Linux Performance Tools

Brendan Gregg
Senior Performance Architect
Performance Engineering Team

bgregg@netflix.com
@brendangregg
A quick tour of many tools...
• Massive AWS EC2 Linux cloud
  – Tens of thousands of instances
  – Autoscale by \(~3k\) each day
  – CentOS and Ubuntu
• FreeBSD for content delivery
  – Approx 33% of US Internet traffic at night
• Performance is critical
  – Customer satisfaction: >50M subscribers
  – $$$ price/performance
  – Develop tools for cloud-wide analysis; use server tools as needed
• Just launched in Europe!
Brendan Gregg

- Senior Performance Architect, Netflix
  - Linux and FreeBSD performance
  - Performance Engineering team (@coburnw)

- Recent work:
  - Linux perf-tools, using ftrace & perf_events
  - Systems Performance, Prentice Hall

- Previous work includes:
  - USE Method, flame graphs, utilization & latency heat maps, DTrace tools, ZFS L2ARC

- Twitter @brendangregg (these slides)
Aim: to show what can be done

Knowing that something can be done is more important than knowing how to do it.
Methodologies & Tools
Methodologies & Tools

• There are dozens of performance tools for Linux
  – Packages: sysstat, procps, coreutils, ...
  – Commercial products

• Methodologies can provide guidance for choosing and using tools effectively
Anti-Methodologies

• The lack of a deliberate methodology...
• Street Light Anti-Method:
  – 1. Pick observability tools that are
    • Familiar
    • Found on the Internet, or at random
  – 2. Run tools
  – 3. Look for obvious issues
• Drunk Man Anti-Method:
  – Tune things at random until the problem goes away
Methodologies

- For example, the USE Method:
  - For every resource, check:
    - Utilization
    - Saturation
    - Errors

- 5 Whys:
  - Ask “why?” 5 times

- Other methods include:
  - Workload characterization, drill-down analysis, event tracing, baseline stats, static performance tuning, ...

- Start with the questions, then find the tools
Command Line Tools

• Useful to study even if you never use them: GUIs and commercial products often use the same interfaces

```bash
$ vmstat 1
procs -----------memory---------- ---swap-- ...
  r  b  swpd  free  buff  cache  si  so ...
 9  0   0  29549320  29252  9299060   0  ...
 2  0   0  29547876  29252  9299332   0  ...
 4  0   0  29548124  29252  9299460   0  ...
 5  0   0  29548840  29252  9299592   0  ...
```
## Tool Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observability</td>
<td>Watch activity. Safe, usually, depending on resource overhead.</td>
</tr>
<tr>
<td>Benchmarking</td>
<td>Load test. Caution: production tests can cause issues due to contention.</td>
</tr>
<tr>
<td>Tuning</td>
<td>Change. Danger: changes could hurt performance, now or later with load.</td>
</tr>
<tr>
<td>Static</td>
<td>Check configuration. Should be safe.</td>
</tr>
</tbody>
</table>
Observability Tools
How do you measure these?
Observability Tools: Basic

- uptime
- top (or htop)
- ps
- vmstat
- iostat
- mpstat
- free
uptime

• One way to print *load averages*:

```bash
$ uptime
07:42:06 up 8:16, 1 user, load average: 2.27, 2.84, 2.91
```

• A measure of resource demand: CPUs + disks
  – Other OSes only show CPUs: easier to interpret

• Exponentially-damped moving averages with time constants of 1, 5, and 15 minutes
  – Historic trend without the line graph

• Load > # of CPUs, may mean CPU saturation
  – Don’t spend more than 5 seconds studying these
top (or htop)

- System and per-process interval summary:

```
$ top - 18:50:26 up  7:43,  1 user,  load average: 4.11, 4.91, 5.22
Tasks: 209 total,  1 running, 206 sleeping,  0 stopped,  2 zombie
Cpu(s): 47.1%us, 4.0%sy, 0.0%ni, 48.4%id, 0.0%wa, 0.0%hi, 0.3%si, 0.2%st
Mem:  70197156k total, 44831072k used, 25366084k free,  36360k buffers
Swap:     0k total,     0k used,     0k free, 11873356k cached

PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
5738 apiprod 20   0  62.6g  29g 352m S 417 44.2   2144:15 java
1386 apiprod 20   0  17452 1388  964 R  0  0.0   0:00.02 top
  1 root      20   0  24340 2272 1340 S  0  0.0   0:01.51 init
  2 root      20   0  0  0  0  0 S  0  0.0   0:00.00 kthreadd
[...]
```

- Can miss short-lived processes (atop won’t)
- Can consume noticeable CPU to read /proc
htop

```
1 [1111111111] 53.6% Tasks: 75, 55 thr; 1 running
2 [1111111111] 53.9%
Mem[1111111111] 489/7450MB
Swp[ ] 0/0MB

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PRI</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>CPU%</th>
<th>MEM%</th>
<th>TIME+</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>21162</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>22672</td>
<td>5216</td>
<td>1720</td>
<td>S</td>
<td>39.0</td>
<td>0.1</td>
<td>0:12.42</td>
<td>-bash</td>
</tr>
<tr>
<td>21542</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>24972</td>
<td>2608</td>
<td>1428</td>
<td>R</td>
<td>1.0</td>
<td>0.0</td>
<td>0:00.56</td>
<td>htop</td>
</tr>
<tr>
<td>1374</td>
<td>snmp</td>
<td>20</td>
<td>0</td>
<td>48320</td>
<td>4628</td>
<td>2352</td>
<td>S</td>
<td>0.0</td>
<td>0.1</td>
<td>1:17.87</td>
<td>/usr/sbin/snmpd --</td>
</tr>
<tr>
<td></td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>24332</td>
<td>2260</td>
<td>1340</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.44</td>
<td>/sbin/init</td>
</tr>
<tr>
<td>335</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>17236</td>
<td>640</td>
<td>452</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.05</td>
<td>upstart-udev-brid</td>
</tr>
<tr>
<td>340</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>21596</td>
<td>1300</td>
<td>800</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>/sbin/udevd --dae</td>
</tr>
<tr>
<td>368</td>
<td>messagebus</td>
<td>20</td>
<td>0</td>
<td>23820</td>
<td>944</td>
<td>640</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>dbus-daem --sys</td>
</tr>
<tr>
<td>421</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>21460</td>
<td>736</td>
<td>340</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>/sbin/daemon --sys</td>
</tr>
<tr>
<td>422</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>21460</td>
<td>736</td>
<td>340</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>/sbin/udevd --dae</td>
</tr>
<tr>
<td>530</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>15192</td>
<td>392</td>
<td>196</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>upstart-socket-br</td>
</tr>
<tr>
<td>604</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>7268</td>
<td>1028</td>
<td>532</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.01</td>
<td>dhclient3 -- IF_M</td>
</tr>
<tr>
<td>703</td>
<td>postfix</td>
<td>20</td>
<td>0</td>
<td>27176</td>
<td>1616</td>
<td>1316</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.01</td>
<td>pickup --l --t fifo</td>
</tr>
<tr>
<td>770</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>14508</td>
<td>976</td>
<td>812</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>/sbin/getty --8 38</td>
</tr>
<tr>
<td>775</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>14508</td>
<td>976</td>
<td>812</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>/sbin/getty --8 38</td>
</tr>
<tr>
<td>780</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>14508</td>
<td>976</td>
<td>812</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>/sbin/getty --8 38</td>
</tr>
<tr>
<td>781</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>14508</td>
<td>980</td>
<td>812</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>/sbin/getty --8 38</td>
</tr>
</tbody>
</table>
```

F1 Help  F2 Setup  F3 Search  F4 Filter  F5 Tree  F6 Sort By  F7 Nice  F8 Nice  F9 Kill  F10 Quit
• Process status listing (eg, “ASCII art forest”):

```
$ ps -ef f
UID   PID  PPID  C STIME TTY STAT TIME CMD
root  4546     1  0 11:08 ?   Ss  0:00 /usr/sbin/sshd -D
root  28261  4546  0 17:24 ?   Ss  0:00 \_ sshd: prod [priv]
    28287 28261  0 17:24 ?   S  0:00 \_ sshd: prod@pts/0
prod 28288 28287  0 17:24 pts/0 Ss  0:00 \_ -bash
    3156 28288  0 19:15 pts/0 R+  0:00 \_ ps -ef f
prod 4965 28288  0 17:24 ?   Ss  0:00 /bin/sh /usr/bin/svscanboot
root  4965     1  0 11:08 ?   Ss  0:00 \_ svscan /etc/service
root  4969  4965  0 11:08 ?   S  0:00 \_ svscan /etc/service
```

• Custom fields:

```
$ ps -eo user,sz,rss,minflt,majflt,pcpu,args
USER        SZ   RSS MINFLT MAJFLT %CPU COMMAND
root        6085  2272  11928    24   0.0 /sbin/init
```
vmstat

- Virtual memory statistics and more:

```
$ vmstat -Sm 1
procs -----------memory---------- ---swap-- -----io----- -system-- ----cpu-----
  r  b  swpd free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa
8  0   0   1620  149  552   0   0   1  179  77  12  25  34   0  0
7  0   0   1598  149  552   0   0   0  205 186  46  13   0  0
8  0   0   1617  149  552   0   0   0  210 435  39  21   0  0
8  0   0   1589  149  552   0   0   0  218 219  42  17   0  0
[...]
```

- USAGE: vmstat [interval [count]]
- First output line has some summary since boot values (should be all; partial is confusing)
- High level CPU summary. “r” is runnable tasks.
iostat

- Block I/O (disk) stats. 1st output is since boot.

```bash
$ iostat -xmdz 1
```

Linux 3.13.0-29 (db001-eb883efa) 08/18/2014 _x86_64_ (16 CPU)

<table>
<thead>
<tr>
<th>Device</th>
<th>rqn/s</th>
<th>wqn/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rMB/s</th>
<th>wMB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>xvda</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>xvdb</td>
<td>213.00</td>
<td>0.00</td>
<td>15299.00</td>
<td>0.00</td>
<td>338.17</td>
<td>0.00</td>
</tr>
<tr>
<td>xvdc</td>
<td>129.00</td>
<td>0.00</td>
<td>15271.00</td>
<td>3.00</td>
<td>336.65</td>
<td>0.01</td>
</tr>
<tr>
<td>md0</td>
<td>0.00</td>
<td>0.00</td>
<td>31082.00</td>
<td>3.00</td>
<td>678.45</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Workload

- Very useful set of stats

Resulting Performance
mpstat

- Multi-processor statistics, per-CPU:

```
$ mpstat -P ALL 1
[...]
08:06:43 PM   CPU  %usr  %nice  %sys  %iowait  %irq  %soft  %steal  %guest  %idle
08:06:44 PM   all  53.45  0.00   3.77   0.00   0.00   0.39   0.13   0.00  42.26
08:06:44 PM    0  49.49  0.00   3.03   0.00   0.00   1.01   1.01   0.00  45.45
08:06:44 PM    1  51.61  0.00   4.30   0.00   0.00   2.15   0.00   0.00  41.94
08:06:44 PM    2  58.16  0.00   7.14   0.00   0.00   0.00   1.02   0.00  33.67
08:06:44 PM    3  54.55  0.00   5.05   0.00   0.00   0.00   0.00   0.00  40.40
08:06:44 PM    4  47.42  0.00   3.09   0.00   0.00   0.00   0.00   0.00  49.48
08:06:44 PM    5  65.66  0.00   3.03   0.00   0.00   0.00   0.00   0.00  31.31
08:06:44 PM    6  50.00  0.00   2.08   0.00   0.00   0.00   0.00   0.00  47.92
[...]
```

- Look for unbalanced workloads, hot CPUs.
free

- Main memory usage:

  ```sh
  $ free -m
  
  total       used       free     shared    buffers     cached
  Mem:          3750       1111       2639          0        147        527
  -/+ buffers/cache:        436       3313
  Swap:            0          0          0
  
  buffers      cached
  147          527
  ```

- buffers: block device I/O cache
- cached: virtual page cache
Observability Tools: Basic
Observability Tools: Intermediate

• strace
• tcpdump
• netstat
• nicstat
• pidstat
• swapon
• lsof
• sar (and collectl, dstat, etc.)
strace

- System call tracer:

```bash
$ strace -tttT -p 313
1408393285.779746 getgroups(0, NULL) = 1 <0.000016>
1408393285.779873 getgroups(1, [0]) = 1 <0.000015>
1408393285.780797 close(3) = 0 <0.000016>
1408393285.781338 write(1, "LinuxCon 2014!\n", 15LinuxCon 2014! ) = 15 <0.000048>
```

- Eg, -ttt: time (us) since epoch; -T: syscall time (s)
- Translates syscall args
  - Very helpful for solving system usage issues
- Currently has massive overhead (ptrace based)
  - Can slow the target by > 100x. Use extreme caution.
tcpdump

- Sniff network packets for post analysis:

$ tcpdump -i eth0 -w /tmp/out.tcpdump

tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C7985 packets captured
8996 packets received by filter
1010 packets dropped by kernel

# tcpdump -nr /tmp/out.tcpdump | head

reading from file /tmp/out.tcpdump, link-type EN10MB (Ethernet)
20:41:05.038437 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 18...
20:41:05.038533 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 48...
20:41:05.038584 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 96...

- Study packet sequences with timestamps (us)
- CPU overhead optimized (socket ring buffers), but can still be significant. Use caution.
netstat

• Various network protocol statistics using -s:
  • A multi-tool:
    -i: interface stats
    -r: route table
    default: list conns
  • netstat -p: shows process details!
• Per-second interval with -c

$ netstat -s
[...]
Tcp:
  736455 active connections openings
  176887 passive connection openings
  33 failed connection attempts
  1466 connection resets received
  3311 connections established
  91975192 segments received
  180415763 segments send out
  223685 segments retransmited
  2 bad segments received.
  39481 resets sent
[...]
TcpExt:
  12377 invalid SYN cookies received
  2982 delayed acks sent
[...]
nicstat

- Network interface stats, iostat-like output:

```
$ ./nicstat 1

<table>
<thead>
<tr>
<th>Time</th>
<th>Int</th>
<th>rKB/s</th>
<th>wKB/s</th>
<th>rPk/s</th>
<th>wPk/s</th>
<th>rAvs</th>
<th>wAvs</th>
<th>%Util</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>21:21:43</td>
<td>lo</td>
<td>823.0</td>
<td>823.0</td>
<td>171.5</td>
<td>171.5</td>
<td>4915.4</td>
<td>4915.4</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>21:21:43</td>
<td>eth0</td>
<td>5.53</td>
<td>1.74</td>
<td>15.11</td>
<td>12.72</td>
<td>374.5</td>
<td>139.8</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>21:21:44</td>
<td>lo</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>21:21:44</td>
<td>eth0</td>
<td>20.42</td>
<td>3394.1</td>
<td>355.8</td>
<td>85.94</td>
<td>58.76</td>
<td>40441.3</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>21:21:45</td>
<td>lo</td>
<td>1409.1</td>
<td>1409.1</td>
<td>327.9</td>
<td>327.9</td>
<td>4400.8</td>
<td>4400.8</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>21:21:45</td>
<td>eth0</td>
<td>75.12</td>
<td>4402.3</td>
<td>1398.9</td>
<td>1513.2</td>
<td>54.99</td>
<td>2979.1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
```

- Check network throughput and interface %util
- I wrote this years ago; Tim Cook ported to Linux
pidstat

- Very useful process stats. eg, by-thread, disk I/O:

```
$ pidstat -t 1
Linux 3.2.0-54 (db002-91befe03) 08/18/2014 _x86_64_ (8 CPU)

<table>
<thead>
<tr>
<th>Time</th>
<th>TGID</th>
<th>TID</th>
<th>%usr</th>
<th>%system</th>
<th>%guest</th>
<th>%CPU</th>
<th>CPU</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:57:52 PM</td>
<td>5738</td>
<td>-</td>
<td>484.75</td>
<td>39.83</td>
<td>0.00</td>
<td>524.58</td>
<td>1</td>
<td>java</td>
</tr>
<tr>
<td>08:57:54 PM</td>
<td>5817</td>
<td>5738</td>
<td>0.85</td>
<td>0.00</td>
<td>0.00</td>
<td>0.85</td>
<td>2</td>
<td>java</td>
</tr>
<tr>
<td>08:57:54 PM</td>
<td>5931</td>
<td>5817</td>
<td>1.69</td>
<td>1.69</td>
<td>0.00</td>
<td>3.39</td>
<td>4</td>
<td>java</td>
</tr>
<tr>
<td>08:57:54 PM</td>
<td>5981</td>
<td>5931</td>
<td>0.85</td>
<td>0.00</td>
<td>0.00</td>
<td>0.85</td>
<td>7</td>
<td>java</td>
</tr>
<tr>
<td>08:57:54 PM</td>
<td>5990</td>
<td>5981</td>
<td>0.85</td>
<td>0.00</td>
<td>0.00</td>
<td>0.85</td>
<td>4</td>
<td>java</td>
</tr>
</tbody>
</table>

[...]

$ pidstat -d 1

[...]

<table>
<thead>
<tr>
<th>Time</th>
<th>PID</th>
<th>kB_rd/s</th>
<th>kB_wr/s</th>
<th>kB_ccwr/s</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:58:27 PM</td>
<td>5738</td>
<td>0.00</td>
<td>815.69</td>
<td>0.00</td>
<td>java</td>
</tr>
</tbody>
</table>
```

- I usually prefer this over `top(1)`
swapon

• Show swap device usage:

```
$ swapon -s
Filename                Type        Size      Used    Priority
/dev/sda3               partition   5245212    284    -1
```

• If you have swap enabled...
ls/of

• More a debug tool, lsof(8) shows file descriptor usage, which for some apps, equals current active network connections:

```bash
# lsof -iTCP -STCP:ESTABLISHED
COMMAND    PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd 755  root 3r  IPv4 13576887 0t0  TCP bgregg-test-i-f106:ssh->prod100.netflix.com:15241 (ESTABLISHED)
platforms 2614 app1 8u IPv4 14618 0t0  TCP localhost:33868->localhost:5433 (ESTABLISHED)
gress 2648 app1 7u IPv4 14619 0t0  TCP localhost:5433->localhost:33868 (ESTABLISHED)
epic_plug 2857 app1 7u IPv4 15678 0t0  TCP localhost:33885->localhost:5433 (ESTABLISHED)
gress 2892 app1 7u IPv4 15679 0t0  TCP localhost:5433->localhost:33885 (ESTABLISHED)
[...]
```

• I’d prefer to: `echo /proc/PID/fd | wc -l`
sar

• System Activity Reporter. Many stats, eg:

```bash
$ sar -n TCP,ETCP,DEV 1
Linux 3.2.55 (test-e4f1a80b) 08/18/2014 _x86_64_ (8 CPU)

09:10:43 PM  IFACE  rxpck/s  txpck/s  rxkB/s  txkB/s  rxcmp/s  txcmp/s  rxmcst/s
09:10:44 PM  lo     14.00   14.00   1.34    1.34    0.00    0.00    0.00
09:10:44 PM  eth0   4114.00 4186.00 4537.46 28513.24 0.00    0.00    0.00

09:10:43 PM  active/s passive/s  iseg/s  oseg/s
09:10:44 PM  21.00    4.00   4107.00 22511.00

09:10:43 PM  atmptf/s  estres/s  retrans/s  isegerr/s  orsts/s
09:10:44 PM  0.00    0.00    36.00    0.00    1.00
[...]
```

• Archive or live mode: (interval [count])
• Well designed. Header naming convention, logical groups: TCP, ETCP, DEV, EDEV, ...
Other Tools

- You may also use collectl, atop, dstat, or another measure-all tool
- The tool isn’t important
- It’s important to have a way to measure everything you want
- In cloud environments, you are probably using a monitoring product, developed in-house or commercial. Same method applies...
How does your monitoring tool measure these?
Observability Tools: Intermediate
Advanced Observability Tools

• Misc:
  – ltrace, ss, iptraf, ethtool, snmpget, lldptool, iotop, blktrace, slabtop, /proc, pcstat

• CPU Performance Counters:
  – perf_events, tiptop, rdmsr

• Advanced Tracers:
  – perf_events, ftrace, eBPF, SystemTap, ktap, LTTng, dtrace4linux, sysdig

• Some selected demos...
More socket statistics:

```bash
$ ss -mop
State       Recv-Q Send-Q      Local Address:Port            Peer Address:Port
CLOSE-WAIT  1 0              127.0.0.1:42295           127.0.0.1:28527
users:({"apacheLogParser",2702,3})
    mem:(r1280,w0,f2816,t0)
ESTAB       0 0              127.0.0.1:5433            127.0.0.1:41312
    timer:(keepalive,36min,0) users:({"postgres",2333,7})
    mem:(r0,w0,f0,t0)
[...]
$ ss -i
$ ss -mop
State       Recv-Q Send-Q      Local Address:Port            Peer Address:Port
CLOSE-WAIT  1 0              127.0.0.1:42295           127.0.0.1:28527
    cubic wscale:6,6 rto:208 rtt:9/6 ato:40 cwnd:10 send 145.6Mbps rcv_space:32792
ESTAB       0 0              10.144.107.101:ssh         10.53.237.72:4532
    cubic wscale:4,6 rto:268 rtt:71.5/3 ato:40 cwnd:10 send 1.5Mbps rcv_rtt:72
    rcv_space:14480
[...]
```
### IPtraf

#### Packet Distribution by Size

**Packet size brackets for interface eth0**

<table>
<thead>
<tr>
<th>Packet Size (bytes)</th>
<th>Count</th>
<th>Packet Size (bytes)</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 75:</td>
<td>62148</td>
<td>751 to 825:</td>
<td>84</td>
</tr>
<tr>
<td>76 to 150:</td>
<td>5734</td>
<td>826 to 900:</td>
<td>61</td>
</tr>
<tr>
<td>151 to 225:</td>
<td>25519</td>
<td>901 to 975:</td>
<td>45</td>
</tr>
<tr>
<td>226 to 300:</td>
<td>20246</td>
<td>976 to 1050:</td>
<td>63</td>
</tr>
<tr>
<td>301 to 375:</td>
<td>5011</td>
<td>1051 to 1125:</td>
<td>49</td>
</tr>
<tr>
<td>376 to 450:</td>
<td>802</td>
<td>1126 to 1200:</td>
<td>47</td>
</tr>
<tr>
<td>451 to 525:</td>
<td>677</td>
<td>1201 to 1275:</td>
<td>65</td>
</tr>
<tr>
<td>526 to 600:</td>
<td>274</td>
<td>1276 to 1350:</td>
<td>52</td>
</tr>
<tr>
<td>601 to 675:</td>
<td>135</td>
<td>1351 to 1425:</td>
<td>339</td>
</tr>
<tr>
<td>676 to 750:</td>
<td>105</td>
<td>1426 to 1500+:</td>
<td>3696</td>
</tr>
</tbody>
</table>

Interface MTU is 1500 bytes, not counting the data-link header
Maximum packet size is the MTU plus the data-link header length
Packet size computations include data-link headers, if any
io\text{top}

- Block device I/O (disk) by process:

```bash
$ iotop

Total DISK READ: 50.47 M/s | Total DISK WRITE: 59.21 M/s

<table>
<thead>
<tr>
<th>TID</th>
<th>PRIO</th>
<th>USER</th>
<th>DISK READ</th>
<th>DISK WRITE</th>
<th>SWAPIN</th>
<th>IO&gt;</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>959</td>
<td>be/4</td>
<td>root</td>
<td>0.00 B/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>99.99 %</td>
<td>[flush-202:1]</td>
</tr>
<tr>
<td>6641</td>
<td>be/4</td>
<td>root</td>
<td>50.47 M/s</td>
<td>82.60 M/s</td>
<td>0.00 %</td>
<td>32.51 %</td>
<td>java –Dnop –X</td>
</tr>
<tr>
<td>1</td>
<td>be/4</td>
<td>root</td>
<td>0.00 B/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>be/4</td>
<td>root</td>
<td>0.00 B/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>[kthreadd]</td>
</tr>
<tr>
<td>3</td>
<td>be/4</td>
<td>root</td>
<td>0.00 B/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>[ksoftirqd/0]</td>
</tr>
<tr>
<td>4</td>
<td>be/4</td>
<td>root</td>
<td>0.00 B/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>[kworker/0:0]</td>
</tr>
<tr>
<td>5</td>
<td>be/4</td>
<td>root</td>
<td>0.00 B/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>[kworker/u:0]</td>
</tr>
<tr>
<td>6</td>
<td>rt/4</td>
<td>root</td>
<td>0.00 B/s</td>
<td>0.00 B/s</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>[migration/0]</td>
</tr>
</tbody>
</table>

[...]```

- Needs kernel support enabled
  - CONFIG\_TASK\_IO\_ACCOUNTING
slabtop

- Kernel slab allocator memory usage:

```
$ slabtop
Active / Total Objects (% used)  : 4692768 / 4751161 (98.8%)
Active / Total Slabs (% used)    : 129083 / 129083 (100.0%)
Active / Total Caches (% used)   : 71 / 109 (65.1%)
Active / Total Size (% used)     : 729966.22K / 738277.47K (98.9%)
Minimum / Average / Maximum Object: 0.01K / 0.16K / 8.00K

<table>
<thead>
<tr>
<th>OBJS</th>
<th>ACTIVE</th>
<th>USE OBJ</th>
<th>SIZE</th>
<th>SLABS</th>
<th>OBJ/SLAB</th>
<th>CACHE</th>
<th>SIZE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>356575</td>
<td>356575</td>
<td>100%</td>
<td>0.10K</td>
<td>91425</td>
<td>39</td>
<td></td>
<td>365700K</td>
<td>buffer_head</td>
</tr>
<tr>
<td>314916</td>
<td>314096</td>
<td>99%</td>
<td>0.19K</td>
<td>14996</td>
<td>21</td>
<td></td>
<td>59984K</td>
<td>dentry</td>
</tr>
<tr>
<td>184192</td>
<td>183751</td>
<td>99%</td>
<td>0.06K</td>
<td>2878</td>
<td>64</td>
<td></td>
<td>11512K</td>
<td>kmalloc-64</td>
</tr>
<tr>
<td>138618</td>
<td>138618</td>
<td>100%</td>
<td>0.94K</td>
<td>4077</td>
<td>34</td>
<td></td>
<td>130464K</td>
<td>xfs_inode</td>
</tr>
<tr>
<td>138602</td>
<td>138602</td>
<td>100%</td>
<td>0.21K</td>
<td>3746</td>
<td>37</td>
<td></td>
<td>29968K</td>
<td>xfs_ili</td>
</tr>
<tr>
<td>102116</td>
<td>99012</td>
<td>96%</td>
<td>0.55K</td>
<td>3647</td>
<td>28</td>
<td></td>
<td>58352K</td>
<td>radix_tree_node</td>
</tr>
<tr>
<td>97482</td>
<td>49093</td>
<td>50%</td>
<td>0.09K</td>
<td>2321</td>
<td>42</td>
<td></td>
<td>9284K</td>
<td>kmalloc-96</td>
</tr>
<tr>
<td>22695</td>
<td>20777</td>
<td>91%</td>
<td>0.05K</td>
<td>267</td>
<td>85</td>
<td></td>
<td>1068K</td>
<td>shared_policy_node</td>
</tr>
<tr>
<td>21312</td>
<td>21312</td>
<td>100%</td>
<td>0.86K</td>
<td>576</td>
<td>37</td>
<td></td>
<td>18432K</td>
<td>ext4_inode_cache</td>
</tr>
<tr>
<td>16288</td>
<td>14601</td>
<td>89%</td>
<td>0.25K</td>
<td>509</td>
<td>32</td>
<td></td>
<td>4072K</td>
<td>kmalloc-256</td>
</tr>
</tbody>
</table>

[...]
pcstat

• Show page cache residency by file:

# ./pcstat data0*

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Pages</th>
<th>Cached</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>data00</td>
<td>104857600</td>
<td>25600</td>
<td>25600</td>
<td>100.000</td>
</tr>
<tr>
<td>data01</td>
<td>104857600</td>
<td>25600</td>
<td>25600</td>
<td>100.000</td>
</tr>
<tr>
<td>data02</td>
<td>104857600</td>
<td>25600</td>
<td>4080</td>
<td>015.938</td>
</tr>
<tr>
<td>data03</td>
<td>104857600</td>
<td>25600</td>
<td>25600</td>
<td>100.000</td>
</tr>
<tr>
<td>data04</td>
<td>104857600</td>
<td>25600</td>
<td>16010</td>
<td>062.539</td>
</tr>
<tr>
<td>data05</td>
<td>104857600</td>
<td>25600</td>
<td>0</td>
<td>000.000</td>
</tr>
</tbody>
</table>

• Uses the mincore(2) syscall. Useful for database performance analysis.
perf_events (counters)

• Performance Monitoring Counters (PMCs):

```bash
$ perf list | grep -i hardware

cpu-cycles OR cycles [Hardware event]
stalled-cycles-frontend OR idle-cycles-frontend [Hardware event]
stalled-cycles-backend OR idle-cycles-backend [Hardware event]
instructions [Hardware event]
[...]
branch-misses [Hardware event]
bus-cycles [Hardware event]
L1-dcache-loads [Hardware cache event]
L1-dcache-load-misses [Hardware cache event]
[...]
rNNN (see 'perf list --help' on how to encode it) [Raw hardware event ...]
mem:<addr>[:access] [Hardware breakpoint]
```

• Identify CPU cycle breakdowns, esp. stall types
  – PMCs not enabled by-default in clouds (yet)
  – Can be time-consuming to use (CPU manuals)

• Use flame graphs to visualize sampled stack traces
perf_events CPU Flame Graph

Broken Java stacks (missing frame pointer)

Locks

GC

Time

Idle thread

epoll

Kernel TCP/IP
### tiptop

**tiptop -**
Tasks: 378 total, 15 displayed

<table>
<thead>
<tr>
<th>PID</th>
<th>%CPU</th>
<th>%SYS</th>
<th>P</th>
<th>Mcycle</th>
<th>Minstr</th>
<th>IPC</th>
<th>%MISS</th>
<th>%BMIS</th>
<th>%BUS</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>5910+</td>
<td>13.4</td>
<td>0.5</td>
<td>0</td>
<td>603.72</td>
<td>461.80</td>
<td>0.76</td>
<td>0.29</td>
<td>0.67</td>
<td>?</td>
<td>plugin-conf</td>
</tr>
<tr>
<td>3249+</td>
<td>11.4</td>
<td>3.5</td>
<td>1</td>
<td>394.35</td>
<td>551.39</td>
<td>1.40</td>
<td>0.10</td>
<td>0.19</td>
<td>?</td>
<td>gnome-term</td>
</tr>
<tr>
<td>17838</td>
<td>9.4</td>
<td>0.0</td>
<td>0</td>
<td>472.37</td>
<td>547.62</td>
<td>1.16</td>
<td>0.24</td>
<td>0.52</td>
<td>?</td>
<td>python</td>
</tr>
<tr>
<td>24782</td>
<td>8.4</td>
<td>7.9</td>
<td>0</td>
<td>47.99</td>
<td>39.76</td>
<td>0.83</td>
<td>0.09</td>
<td>1.02</td>
<td>?</td>
<td>find</td>
</tr>
<tr>
<td>2889+</td>
<td>4.0</td>
<td>0.5</td>
<td>5</td>
<td>114.78</td>
<td>30.42</td>
<td>0.27</td>
<td>2.38</td>
<td>1.81</td>
<td>?</td>
<td>enlightenmm</td>
</tr>
<tr>
<td>3311+</td>
<td>4.0</td>
<td>0.5</td>
<td>3</td>
<td>186.75</td>
<td>96.11</td>
<td>0.51</td>
<td>0.71</td>
<td>0.85</td>
<td>?</td>
<td>firefox</td>
</tr>
<tr>
<td>3534+</td>
<td>4.0</td>
<td>1.0</td>
<td>1</td>
<td>157.75</td>
<td>69.34</td>
<td>0.44</td>
<td>1.23</td>
<td>0.74</td>
<td>?</td>
<td>chromium-b</td>
</tr>
<tr>
<td>3518+</td>
<td>1.5</td>
<td>0.0</td>
<td>7</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>chromium-b</td>
</tr>
<tr>
<td>3307+</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>15.31</td>
<td>3.30</td>
<td>0.22</td>
<td>1.86</td>
<td>1.98</td>
<td>?</td>
<td>chromium-b</td>
</tr>
<tr>
<td>24717</td>
<td>1.0</td>
<td>1.0</td>
<td>3</td>
<td>13.29</td>
<td>13.60</td>
<td>1.02</td>
<td>0.05</td>
<td>0.65</td>
<td>?</td>
<td>tiptop</td>
</tr>
<tr>
<td>3635+</td>
<td>0.5</td>
<td>0.0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>chromium-b</td>
</tr>
</tbody>
</table>

- IPC by process, %MISS, %BUS
- Needs some love. perfmon2 library integration?
- Still can’t use it in clouds yet (needs PMCs enabled)
Model Specific Registers (MSRs), unlike PMCs, can be read by default in Xen guests:

- Timestamp clock, temp, power, ...
- Use rdmsr(1) from the msr-tools package to read them
- Uses include (https://github.com/brendangregg/msr-cloud-tools):

```
ec2-guest# ./showboost
[...]
TIME       C0_MCYC   C0_ACYC   UTIL  RATIO    MHz
06:11:35   6428553166 7457384521  51%   116%   2900
06:11:40   6349881107 7365764152  50%   115%   2899
06:11:45   6240610655 7239046277  49%   115%   2899
[...]
```

```
ec2-guest# ./cputemp 1
CPU1 CPU2 CPU3 CPU4
61 61 60 59
60 61 60 60
[...]
```
More Advanced Tools...

- Some others worth mentioning:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltrace</td>
<td>Library call tracer</td>
</tr>
<tr>
<td>ethtool</td>
<td>Mostly interface tuning; some stats</td>
</tr>
<tr>
<td>snmpget</td>
<td>SNMP network host statistics</td>
</tr>
<tr>
<td>lldptool</td>
<td>Can get LLDP broadcast stats</td>
</tr>
<tr>
<td>blktrace</td>
<td>Block I/O event tracer</td>
</tr>
<tr>
<td>/proc</td>
<td>Many raw kernel counters</td>
</tr>
<tr>
<td>pmu-tools</td>
<td>On- and off-core CPU counter tools</td>
</tr>
</tbody>
</table>
Advanced Tracers

• Many options on Linux:
  – perf_events, ftrace, eBPF, SystemTap, ktap, LTTng, dtrace4linux, sysdig

• Most can do static and dynamic tracing
  – Static: pre-defined events (tracepoints)

• Many are in-development.
  – I’ll summarize their state later...
Linux Observability Tools

- strace
- lsof
- pcstat
- pidstat
- perf
- ftrace
- stap
- ktap
- ebpf
- dtrace
- lttng
- netstat
- sysdig
- perf
- mpstat
- CPU Interconnect
- top
- ps
- pidstat
- vmstat
- slabtop
- free
- perf
- tiptop
- iostat
- iotop
- blktrace
- I/O Controller
- Disk
- Disk
- Swap
- Network Controller
- Port
- Port
- ethtool
- snmpget
- lldptool

Hardware

Various:
- sar
- /proc
dstat

Memory Bus

CPU 1

DRAM
Benchmarking Tools
Benchmarking Tools

• Multi:
  – UnixBench, Imbench, sysbench, perf bench

• FS/disk:
  – dd, hdparm, fio

• App/lib:
  – ab, wrk, jmeter, openssl

• Networking:
  – ping, hping3, iperf, ttcp, traceroute, mtr, pchar
Active Benchmarking

• Most benchmarks are misleading or wrong
  – You benchmark A, but actually measure B, and conclude that you measured C

• Active Benchmarking
  1. Run the benchmark for hours
  2. While running, analyze and confirm the performance limiter using observability tools

• We just covered those tools – use them!
Imbench

- CPU, memory, and kernel micro-benchmarks
- Eg, memory latency by stride size:

```bash
$ lat_mem_rd 100m 128 > out.latencies
some R processing...
```

![Graph showing memory latency by size](image)

- L1 cache
- L2 cache
- L3 cache
- Main Memory

Latency (ns)

Size (Kbytes)
fio

• FS or disk I/O micro-benchmarks

$ fio --name=seqwrite --rw=write --bs=128k --size=122374m

[...]
seqwrite: (groupid=0, jobs=1): err= 0: pid=22321
write: io=122374MB, bw=840951KB/s, iops=6569 , runt=149011msec
  clat (usec): min=41 , max=133186 , avg=148.26, stdev=1287.17
  lat (usec): min=44 , max=133188 , avg=151.11, stdev=1287.21
  bw (KB/s) : min=10746, max=1983488, per=100.18%, avg=842503.94,
  stdev=262774.35
  cpu : usr=2.67%, sys=43.46%, ctx=14284, majf=1, minf=24
  IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
  submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
  complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
  issued r/w/d: total=0/978992/0, short=0/0/0
  lat (usec): 50=0.02%, 100=98.30%, 250=1.06%, 500=0.01%, 750=0.01%
  lat (usec): 1000=0.01%
  lat (msec): 2=0.01%, 4=0.01%, 10=0.25%, 20=0.29%, 50=0.06%
  lat (msec): 100=0.01%, 250=0.01%

• Results include basic latency distribution
pchar

• Traceroute with bandwidth per hop!

$ pchar 10.71.83.1
[...]
4: 10.110.80.1 (10.110.80.1)
  Partial loss: 0 / 5 (0%)
  Partial char: rtt = 9.351109 ms, (b = 0.004961 ms/B), r2 = 0.184105
                stddev rtt = 4.967992, stddev b = 0.006029
  Partial queueing: avg = 0.000000 ms (0 bytes)
  Hop char: rtt = --.--- ms, bw = 1268.975773 Kbps
  Hop queueing: avg = 0.000000 ms (0 bytes)
5: 10.193.43.181 (10.193.43.181)
  Partial loss: 0 / 5 (0%)
  Partial char: rtt = 25.461597 ms, (b = 0.011934 ms/B), r2 = 0.228707
                stddev rtt = 10.426112, stddev b = 0.012653
  Partial queueing: avg = 0.000000 ms (0 bytes)
  Hop char: rtt = 16.110487 ms, bw = 1147.210397 Kbps
  Hop queueing: avg = 0.000000 ms (0 bytes)
[...]

• Needs love. Based on pathchar (Linux 2.0.30).
Tuning Tools
Tuning Tools

• Generic interfaces:
  – sysctl, /sys

• Many areas have custom tuning tools:
  – Applications: their own config
  – CPU/scheduler: nice, renice, taskset, ulimit, chcpu
  – Storage I/O: tune2fs, ionice, hdparm, blockdev, ...
  – Network: ethtool, tc, ip, route
  – Dynamic patching: stap, kpatch
Tuning Methods

• Scientific Method:
  1. Question
  2. Hypothesis
  3. Prediction
  4. Test
  5. Analysis

• Any *observational* or *benchmarking* tests you can try before tuning?

• Consider risks, and see previous tools
Tuning Tools

Operating System
- Applications
  - System Libraries
    - VFS
    - Sockets
    - File Systems
    - TCP/UDP
    - Volume Manager
    - Socket Buffers
    - Ethernet
- Device Drivers
  - Block Device Interface
  - I/O Bus
- I/O Bridge
  - I/O Controller
    - Disk
    - Disk
    - Swap
- Interface Transports
  - Network Controller
    - Port
    - Port
- Expander Interconnect
- Memory Bus
- CPU Interconnect
  - CPU
  - Memory
  - Various:
    - syscall
    - /sys
    - chcpu
    - nice
    - renice
    - taskset
    - /dev/cpuset
    - ulimit

Linux Kernel
- App Config
  - tune2fs
- stap
- kpatch
- ionice
- ip
- route
- MegaCli
- hdiutil
- blockdev
- swapon
- tc
- ip
Static Tools
Static Tools

• Static Performance Tuning: check the static state and configuration of the system
  – CPU types
  – Storage devices
  – File system capacity
  – File system and volume configuration
  – Route table
  – State of hardware

• What can be checked on a system without load
Static Tools
Tracing
Tracing Frameworks: Tracepoints

- Statically placed at logical places in the kernel
- Provides key event details as a “format” string
Tracing Frameworks: + probes

- **kprobes**: dynamic kernel tracing
  - function calls, returns, line numbers
- **uprobes**: dynamic user-level tracing
Tracing Tools

- Options:
  - ftrace
  - perf_events
  - eBPF
  - SystemTap
  - ktap
  - LTTng
  - dtrace4linux
  - Oracle Linux DTrace
  - sysdig

- Too many choices, and many still in-development
Imagine Linux with Tracing

• With a programmable tracer, high level tools can be written, such as:
  – iosnoop
  – iolatency
  – opensnoop
  – ...

iosnoop

- Block I/O (disk) events with latency:

```bash
# ./iosnoop -ts
Tracing block I/O. Ctrl-C to end.
STARTs   ENDS   COMM     PID   TYPE DEV    BLOCK     BYTES   LATms
5982800.302061 5982800.302679 supervise 1809  W 202,1  17039600 4096   0.62
5982800.302423 5982800.302842 supervise 1809  W 202,1  17039608 4096   0.42
5982800.304962 5982800.305446 supervise 1801  W 202,1  17039616 4096   0.48
5982800.305250 5982800.305676 supervise 1801  W 202,1  17039624 4096   0.43
[...]
```

```bash
# ./iosnoop -h
USAGE: iosnoop [-hQst] [-d device] [-i iotype] [-p PID] [-n name] [duration]
-d device       # device string (eg, "202,1)
-i iotype       # match type (eg, "*R*" for all reads)
-n name         # process name to match on I/O issue
-p PID          # PID to match on I/O issue
-Q              # include queueing time in LATms
-s              # include start time of I/O (s)
-t              # include completion time of I/O (s)
-h              # this usage message
duration       # duration seconds, and use buffers
[...]
```
# ./iolatency
Tracing block I/O. Output every 1 seconds. Ctrl-C to end.

<table>
<thead>
<tr>
<th>&gt;= (ms) .. &lt; (ms)</th>
<th>I/O</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>2104</td>
<td>#</td>
</tr>
<tr>
<td>1 -&gt; 2</td>
<td>280</td>
<td>#</td>
</tr>
<tr>
<td>2 -&gt; 4</td>
<td>2</td>
<td>#</td>
</tr>
<tr>
<td>4 -&gt; 8</td>
<td>0</td>
<td>#</td>
</tr>
<tr>
<td>8 -&gt; 16</td>
<td>202</td>
<td>#</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&gt;= (ms) .. &lt; (ms)</th>
<th>I/O</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>1144</td>
<td>#</td>
</tr>
<tr>
<td>1 -&gt; 2</td>
<td>267</td>
<td>#</td>
</tr>
<tr>
<td>2 -&gt; 4</td>
<td>10</td>
<td>#</td>
</tr>
<tr>
<td>4 -&gt; 8</td>
<td>5</td>
<td>#</td>
</tr>
<tr>
<td>8 -&gt; 16</td>
<td>248</td>
<td>#</td>
</tr>
<tr>
<td>16 -&gt; 32</td>
<td>601</td>
<td>#</td>
</tr>
<tr>
<td>32 -&gt; 64</td>
<td>117</td>
<td>#</td>
</tr>
</tbody>
</table>

[...]

- Block I/O (disk) latency distributions:
**opensnoop**

- Trace open() syscalls showing filenames:

```plaintext
# ./opensnoop -t
Tracing open()s. Ctrl-C to end.

<table>
<thead>
<tr>
<th>TIMEs</th>
<th>COMM</th>
<th>PID</th>
<th>FD</th>
<th>FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4345768.332626</td>
<td>postgres</td>
<td>23886</td>
<td>0x8</td>
<td>/proc/self/oom_adj</td>
</tr>
<tr>
<td>4345768.333923</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>global/pg_fileno.init</td>
</tr>
<tr>
<td>4345768.33971</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>global/pg_internal.init</td>
</tr>
<tr>
<td>4345768.34813</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/PG_VERSION</td>
</tr>
<tr>
<td>4345768.34877</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/pg_fileno.init</td>
</tr>
<tr>
<td>4345768.34891</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/pg_fileno.init</td>
</tr>
<tr>
<td>4345768.35821</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/11725</td>
</tr>
<tr>
<td>4345768.347911</td>
<td>svstat</td>
<td>24649</td>
<td>0x4</td>
<td>supervise/ok</td>
</tr>
<tr>
<td>4345768.347921</td>
<td>svstat</td>
<td>24649</td>
<td>0x4</td>
<td>supervise/status</td>
</tr>
<tr>
<td>4345768.350340</td>
<td>stat</td>
<td>24651</td>
<td>0x3</td>
<td>/etc/ld.so.cache</td>
</tr>
<tr>
<td>4345768.350372</td>
<td>stat</td>
<td>24651</td>
<td>0x3</td>
<td>/lib/x86_64-linux-gnu/libselinux...</td>
</tr>
<tr>
<td>4345768.350460</td>
<td>stat</td>
<td>24651</td>
<td>0x3</td>
<td>/lib/x86_64-linux-gnu/libc.so.6</td>
</tr>
<tr>
<td>4345768.350526</td>
<td>stat</td>
<td>24651</td>
<td>0x3</td>
<td>/lib/x86_64-linux-gnu/libdl.so.2</td>
</tr>
<tr>
<td>4345768.350981</td>
<td>stat</td>
<td>24651</td>
<td>0x3</td>
<td>/proc/filesystems</td>
</tr>
<tr>
<td>4345768.351182</td>
<td>stat</td>
<td>24651</td>
<td>0x3</td>
<td>/etc/nsswitch.conf</td>
</tr>
</tbody>
</table>
[...]```
funcgraph

• Trace a graph of kernel code flow:

```bash
# ./funcgraph -Htp 5363 vfs_read
Tracing "vfs_read" for PID 5363... Ctrl-C to end.
# tracer: function_graph

# | TIME          | CPU | DURATION | FUNCTION CALLS |
# |               |     |          |               |
4346366.073832 | 0   |          |               |
4346366.073834 | 0   |          |               |
4346366.073834 | 0   |          |               |
4346366.073834 | 0   |          |               |
4346366.073835 | 0   | 0.153 us |               |
4346366.073836 | 0   | 0.947 us |               |
4346366.073836 | 0   | 0.066 us |               |
4346366.073836 | 0   | 0.080 us |               |
4346366.073837 | 0   | 2.174 us |               |
4346366.073837 | 0   | 2.656 us |               |
4346366.073837 | 0   |          |               |
4346366.073837 | 0   | 0.060 us |               |
[...]
```

```c
vfs_read() {
    rw_verify_area() {
        security_file_permission() {
            apparmor_file_permission() {
                common_file_perm();
            }
            __fsnotify_parent();
            fsnotify();
        }
    }
    tty_read() {
        tty_paranoia_check();
    }
}```
kprobe

• Dynamically trace a kernel function call or return, with variables, and in-kernel filtering:

```bash
# ./kprobe 'p:open do_sys_open filename=+0(%si):string' 'filename ~ "*stat"'
Tracing kprobe myopen. Ctrl-C to end.
  postgres-1172 [000] d... 6594028.787166: open: (do_sys_open
+0x0/0x220) filename="pg_stat_tmp/pgstat.stat"
  postgres-1172 [001] d... 6594028.797410: open: (do_sys_open
+0x0/0x220) filename="pg_stat_tmp/pgstat.stat"
  postgres-1172 [001] d... 6594028.797467: open: (do_sys_open
+0x0/0x220) filename="pg_stat_tmp/pgstat.stat"
^C
Ending tracing...
```

• Add -s for stack traces; -p for PID filter in-kernel.
• Quickly confirm kernel behavior; eg: did a tunable take effect?
Imagine Linux with Tracing

• These tools aren’t using dtrace4linux, SystemTap, ktap, or any other add-on tracer
• These tools use **existing Linux capabilities**
  – No extra kernel bits, not even kernel debuginfo
  – Just Linux’s built-in **ftrace** profiler
  – Demoed on **Linux 3.2**
• Solving real issues **now**
ftrace

- Added by Steven Rostedt and others since 2.6.27
- Already enabled on our servers (3.2+)
  - CONFIG_FTRACE, CONFIG_FUNCTION_PROFILER, ...
  - Use directly via /sys/kernel/debug/tracing
- My front-end tools to aid usage
  - https://github.com/brendangregg/perf-tools
  - Unsupported hacks: see WARNINGs
  - Also see the trace-cmd front-end, as well as perf
- lwn.net: “Ftrace: The Hidden Light Switch”
My perf-tools (so far...)
Tracing Summary

- ftrace
- perf_events
- eBPF
- SystemTap
- ktap
- LTTng
- dtrace4linux
- sysdig
perf_events

• aka “perf” command

• **In Linux.** Add from `linux-tools-common`, ...

• Powerful multi-tool and profiler
  – interval sampling, CPU performance counter events
  – user and kernel dynamic tracing
  – kernel line tracing and local variables (debuginfo)
  – kernel filtering, and in-kernel counts (perf stat)

• Not very programmable, yet
  – limited kernel summaries. May improve with eBPF.
perf_events Example

```bash
# perf record -e skb:consume_skb -ag
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.065 MB perf.data (~2851 samples) ]
# perf report
[...]
74.42% swapper [kernel.kallsyms] [k] consume_skb
   --- consume_skb
      arp_process
      arp_rcv
      __netif_receive_skb_core
      __netif_receive_skb
      netif_receive_skb
      virtnet_poll
      net_rx_action
      __do_softirq
      irq_exit
      do_IRQ
      ret_from_intr
      default_idle
      cpu_idle
      start_secondary
```

Summarizing stack traces for a tracepoint

perf_events can do many things – hard to pick just one example
eBPF

- Extended BPF: programs on tracepoints
  - High performance filtering: JIT
  - In-kernel summaries: maps
- Linux in 3.18? Enhance perf_events/ftrace/...?

# ./bitesize 1
writing bpf-5 -> /sys/kernel/debug/tracing/events/block/block_rq_complete/filter

<table>
<thead>
<tr>
<th>I/O sizes:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kbytes</td>
<td>Count</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>131</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>32</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>1</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>46</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>0</td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>15</td>
</tr>
</tbody>
</table>

[...]
SystemTap

• Fully programmable, fully featured
• Compiles tracing programs into kernel modules
  – Needs a compiler, and takes time
• “Works great on Red Hat”
  – I keep trying on other distros and have hit trouble in the past; make sure you are on the latest version.
  – I’m liking it a bit more after finding ways to use it without kernel debuginfo (a difficult requirement in our environment). Work in progress.
• Ever be mainline?
ktap

• Sampling, static & dynamic tracing
• Lightweight, simple. Uses bytecode.
• Suited for embedded devices
• Development appears suspended after suggestions to integrate with eBPF (which itself is in development)
• ktap + eBPF would be awesome: easy, lightweight, fast. Likely?
sysdig

• sysdig: Innovative new tracer. Simple expressions:

```plaintext
sysdig fd.type=file and evt.failed=true
sysdig evt.type=open and fd.name contains /etc
sysdig -p"%proc.name %fd.name" "evt.type=accept and proc.name!=httpd"
```

• Replacement for strace? (or “perf trace” will)

• Programmable “chisels”. Eg, one of mine:

```plaintext
# sysdig -c fileslower 1
TIME                  PROCESS  TYPE      LAT(ms)  FILE
2014-04-13 20:40:43.973 cksum  read      2 /mnt/partial.0.0
2014-04-13 20:40:44.187 cksum  read      1 /mnt/partial.0.0
2014-04-13 20:40:44.689 cksum  read      2 /mnt/partial.0.0
[...]
```

• Currently syscall and user-level processing only. It is optimized, but I’m not sure it can be enough for kernel tracing
Present & Future

• Present:
  – ftrace can serve many needs today
  – perf_events some more, esp. with debuginfo
  – ad hoc SystemTap, ktap, ... as needed

• Future:
  – ftrace/perf_events/ktap with eBPF, for a fully featured and mainline tracer?
  – One of the other tracers going mainline?
The Tracing Landscape, Oct 2014

Ease of use

(my opinion)

Scope & Capability

(less brutal)

sysdig

dtrace4L.

(ktap

perf

(stap

ftrace

eBPF

Stage of Development

(brutal)

(alpha) → (mature)
In Summary...

• Plus diagrams for benchmarking, tuning, tracing
• Try to start with the questions (methodology), to help guide your use of the tools
• I hopefully turned some unknown unknowns into known unknowns
References & Links

- Systems Performance: Enterprise and the Cloud, Prentice Hall, 2014
- http://www.brendangregg.com/perf.html#FlameGraphs
- nicstat: http://sourceforge.net/projects/nicstat/
- tiptop: http://tiptop.gforge.inria.fr/
- ftrace & perf-tools
  - https://github.com/brendangregg/perf-tools
  - http://lwn.net/Articles/608497/
- MSR tools: https://github.com/brendangregg/msr-cloud-tools
- pcstat: https://github.com/tobert/pcstat
- eBPF: http://lwn.net/Articles/603983/
- ktap: http://www.ktap.org/
- SystemTap: https://sourceware.org/systemtap/
- sysdig: http://www.sysdig.org/
- Tux by Larry Ewing; Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.
Thanks

• Questions?
• http://slideshare.net/brendangregg
• http://www.brendangregg.com
• bgregg@netflix.com
• @brendangregg