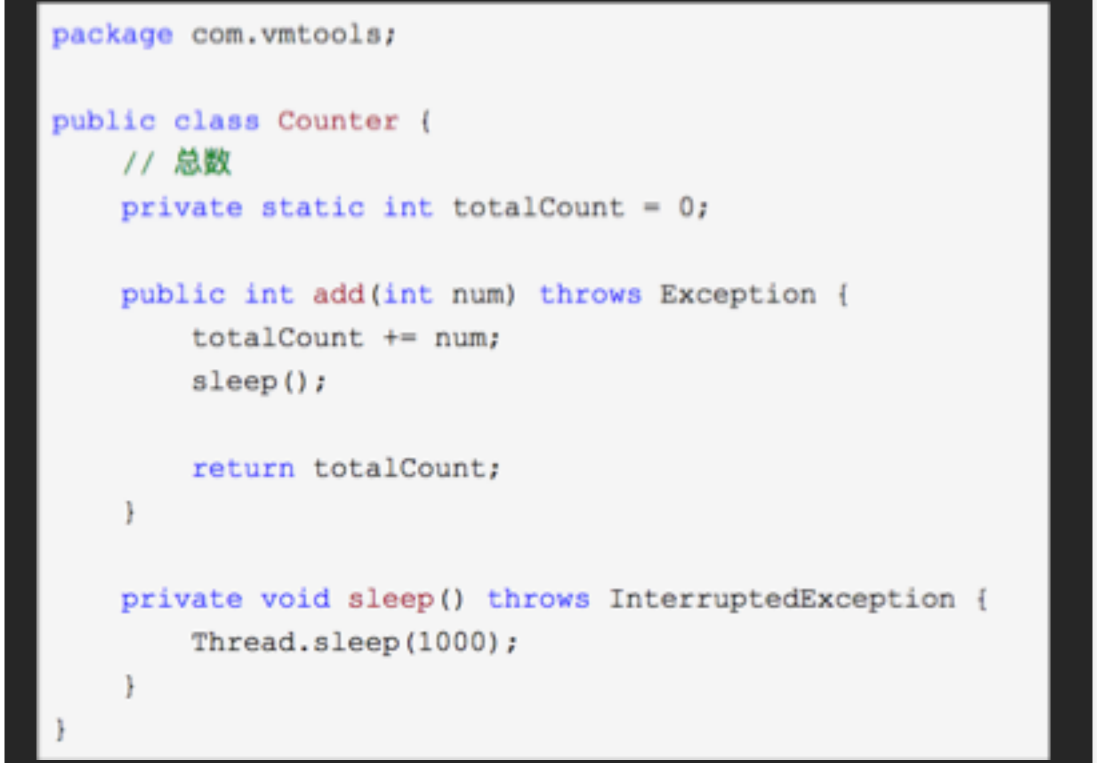
http://gceasy.io/index.jsp



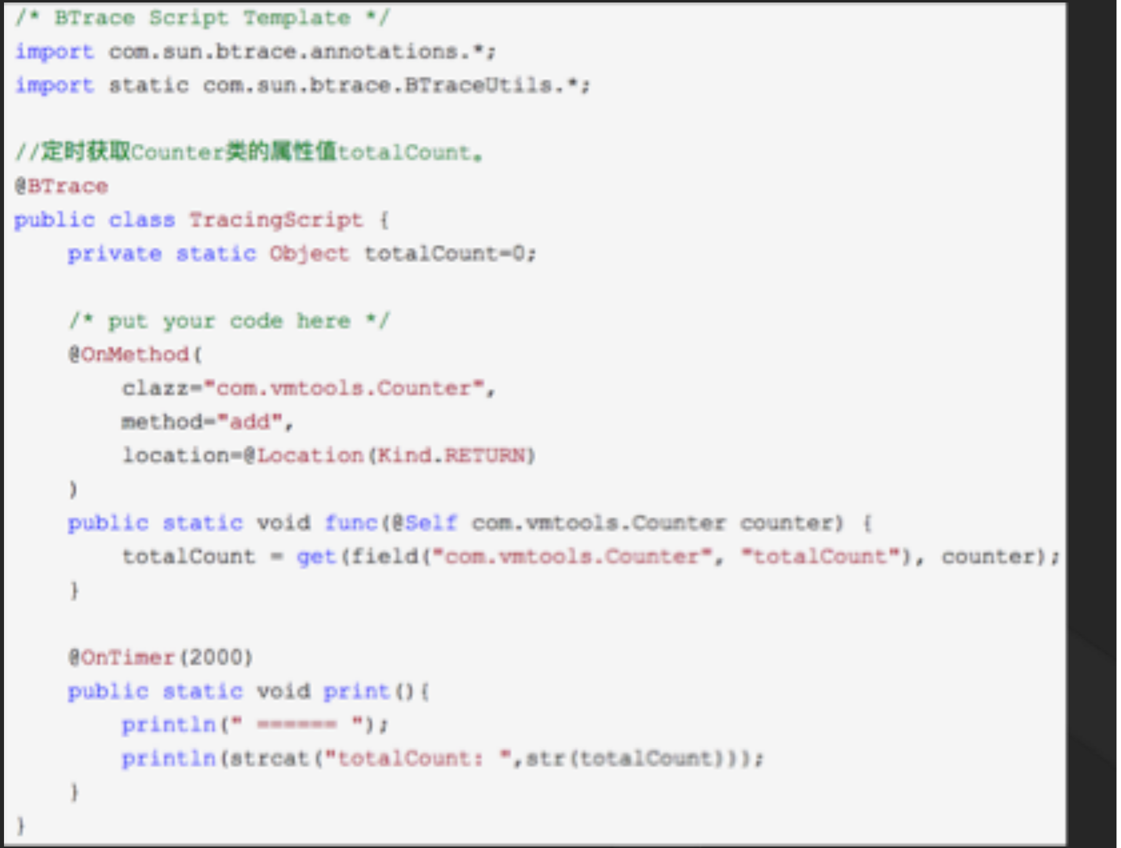
CMS GC停顿时间图

G1 GC停顿时间图

btrace-Java探针



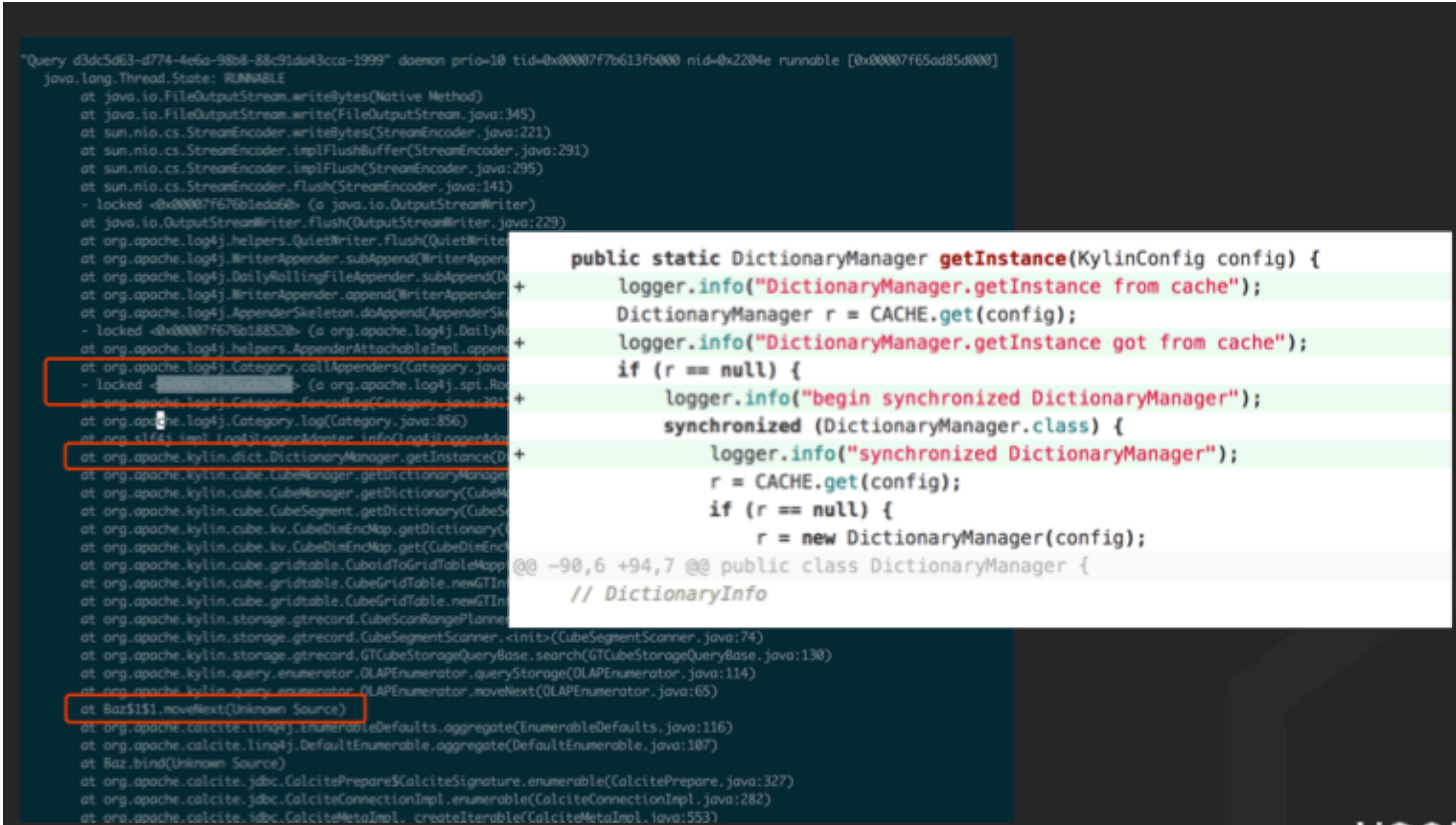
```java
package com.vmtools;

public class Counter {
    // 总数
    private static int totalCount = 0;

    public int add(int num) throws Exception {
        totalCount += num;
        sleep();

        return totalCount;
    }

    private void sleep() throws InterruptedException {
        Thread.sleep(1000);
    }
}
```



```java
/* BTrace Script Template */
import com.sun.btrace.annotations.*;
import static com.sun.btrace.BTraceUtils.*;

//定时获取Counter类的属性值totalCount。
@BTrace
public class TracingScript {
    private static Object totalCount=0;

    /* put your code here */
    @OnMethod(
        clazz="com.vmtools.Counter",
        method="add",
        location=@Location(Kind.RETURN)
    )
    public static void func(@Self com.vmtools.Counter counter) {
        totalCount = get(field("com.vmtools.Counter", "totalCount"), counter);
    }

    @OnTimer(2000)
    public static void print(){
        println(" ====== ");
        println(strcat("totalCount: ",str(totalCount)));
    }
}
```

目录

- 系统性能指标和观测方法
- 常用命令及平台-分析系统负载
- 动态追踪-了解程序在做什么
- 实战案例
- Q&A

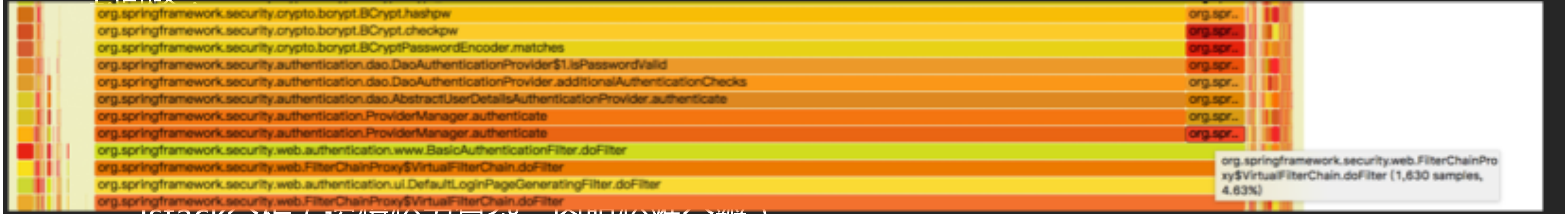


```
"Query d3dc5d63-d774-4e6a-98b8-88c91da43cca-1999" daemon prio=10 tid=0x00007f7b613fb000 nid=0x2204e runnable [0x00007f65ad85d000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileOutputStream.writeBytes(Native Method)
        at java.io.FileOutputStream.write(FileOutputStream.java:345)
        at sun.nio.cs.StreamEncoder.writeBytes(StreamEncoder.java:221)
        at sun.nio.cs.StreamEncoder.implFlushBuffer(StreamEncoder.java:291)
        at sun.nio.cs.StreamEncoder.implFlush(StreamEncoder.java:295)
        at sun.nio.cs.StreamEncoder.flush(StreamEncoder.java:141)
        - locked <0x00007f676b1eda60> (a java.io.OutputStreamWriter)
        at java.io.OutputStreamWriter.flush(OutputStreamWriter.java:229)
        at org.apache.log4j.helpers.QuietWriter.flush(QuietWrite
        at org.apache.log4j.WriterAppender.subAppend(WriterAppen
        at org.apache.log4j.DailyRollingFileAppender.subAppend(D
        at org.apache.log4j.WriterAppender.append(WriterAppender
        at org.apache.log4j.AppenderSkeleton.doAppend(AppenderSk
        - locked <0x00007f676b188520> (a org.apache.log4j.DailyR
        at org.apache.log4j.helpers.AppenderAttachableImpl.appen
        at org.apache.log4j.Category.callAppenders(Category.java
        - locked <[redacted]> (a org.apache.log4j.spi.Ro
        at org.apache.log4j.Category.forcedLog(Category.java:391
        at org.apache.log4j.Category.log(Category.java:856)
        at org.slf4j.impl.Log4jLoggerAdapter.info(Log4jLoggerAda
        at org.apache.kylin.dict.DictionaryManager.getInstance(D
        at org.apache.kylin.cube.CubeManager.getDictionaryManage
        at org.apache.kylin.cube.CubeManager.getDictionary(CubeM
        at org.apache.kylin.cube.CubeSegment.getDictionary(CubeS
        at org.apache.kylin.cube.kv.CubeDimEncMap.getDictionary(
        at org.apache.kylin.cube.kv.CubeDimEncMap.get(CubeDimEnc
        at org.apache.kylin.cube.gridtable.CuboidToGridTableMapp
        at org.apache.kylin.cube.gridtable.CubeGridTable.newGTIn
        at org.apache.kylin.cube.gridtable.CubeGridTable.newGTIn
        at org.apache.kylin.storage.gtrecord.CubeScanRangePlanne
        at org.apache.kylin.storage.gtrecord.CubeSegmentScanner.<init>(CubeSegmentScanner.java:74)
        at org.apache.kylin.storage.gtrecord.GTCubeStorageQueryBase.search(GTCubeStorageQueryBase.java:130)
        at org.apache.kylin.query.enumerator.OLAPEnumerator.queryStorage(OLAPEnumerator.java:114)
        at org.apache.kylin.query.enumerator.OLAPEnumerator.moveNext(OLAPEnumerator.java:65)
        at Baz$1$1.moveNext(Unknown Source)
        at org.apache.calcite.linq4j.EnumerableDefaults.aggregate(EnumerableDefaults.java:116)
        at org.apache.calcite.linq4j.DefaultEnumerable.aggregate(DefaultEnumerable.java:107)
        at Baz.bind(Unknown Source)
        at org.apache.calcite.jdbc.CalcitePrepare$CalciteSignature.enumerable(CalcitePrepare.java:327)
        at org.apache.calcite.jdbc.CalciteConnectionImpl.enumerable(CalciteConnectionImpl.java:282)
        at org.apache.calcite.jdbc.CalciteMetaImpl._createIterable(CalciteMetaImpl.java:553)
```

```
     public static DictionaryManager getInstance(KylinConfig config) {
+        logger.info("DictionaryManager.getInstance from cache");
         DictionaryManager r = CACHE.get(config);
+        logger.info("DictionaryManager.getInstance got from cache");
         if (r == null) {
+            logger.info("begin synchronized DictionaryManager");
             synchronized (DictionaryManager.class) {
+                logger.info("synchronized DictionaryManager");
                 r = CACHE.get(config);
                 if (r == null) {
                     r = new DictionaryManager(config);
@@ -90,6 +94,7 @@ public class DictionaryManager {
     // DictionaryInfo
```

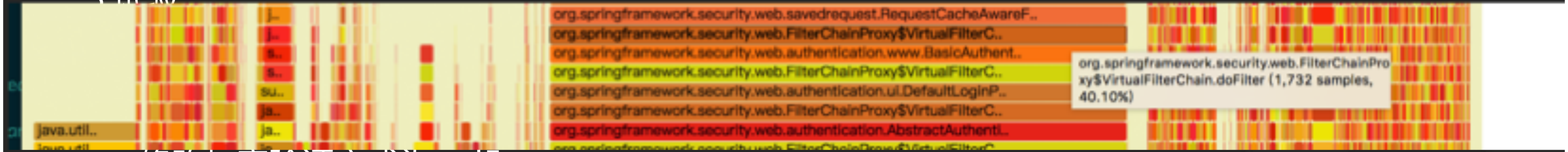
Kylin服务cpu占用过大问题

1.问题：


org.springframework.security.crypto.bcrypt.BCrypt.hashpw
org.springframework.security.crypto.bcrypt.BCrypt.checkpw
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder.matches
org.springframework.security.authentication.dao.DaoAuthenticationProvider$1.isPasswordValid
org.springframework.security.authentication.dao.DaoAuthenticationProvider.additionalAuthenticationChecks
org.springframework.security.authentication.dao.AbstractUserDetailsAuthenticationProvider.authenticate
org.springframework.security.authentication.ProviderManager.authenticate
org.springframework.security.authentication.ProviderManager.authenticate
org.springframework.security.web.authentication.www.BasicAuthenticationFilter.doFilter
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter
org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter.doFilter
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter
org.spr..
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter (1,630 samples, 4.63%)

jstack分析（逻辑较为复杂，肉眼较难分辨）

3.试验：


org.springframework.security.web.savedrequest.RequestCacheAwareF..
org.springframework.security.web.FilterChainProxy$VirtualFilterC..
org.springframework.security.web.authentication.www.BasicAuthent..
org.springframework.security.web.FilterChainProxy$VirtualFilterC..
org.springframework.security.web.authentication.ui.DefaultLoginP..
org.springframework.security.web.FilterChainProxy$VirtualFilterC..
org.springframework.security.web.authentication.AbstractAuthenti..
java.util..
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter (1,732 samples, 40.10%)

修改加密验证方式为md5

性能提升5倍

Cache对hbase的影响

1.疑惑：

hbase压测时经常宕掉，在将机器内存从64G升级128G非常稳定

2.排查/猜想：

hbase进程内存设置为50G，升级内存后也未调整

hbase本身堆使用率不高

多增加的内存会被OS用来做文件缓存(应该会大幅提升命中率)

相关资料提示hbase为IO敏感型，缓存命中率与稳定性有啥关系

3.试验：

借助systemtap统计cache命中率

systemtap分析cache

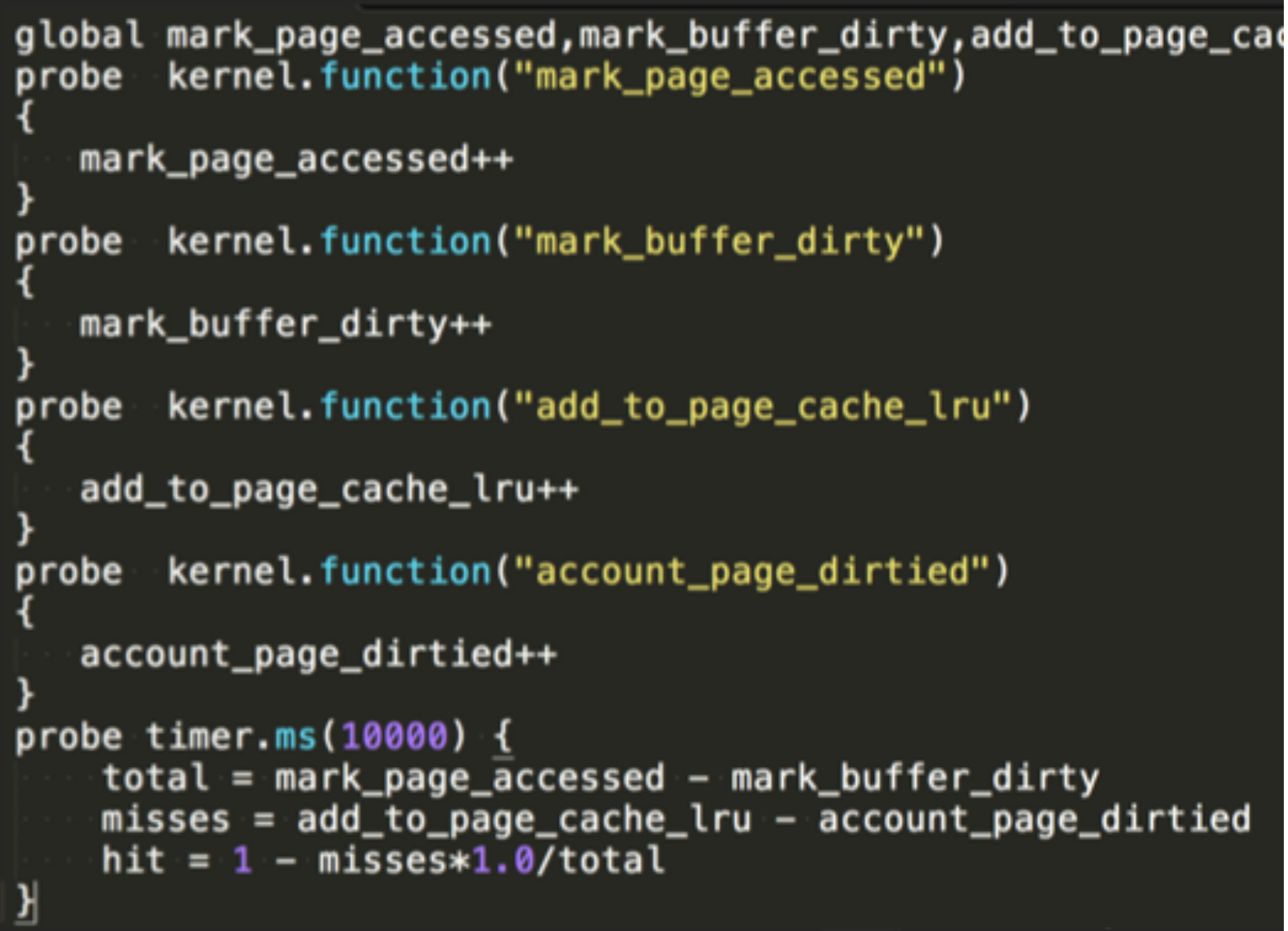
$$\text{Cache命中率} = 100\% - \frac{\text{添加page缓存次数(miss)}}{\text{page访问次数（}total\text{）}}$$

4.结果：

64G, hit=50-60%

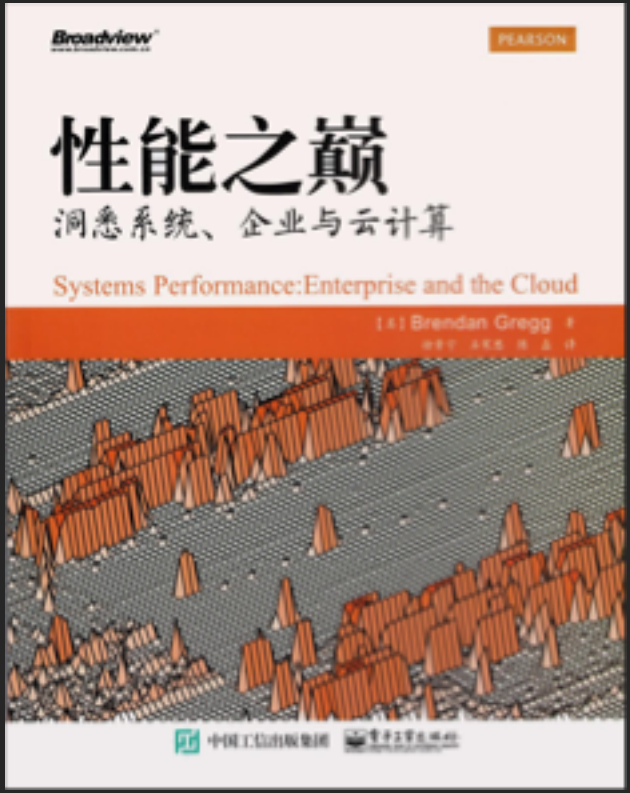
128G，hit=80-90%

hit>80%, 能保障稳定性sla



```
global mark_page_accessed,mark_buffer_dirty,add_to_page_cac
probe  kernel.function("mark_page_accessed")
{
    mark_page_accessed++
}
probe  kernel.function("mark_buffer_dirty")
{
    mark_buffer_dirty++
}
probe  kernel.function("add_to_page_cache_lru")
{
    add_to_page_cache_lru++
}
probe  kernel.function("account_page_dirtied")
{
    account_page_dirtied++
}
probe timer.ms(10000) {
     total = mark_page_accessed - mark_buffer_dirty
     misses = add_to_page_cache_lru - account_page_dirtied
     hit = 1 - misses*1.0/total
}
```

推荐资料



Brendan Gregg:
原SUN公司首席性能和内核专家
Solaris,dtrace等系统
Netflix首席架构师
http://www.brendangregg.com

章亦春：
OpenResty（nginx高性能模块）
开源项目创始人
阿里技术专家

Thanks!