Linux Performance Tools

Brendan Gregg
Senior Performance Architect
Performance Engineering Team
bgregg@netflix.com
@brendangregg
This Tutorial

• A tour of many Linux performance tools
  – To show you what can be done
  – With guidance for how to do it

• This includes objectives, discussion, live demos
  – See the video of this tutorial
Massive AWS EC2 Linux cloud
  - 10s of thousands of cloud instances
FreeBSD for content delivery
  - ~33% of US Internet traffic at night
Over 50M subscribers
  - Recently launched in ANZ
Use Linux server tools as needed
  - After cloud monitoring (Atlas, etc.) and instance monitoring (Vector) tools
Agenda

• Methodologies
• Tools
• Tool Types:
  – Observability
  – Benchmarking
  – Tuning
  – Static
• Profiling
• Tracing
Methodologies
Methodologies

• Objectives:
  – Recognize the Streetlight Anti-Method
  – Perform the Workload Characterization Method
  – Perform the USE Method
  – Learn how to start with the questions, before using tools
  – Be aware of other methodologies
My system is slow...

DEMO

&

DISCUSSION
Methodologies

• There are dozens of performance tools for Linux
  – Packages: sysstat, procps, coreutils, …
  – Commercial products
• Methodologies can provide guidance for choosing and using tools effectively
• A starting point, a process, and an ending point
Anti-Methodologies

• The lack of a deliberate methodology…
Street Light Anti-Method

1. Pick observability tools that are:
   - Familiar
   - Found on the Internet
   - Found at random

2. Run tools

3. Look for obvious issues
Drunk Man Anti-Method

• Tune things at random until the problem goes away
Blame Someone Else Anti-Method

1. Find a system or environment component you are not responsible for
2. Hypothesize that the issue is with that component
3. Redirect the issue to the responsible team
4. When proven wrong, go to 1
Actual Methodologies

- Problem Statement Method
- Workload Characterization Method
- USE Method
- Off-CPU Analysis
- CPU Profile Method
- RTFM Method
- Active Benchmarking (*covered later*)
- Static Performance Tuning (*covered later*)
- ...

Problem Statement Method

1. What makes you **think** there is a performance problem?
2. Has this system **ever** performed well?
3. What has **changed** recently? (Software? Hardware? Load?)
4. Can the performance degradation be expressed in terms of **latency** or run time?
5. Does the problem affect **other** people or applications (or is it just you)?
6. What is the **environment**? Software, hardware, instance types? Versions? Configuration?
Workload Characterization Method

1. **Who** is causing the load? PID, UID, IP addr, ...
2. **Why** is the load called? code path, stack trace
3. **What** is the load? IOPS, tput, type, r/w
4. **How** is the load changing over time?
The USE Method

• For every resource, check:
  1. Utilization
  2. Saturation
  3. Errors

• Definitions:
  – Utilization: busy time
  – Saturation: queue length or queued time
  – Errors: easy to interpret (objective)

• Helps if you have a functional (block) diagram of your system / software / environment, showing all resources

Start with the questions, then find the tools
USE Method for Hardware

- For every resource, check:
  1. Utilization
  2. Saturation
  3. Errors
- Including busses & interconnects
## Linux USE Method Example

**USE Method: Linux Performance Checklist**

The **USE Method** provides a strategy for performing a complete check of system health, identifying common bottlenecks and errors. For each system resource, metrics for utilization, saturation and errors are identified and checked. Any issues discovered are then investigated using further strategies.

This is an example USE-based metric list for Linux operating systems (e.g., Ubuntu, CentOS, Fedora). This is primarily intended for system administrators of the physical systems, who are using command line tools. Some of these metrics can be found in remote monitoring tools.

### Physical Resources

http://www.brendangregg.com/USEmethod/use-linux.html

<table>
<thead>
<tr>
<th>component</th>
<th>type</th>
<th>metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>utilization</td>
<td>system-wide: vmstat 1, &quot;us&quot; + &quot;sy&quot; + &quot;st&quot;; sar -u, sum fields except &quot;%idle&quot; and &quot;%iowait&quot;; dstat -c, sum fields except &quot;idle&quot; and &quot;wait&quot;; per-cpu: mpstat -P ALL 1, sum fields except &quot;%idle&quot; and &quot;%iowait&quot;; sar -p ALL, same as mpstat; per-process: top, &quot;%CPU&quot;; htop, &quot;CPU%&quot;; ps -o pcpu; pidstat 1, &quot;%CPU&quot;, per-kernel-thread: top/htop (&quot;K&quot; to toggle), where VIRT == 0 (heuristic). [1]</td>
</tr>
<tr>
<td>CPU</td>
<td>saturation</td>
<td>system-wide: vmstat 1, &quot;r&quot; &gt; CPU count [2]; sar -q, &quot;runq-sz&quot; &gt; CPU count; dstat -p, &quot;run&quot; &gt; CPU count; per-process: /proc/PID/schedstat 2nd field (sched_info.run_delay); perf sched latency (shows &quot;Average&quot; and &quot;Maximum&quot; delay per-schedule); dynamic tracing, eg, SystemTap schedtimes.stp &quot;queued(us)&quot; [3]</td>
</tr>
<tr>
<td>CPU</td>
<td>errors</td>
<td>perf (LPE) if processor specific error events (CPC) are available; eg, AMD64's &quot;04Ah Single-bit ECC Errors Recorded by Scrubber&quot; [4]</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>utilization</td>
<td>system-wide: free -m, &quot;Mem:&quot; (main memory), &quot;Swap:&quot; (virtual memory); vmstat 1, &quot;free&quot; (main memory), &quot;swap&quot; (virtual memory); sar -r, &quot;%memused&quot;; dstat -m, &quot;free&quot;; slabtop -s c for kmem slab usage; per-process: top/htop, &quot;RES&quot; (resident main memory), &quot;VIRT&quot; (virtual memory). &quot;Mem&quot; for system-wide summary</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>saturation</td>
<td>system-wide: vmstat 1, &quot;si&quot;/&quot;so&quot; (swapping); sar -B, &quot;psscan&quot; + &quot;psscanl&quot; (scanning); sar -w; per-process: 10th field (min_flt) from /proc/PID/stat for minor-fault rate, or dynamic tracing [5]; OOM killer: dmesg</td>
</tr>
<tr>
<td>Memory</td>
<td>errors</td>
<td>dmesg for physical failures; dynamic tracing, eg, SystemTap uprobes for failed malloc0s</td>
</tr>
</tbody>
</table>
Off-CPU Analysis

Thread State Transition Diagram

Off-CPU Profiling

On-CPU Profiling

Off-CPU Profiling (everything else)
CPU Profile Method

1. Take a CPU profile
2. Understand all software in profile > 1%
   - Discovers a wide range of performance issues by their CPU usage
   - Narrows software to study
RTFM Method

• How to understand performance tools or metrics:
  1. Man pages
  2. Books
  3. Web search
  4. Co-workers
  5. Prior talk slides/video (this one!)
  6. Support services
  7. Source code
  8. Experimentation
  9. Social
Tools
Tools

• Objectives:
  – Perform the USE Method for resource utilization
  – Perform Workload Characterization for disks, network
  – Perform the CPU Profile Method using flame graphs
  – Have exposure to various observability tools:
    • Basic: vmstat, iostat, mpstat, ps, top, …
    • Intermediate: tcpdump, netstat, nicstat, pidstat, sar, …
    • Advanced: ss, slaptop, perf_events, …
  – Perform Active Benchmarking
  – Understand tuning risks
  – Perform Static Performance Tuning
Command Line Tools

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

$ 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  ...

Kernel

/proc, /sys, ...

/graphic
## 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
One way to print load averages:

- A measure of resource demand: CPUs + disks
  - Other OSes only show CPUs: easier to interpret
- Exponentially-damped moving averages
- 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

$ uptime
07:42:06 up  8:16,  1 user,  load average: 2.27, 2.84, 2.91
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  S    0  0.0   0:00.00 kthreadd
```

- %CPU is summed across all CPUs
- Can miss short-lived processes (atop won’t)
- Can consume noticeable CPU to read /proc
htop

<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>1</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>messagebu</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</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 --e 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>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>
<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>
• Process status listing (eg, “ASCII art forest”):

```
$ ps -ef f

<table>
<thead>
<tr>
<th>USER</th>
<th>SZ</th>
<th>RSS</th>
<th>MINFLT</th>
<th>MAJFLT</th>
<th>%CPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>6085</td>
<td>2272</td>
<td>11928</td>
<td>24</td>
<td>0.0</td>
<td>/sbin/init</td>
</tr>
</tbody>
</table>
```

• Custom fields:

```
$ ps -eo user,sz,rss,minflt,majflt,pcpu,args

<table>
<thead>
<tr>
<th>USER</th>
<th>SZ</th>
<th>RSS</th>
<th>MINFLT</th>
<th>MAJFLT</th>
<th>%CPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>6085</td>
<td>2272</td>
<td>11928</td>
<td>24</td>
<td>0.0</td>
<td>/sbin/init</td>
</tr>
</tbody>
</table>
```
vmstat

- Virtual memory statistics and more:

```bash
$ 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  0  205 186  46  13  0  0
 8  0   0  1617  149  552    0  0   0  8  210 435  39  21  0  0
 8  0   0  1589  149  552    0  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.

```
$ iostat -xmdz 1

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

Device:  rrqm/s  wrqm/s  r/s  w/s  rMB/s  wMB/s  \ ...
  xvda  0.00  0.00  0.00  0.00  0.00  0.00  / ...
  xvdb  213.00  0.00  15299.00  0.00  338.17  0.00  / ...
  xvdc  129.00  0.00  15271.00  3.00  336.65  0.01  / ...
  md0  0.00  0.00  31082.00  3.00  678.45  0.01  / ...
```

Workload

- Very useful set of stats

```
... \ avgqu-sz  await  r-await  w-await  svctm  %util 
... /     0.00  0.00  0.00  0.00  0.00  0.00
... \ 126.09  8.22  8.22  0.00  0.06  86.40
... /  99.31  6.47  6.47  0.00  0.06  86.00
... \ 0.00  0.00  0.00  0.00  0.00  0.00
```

Resulting Performance
mpstat

- Multi-processor statistics, per-CPU:

\[\text{\$ mpstat \(-P\ \text{ALL} \ 1\)}\]

<table>
<thead>
<tr>
<th>Time</th>
<th>CPU</th>
<th>%usr</th>
<th>%nice</th>
<th>%sys</th>
<th>%iowait</th>
<th>%irq</th>
<th>%soft</th>
<th>%steal</th>
<th>%guest</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:06:43 PM</td>
<td>all</td>
<td>53.45</td>
<td>0.00</td>
<td>3.77</td>
<td>0.00</td>
<td>0.00</td>
<td>0.39</td>
<td>0.13</td>
<td>0.00</td>
<td>42.26</td>
</tr>
<tr>
<td>08:06:44 PM</td>
<td>0</td>
<td>49.49</td>
<td>0.00</td>
<td>3.03</td>
<td>0.00</td>
<td>0.00</td>
<td>1.01</td>
<td>1.01</td>
<td>0.00</td>
<td>45.45</td>
</tr>
<tr>
<td>08:06:44 PM</td>
<td>1</td>
<td>51.61</td>
<td>0.00</td>
<td>4.30</td>
<td>0.00</td>
<td>0.00</td>
<td>2.15</td>
<td>0.00</td>
<td>0.00</td>
<td>41.94</td>
</tr>
<tr>
<td>08:06:44 PM</td>
<td>2</td>
<td>58.16</td>
<td>0.00</td>
<td>7.14</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.02</td>
<td>0.00</td>
<td>33.67</td>
</tr>
<tr>
<td>08:06:44 PM</td>
<td>3</td>
<td>54.55</td>
<td>0.00</td>
<td>5.05</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>40.40</td>
</tr>
<tr>
<td>08:06:44 PM</td>
<td>4</td>
<td>47.42</td>
<td>0.00</td>
<td>3.09</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>49.48</td>
</tr>
<tr>
<td>08:06:44 PM</td>
<td>5</td>
<td>65.66</td>
<td>0.00</td>
<td>3.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>31.31</td>
</tr>
<tr>
<td>08:06:44 PM</td>
<td>6</td>
<td>50.00</td>
<td>0.00</td>
<td>2.08</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>47.92</td>
</tr>
</tbody>
</table>

- Look for unbalanced workloads, hot CPUs.
free

- Main memory usage:

```
$ free -m

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

- buffers: block device I/O cache
- cached: virtual page cache
Latency is now much higher...

DEMO

&

DISCUSSION
Observability Tools: Basic
Observability Tools: Intermediate

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

- System call tracer:
  
  ```
  $ 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:

```bash
$ 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)

08:57:52 PM   TGID   TID   %usr %system %guest %CPU   CPU   Command
08:57:54 PM   5738   -     484.75 39.83  0.00  524.58 1     java
08:57:54 PM   -     5817   0.85  0.00  0.00   0.85  2   |__java
08:57:54 PM   -     5931   1.69  1.69  0.00   3.39  4   |__java
08:57:54 PM   -     5981   0.85  0.00  0.00   0.85  7   |__java
08:57:54 PM   -     5990   0.85  0.00  0.00   0.85  4   |__java
[...]
```

```
$ pidstat -d 1
[
```

```
08:58:27 PM   PID   kB_rd/s kB_wr/s kB_ccwr/s Command
08:58:28 PM   5738   0.00   815.69  0.00   java
[...]
```

```
swapon

- Show swap device usage:

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

- If you have swap enabled...
lsof

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

```
# 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)
postgres  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)
postgres  2892  app1  7u  IPv4    15679      0t0  TCP localhost:5433->localhost:33885 (ESTABLISHED)
[...]
```
sar

• System Activity Reporter. Many stats, eg:

```
$ 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
• In cloud environments, you are probably using a monitoring product, developed in-house or commercial.
  – We develop Atlas for cloud-wide monitoring, and Vector for instance-level analysis (both NetflixOSS)
  – Same method applies…
How does your monitoring tool measure these?
App is taking forever...

DEMO

&

DISCUSSION
Observability Tools: Intermediate

- strace
- lsof
- netstat
- pidstat
- uptime
- mpstat
- top ps
- vmstat
- free
- tcpdump
- nicstat
- netstat
- swapon
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:

```
$ 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
```

```
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*
• Block device I/O (disk) by process:

```
$ iotop
Total DISK READ: 50.47 M/s | Total DISK WRITE: 59.21 M/s

TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN   IO>    COMMAND
959  be/4  root       0.00 B/s    0.00 B/s  0.00 % 99.99 % [flush-202:1]
6641 be/4  root       50.47 M/s   82.60 M/s  0.00 % 32.51 % java -Dnop -X
   be/4  root       0.00 B/s    0.00 B/s  0.00 %  0.00 % init
   be/4  root       0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
   be/4  root       0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
   be/4  root       0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/0:0]
   be/4  root       0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/u:0]
   rt/4  root       0.00 B/s    0.00 B/s  0.00 %  0.00 % [migration/0]

[...]
```

• Needs kernel support enabled
  – CONFIG_TASK_IO_ACCOUNTING
### Slabtop

- **Kernel slab allocator memory usage:**

```bash
$ 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 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>3565575</td>
<td>3565575</td>
<td>100%</td>
<td>0.10K</td>
<td>91425</td>
<td>39</td>
<td>365700K</td>
<td>buffer_head</td>
</tr>
<tr>
<td>314916</td>
<td>314066</td>
<td>99%</td>
<td>0.19K</td>
<td>14996</td>
<td>21</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>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>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>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>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>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>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>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>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

• Provides the "perf" command
• In Linux source code: tools/perf
  – Usually pkg added by linux-tools-common, etc.
• **Multi-tool** with many capabilities
  – CPU profiling
  – PMC profiling
  – Static & dynamic tracing
• *Covered later in Profiling & Tracing*
• IPC by process, %MISS, %BUS
• Needs some love. perfmon2 library integration?
• Still can’t use it in clouds yet (needs PMCs enabled)
rdmsr

- 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
  - From 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
[...]```

Real CPU MHz

CPU Temperature
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)
  – Dynamic: instrument any software (kprobes, uprobes).
    Custom metrics on-demand. Catch all.

• Many are in-development
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
Benchmarking

• ~100% of benchmarks are wrong
• Results are usually misleading: you benchmark A, but actually measure B, and conclude you measured C
• Common mistakes:
  – Testing the wrong target: eg, FS cache instead of disk
  – Choosing the wrong target: eg, disk instead of FS cache
    … doesn’t resemble real world usage
  – Invalid results: eg, bugs
• The energy needed to refute benchmarks is multiple orders of magnitude bigger than to run them
Active Benchmarking (Method)

1. Run the benchmark for hours
2. While running, analyze and confirm the performance limiter using observability tools
   - Disk benchmark: run iostat, …
   - CPU benchmark: run pidstat, perf, flame graphs, …
   - …

• Answer the question: why isn't the result 10x?

We just covered the observability 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...
```

![Diagram showing cache levels and memory latency](chart.png)
fio

- FS or disk I/O micro-benchmarks

```bash
$ 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!

```sh
$ 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
Static Tools
Static Tools

• Static Performance Tuning: check the static state and configuration of the system
  – CPU types & flags
  – CPU frequency scaling config
  – Storage devices
  – File system capacity
  – File system and volume configuration
  – Route table
  – State of hardware
  – etc.

• What can be checked on a system without load
• Methodology by Richard Elling (2000)
CPU Types & Flags

$ more /proc/cpuinfo

processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 42
model name : Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
stepping : 7
microcode : 0x1a
cpu MHz : 1600.000
cache size : 6144 KB
physical id : 0
siblings : 4
core id : 0
cpu cores : 4
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuID level : 13
wp : yes

flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc a

c
ch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni
pc

c
clmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2

c
x2apic popcnt tsc_deadline_timer aes xsave avx 1ahf_lm ida arat epb xsaveopt pln pts
d

c
dtherm tpr_shadow vnmi flexpriority ept vpid
[...]

CPU speed still matters
CPU Frequency Scaling

• Kernel may be configured to dynamically modify CPU frequency:

```bash
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
3101000 3100000 2900000 2700000 2500000 2300000 2100000 1900000 1700000 1600000
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
ondemand
```

- See Documentation/cpu-freq/governors.txt, and scaling_governor == performance

• Not to be confused with Intel Turbo Boost (which is H/W)


### Storage Devices

```bash
# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Type: Direct-Access           ANSI   SCSI revision: 05
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: PLDS     Model: DVD-RW DH16ABSH  Rev: YL32
  Type: CD-ROM                 ANSI   SCSI revision: 05

# lsscsi
[0:0:0:0] disk  ATA   ST3320413AS  JC65  /dev/sda
[1:0:0:0] cd/dvd PLDS  DVD-RW DH16ABSH  YL32  /dev/sr0
```

- Micro-benchmarking disks (not file systems!) is also useful for understanding their characteristics
Routing Table

- Use "ip route get" to test a given IP:

```bash
$ netstat -rn
Kernel IP routing table

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Genmask</th>
<th>Flags</th>
<th>MSS</th>
<th>Window</th>
<th>irtt</th>
<th>Iface</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>192.11.0.1</td>
<td>0.0.0.0</td>
<td>UG</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>eth0</td>
</tr>
<tr>
<td>169.254.169.254</td>
<td>0.0.0.0</td>
<td>255.255.255.255</td>
<td>UH</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>eth0</td>
</tr>
<tr>
<td>192.11.0.0</td>
<td>0.0.0.0</td>
<td>255.255.240.0</td>
<td>U</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>eth0</td>
</tr>
</tbody>
</table>

$ ip route get 54.214.28.210
54.214.28.210 via 192.11.0.1 dev eth0 src 192.11.7.201
cache
etc...

- System messages: `dmesg`
- Network interface config: `ifconfig -a; ip link`
- File system capacity: `df -h`
- Volume config: `mdadm --misc -D /dev/md0`
- Storage device info: `smartctl`
- NUMA config: `numactl -s; numactl -H`
- PCI info: `lspci`
- Installed kernel modules: `lsmod`
- Root crontab config: `crontab -l`
- Services: `service --status-all`
- ...

Static Tools
Profiling
Profiling

• Objectives:
  – Profile CPU usage by stack sampling
  – Generate CPU flame graphs
  – Understand gotchas with stacks & symbols
CPU Profiling

- Record stacks at a timed interval: simple and effective
  - Pros: Low (deterministic) overhead
  - Cons: Coarse accuracy, but usually sufficient

```
stack samples:   A   B   A   B   A
                 ↓   ↓   ↓   ↓   ↓
                 B   syscall
                 ↓   ↓
                 A   on-CPU
                 ↓
                 block
                 ↓
                 interrupt
```

perf_events

- Introduced earlier: multi-tool, profiler. Provides "perf".

usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

The most commonly used perf commands are:

- annotate: Read perf.data (created by perf record) and display annotated code
- archive: Create archive with object files with build-ids found in perf.data file
- bench: General framework for benchmark suites
- buildid-cache: Manage build-id cache.
- buildid-list: List the buildids in a perf.data file
- data: Data file related processing
- diff: Read perf.data files and display the differential profile
- evlist: List the event names in a perf.data file
- inject: Filter to augment the events stream with additional information
- kmem: Tool to trace/measure kernel memory(slab) properties
- kvm: Tool to trace/measure kvm guest os
- list: List all symbolic event types
- lock: Analyze lock events
- mem: Profile memory accesses
- record: Run a command and record its profile into perf.data
- report: Read perf.data (created by perf record) and display the profile
- sched: Tool to trace/measure scheduler properties (latencies)
- script: Read perf.data (created by perf record) and display trace output
- stat: Run a command and gather performance counter statistics
- test: Runs sanity tests.
- timechart: Tool to visualize total system behavior during a workload
- top: System profiling tool.
- trace: strace inspired tool
- probe: Define new dynamic tracepoints

See 'perf help COMMAND' for more information on a specific command.
perf_events: CPU profiling

- Sampling full stack traces at 99 Hertz, for 30 secs:

```bash
# perf record -F 99 -ag -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2.745 MB perf.data (~119930 samples) ]
# perf report -n --stdio
1.40%  162 java [kernel.kallsyms]       [k] _raw_spin_lock
   |--- _raw_spin_lock
   |---63.21%-- try_to_wake_up
   |     |---63.91%-- default_wake_function
   |     |     |---56.11%-- __wake_up_common
   |     |     |---59.19%-- sock_def_readable
   |---56.11%-- __wake_up_common
   |---56.11%-- __wake_up_common
   |---56.11%-- __wake_up_common
   |---56.11%-- __wake_up_common
   |---56.11%-- __wake_up_common
[...78,000 lines truncated...]
```
perf_events: Full "report" Output
... as a Flame Graph
**perf_events: Flame Graphs**

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
cd FlameGraph
perf record -F 99 -a -g -- sleep 30
perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > perf.svg
```

- **Flame Graphs:**
  - **x-axis**: alphabetical stack sort, to maximize merging
  - **y-axis**: stack depth
  - **color**: random, or hue can be a dimension (eg, diff)

- **Interpretation:**
  - Top edge is on-CPU, beneath it is ancestry

- **Currently made from Perl + JavaScript & SVG**

- **Easy to get working**
  - [http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html](http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html)
Mysterious CPU consumer...

DEMO

&

DISCUSSION
perf_events: Workflow

Typical Workflow

- `perf list`
- `perf stat`
- `perf record`
- `perf report`
- `perf.data`
- `perf script`
- `stackcollapse-perf.pl`
- `flamegraph.pl`
perf_events: CPU Flame Graph

Broken Java stacks (missing frame pointer)

Locked

GC

Idle thread

epoll

Kernel TCP/IP

Time

Locks

Idle thread

Broken

Java stacks (missing frame pointer)
perf_events: Mixed-Mode CPU Flame Graph

Fixed Java Stacks!

Kernel (C)

JVM (C++)
perf_events: Gotchas

• Stack traces and symbols often don't work.
  – Can be a significant project to fix them. It is worth it!

• C:
  – stacks: --fno-omit-frame-pointer

• Java:
  – stacks on x86: -XX:+PreserveFramePointer
    (JDK-8068945 for JDK9, JDK-8072465 for JDK8)
  – symbols: perf-map-agent (other solutions exist)

• Node.js:
  – symbols: run v8 with --perf_basic_prof

perf_events: Counters

- Performance Monitoring Counters (PMCs):
  
  $ 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)
  - Please develop front-ends. Eg, tiptop.
Tracing
Tracing

- Objectives:
  - Understand frameworks: tracepoints, kprobes, uprobes
  - Understand mainline tracers: ftrace, perf_events, eBPF
  - Awareness of other tracers: SystemTap, LTTng, ktap, sysdig
  - Awareness of what tracing can accomplish (eg, perf-tools)
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
Linux Tracing Tools

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

- Many choices (too many?), all still in development
Linux Tracing is Magic!

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

• (Thanks Deirdré Straughan & General Zoi's Pony Creator)
Choosing a Tracer

• Some companies standardize on one tracer
  – eg, SystemTap, LTTng, …
Choosing a Tracer

- My approach is:

  Study what Linux already has built-in (perf_events, ftrace, eBPF?)

  - Purpose?
    - live tracing, counting
    - ftrace
    - perf_events
    - eBPF
    - PMC, stack profiling, trace-dump-analyze

  Is it sufficient?

    Y
    - Try SystemTap

    N
    - Try LTTng
    - ...

...
ftrace
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:

```bash
linux-4.0.0+ # ls /sys/kernel/debug/tracing/
available_events   max_graph_depth   stack_max_size
available_filter_functions  options       stack_trace
available_tracers      per_cpu         stack_trace_filter
buffer_size_kb        printk_formats   trace
buffer_total_size_kb   README          trace_clock
current_tracer         saved_cmdlines   trace_marker
dyn_ftrace_total_info  saved_cmdlines_size trace_options
enabled_functions      set_event        trace_pipe
events                 set_ftrace_filter  trace_stat
free_buffer            set_ftrace_notrace tracing_cpumask
function_profile_enabled set_ftrace_pid   tracing_max_latency
instances              set_graph_function tracing_on
kprobe_events          set_graph_notrace tracing_threash
kprobe_profile         snapshot
```

- See Linux source: Documentation/trace/ftrace.txt
ftrace Front-Ends

• Steven wrote a front-end: trace-cmd
  – Multi-tool, works well

• I've developed the "perf-tools" front-ends
  – https://github.com/brendangregg/perf-tools
  – Both single & multi-purpose, Unix-like
  – Unsupported hacks: see WARNINGS

• perf-tools:
  – single-purpose: iosnoop, iolatency, opensnoop
  – multi-tools: funccount, funcgraph, kprobe
iosnoop

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

```
# ./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

[...]
```

```
# ./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

- Block I/O (disk) latency distributions:

```bash
# ./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>
```

[...]
opensnoop

• Trace open() syscalls showing filenames:

```bash
# ./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.node.map</td>
</tr>
<tr>
<td>4345768.333971</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>global/pg_internal.init</td>
</tr>
<tr>
<td>4345768.334813</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/PG_VERSION</td>
</tr>
<tr>
<td>4345768.334877</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/pg_fileno.node.map</td>
</tr>
<tr>
<td>4345768.334891</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/pg_internal.init</td>
</tr>
<tr>
<td>4345768.335821</td>
<td>postgres</td>
<td>23886</td>
<td>0x5</td>
<td>base/16384/11725</td>
</tr>
<tr>
<td>4345768.334791</td>
<td>svstat</td>
<td>24649</td>
<td>0x4</td>
<td>supervise/ok</td>
</tr>
<tr>
<td>4345768.334792</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>
[...]
```
tpoint

• Who is creating disk I/O, and of what type?

```
# .%/tpoint -H block:block_rq_insert
Tracing block:block_rq_insert. Ctrl-C to end.
# tracer: nop
#
#     TASK-PID    CPU#    TIMESTAMP  FUNCTION
#        | |       |          |         |
flush-9:0-9318  [013] 1936182.007939: block_rq_insert: 202,16 W 0 () 280100936 + 8 [flush-9:0]
java-9469  [014] 1936182.316184: block_rq_insert: 202,1 R 0 () 1319592 + 72 [java]
java-9469  [000] 1936182.331270: block_rq_insert: 202,1 R 0 () 1125744 + 8 [java]
java-9469  [000] 1936182.341418: block_rq_insert: 202,1 R 0 () 2699008 + 88 [java]
java-9469  [000] 1936182.341419: block_rq_insert: 202,1 R 0 () 2699096 + 88 [java]
java-9469  [000] 1936182.345870: block_rq_insert: 202,1 R 0 () 1320304 + 24 [java]
java-9469  [000] 1936182.351590: block_rq_insert: 202,1 R 0 () 1716848 + 16 [java]
^C
Ending tracing...
```

• tpoint traces a given tracepoint. -H prints the header.
tpoint -l

Listing tracepoints

- 1,257 tracepoints for this Linux kernel
funccount

• Count a kernel function call rate:

```bash
# ./funccount -i 1 'bio_*'
Tracing "bio_*"... Ctrl-C to end.

<table>
<thead>
<tr>
<th>FUNC</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bio_attempt_back_merge</td>
<td>26</td>
</tr>
<tr>
<td>bio_get_nr_vecs</td>
<td>361</td>
</tr>
<tr>
<td>bio_alloc</td>
<td>536</td>
</tr>
<tr>
<td>bio_alloc_bioset</td>
<td>536</td>
</tr>
<tr>
<td>bio_endio</td>
<td>536</td>
</tr>
<tr>
<td>bio_free</td>
<td>536</td>
</tr>
<tr>
<td>bio_fs_destructor</td>
<td>536</td>
</tr>
<tr>
<td>bio_init</td>
<td>536</td>
</tr>
<tr>
<td>bio_integrity_enabled</td>
<td>536</td>
</tr>
<tr>
<td>bio_put</td>
<td>729</td>
</tr>
<tr>
<td>bio_add_page</td>
<td>1004</td>
</tr>
</tbody>
</table>

Counts are in-kernel, for low overhead

[...]

– -i: set an output interval (seconds), otherwise until Ctrl-C
• 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

<table>
<thead>
<tr>
<th>TIME</th>
<th>CPU</th>
<th>DURATION</th>
<th>FUNCTION CALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4346366.073832</td>
<td>0)</td>
<td></td>
<td>vfs_read() {</td>
</tr>
<tr>
<td>4346366.073834</td>
<td>0)</td>
<td></td>
<td>rw_verify_area() {</td>
</tr>
<tr>
<td>4346366.073834</td>
<td>0)</td>
<td></td>
<td>security_file_permission() {</td>
</tr>
<tr>
<td>4346366.073834</td>
<td>0)</td>
<td></td>
<td>apparmor_file_permission() {</td>
</tr>
<tr>
<td>4346366.073835</td>
<td>0)</td>
<td>0.153 us</td>
<td>common_file_perm();</td>
</tr>
<tr>
<td>4346366.073836</td>
<td>0)</td>
<td>0.947 us</td>
<td>__fsnotify_parent();</td>
</tr>
<tr>
<td>4346366.073836</td>
<td>0)</td>
<td>0.066 us</td>
<td>fsnotify();</td>
</tr>
<tr>
<td>4346366.073836</td>
<td>0)</td>
<td>0.080 us</td>
<td>}</td>
</tr>
<tr>
<td>4346366.073837</td>
<td>0)</td>
<td>2.174 us</td>
<td>tty_read() {</td>
</tr>
<tr>
<td>4346366.073837</td>
<td>0)</td>
<td>2.656 us</td>
<td>tty_paranoia_check();</td>
</tr>
<tr>
<td>4346366.073837</td>
<td>0)</td>
<td>0.060 us</td>
<td>}</td>
</tr>
</tbody>
</table>
[...]```
kprobe

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

```
# ./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?
perf-tools (so far...)

Operating System

System Libraries

Applications

System Call Interface

CPU Interconnect

Hardware

Various:

tpoint

CPU

1

Memory Bus

DRAM

I/O Bridge

I/O Controller

Disk

Disk

Swap

Interface Transports

Network Controller

Port

Port

Function Graph

funcgraph

writes

read

funcslow

functime

funcdtime

funcdtime

hw-affinity

Operating System

System Libraries

CPU Interconnect

Hardware

Various:

tpoint

CPU

1

Memory Bus

DRAM

I/O Bridge

I/O Controller

Disk

Disk

Swap

Interface Transports

Network Controller

Port

Port

Function Graph

funcgraph

writes

read

funcslow

functime

funcdtime

funcdtime

hw-affinity
perf-tools (so far...)

Various:
- tpoint
- tpoint ext4::*
- tpoint vmscan::*
- tpoint scsi::*
- tpoint irq::*
- tpoint net::*

Hardware
- execsnoop
- killsnoop

CPU Interconnect
- CPU 1
- Memory Bus
- DRAM

Device Drivers
- Device
- Expander Interconnect
- I/O Bridge
- Interface Transports
- I/O Controller
- Network Controller
- Port
- Port

System Libraries
- VFS
- Sockets
- Scheduler
- IP
- TCP/UDP
- Virtual Memory

System Call Interface
- System Call Interface

Applications
- Operating System

System Call Interface (so far...)

Open sources
- opensnoop
- syscall
- tpoint syscalls::*
- tpoint ext4::*
- funccount
- functrace
- funcslower
- funcgraph
- kprobe

I/O Controller
- Disk
- Disk
- Swap

Memory Bus
- Memory
- DRAM
perf_events
perf_events

• Powerful profiler (covered earlier)

• … and tracer:
  – User-level and kernel dynamic tracing
  – Kernel line tracing and local variables (debuginfo)
  – Kernel filtering expressions
  – Efficient in-kernel counts (perf stat)

• Not very programmable, yet
  – Limited kernel summaries. Should improve with eBPF.
perf_events Listing Tracepoints

# perf list 'block:*'
  skb:kfree_skb
  skb:consume_skb
  skb:skb_copy_datagram_iovec
  net:net_dev_xmit
  net:net_dev_queue
  net:netif_receive_skb
  net:netif_rx
[...]
  block:block_touch_buffer
  block:block_dirty_buffer
  block:block_rq_abort
  block:block_rq_requeue
  block:block_rq_complete
  block:block_rq_insert
  block:block_rq_issue
  block:block_bio_bounce
  block:block_bio_complete
  block:block_bio_backmerge
[...]
Linux Event Sources

Dynamic Tracing

uprobes

kprobes

Tracepoints

Operating System

Applications

System Libraries

System Call Interface

VFS

Sockets

TCP/UDP

File Systems

IP

Volume Manager

Scheduler

Block Device Interface

Ethernet

Device Drivers

Software Events

cpu-clock

cs migrations

page-faults

minor-faults

major-faults

syscalls:

sched:

task:

signal:

timer:

workqueue:

sock:

kmem:

vmscan:

writeback:

irq:

cycles

instructions

branch-*

L1-*

LLC-*

PMCs

CPU

Memory Bus

DRAM

mem-load

mem-store

jbd2:

block:

scsi:

net:

skb:

http://www.brendangregg.com/perf.html 2015
perf_events Tracing Tracepoints

- If `-g` is used in "perf record", stack traces are included.
- If "perf script" output is too verbose, try "perf report", or making a flame graph.
perf_events Report

```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 -n --stdio
[...]
  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

[...]```
eBPF
eBPF

- Extended BPF: programs on tracepoints
  - High performance filtering: JIT
  - In-kernel summaries: maps
  - Developed by Alexei Starovoitov (Plumgrid)
- Currently being integrated in parts (Linux 3.18, 4.1, …)

```
# ./bitehist
Tracing block device I/O... Interval 5 secs. Ctrl-C to end.

<table>
<thead>
<tr>
<th>kbytes</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>3</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>0</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>3395</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>1</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>2</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>738</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>3</td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>1</td>
</tr>
</tbody>
</table>

distribution

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>********************</td>
</tr>
<tr>
<td>********************</td>
</tr>
<tr>
<td>********************</td>
</tr>
<tr>
<td>********************</td>
</tr>
<tr>
<td>********************</td>
</tr>
<tr>
<td>********************</td>
</tr>
<tr>
<td>********************</td>
</tr>
</tbody>
</table>

in-kernel summary
```
eBPF

- Example in-kernel latency heat map:
Other Tracers
SystemTap

- Fully programmable, fully featured
  - Including access to user-level tracepoints
- 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 (>=2.6)
- "Works great with kernel debuginfo"
  - Suited for kernel developers who have it handy
  - A difficult requirement for our cloud environment
"Lightweight" SystemTap

• SystemTap can be used without kernel debuginfo
  – Makes life harder, but some tasks are still possible
  – **providers**: nd_syscall, kprobe.function, kernel.trace, …
  – **args** via: int_arg(), uint_arg(), pointer_arg(), user_string()

• Something I've experimented with. Examples:
  – https://github.com/brendangregg/systemtap-lwtools/

```bash
# stap -e 'global a; probe nd_syscall.write { a <<< int_arg(3); } probe end
{ print(@hist_log(a)); }'
```

<table>
<thead>
<tr>
<th>value</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>32</td>
<td>15</td>
</tr>
<tr>
<td>64</td>
<td>17</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>512</td>
<td>0</td>
</tr>
</tbody>
</table>
ktap

• Was a very promising new Linux tracer:
  – Sampling, static & dynamic tracing
  – Suited for embedded devices
• Development suspended while eBPF integrates
• Will it restart?
sysdig

- sysdig: Innovative new tracer. Simple expressions:

```bash
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:

```bash
# sysdig -c fileslower 1
```

```
<table>
<thead>
<tr>
<th>TIME</th>
<th>PROCESS</th>
<th>TYPE</th>
<th>LAT(ms)</th>
<th>FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-04-13 20:40:43.973</td>
<td>cksum</td>
<td>read</td>
<td>2</td>
<td>/mnt/partial.0.0</td>
</tr>
<tr>
<td>2014-04-13 20:40:44.187</td>
<td>cksum</td>
<td>read</td>
<td>1</td>
<td>/mnt/partial.0.0</td>
</tr>
<tr>
<td>2014-04-13 20:40:44.689</td>
<td>cksum</td>
<td>read</td>
<td>2</td>
<td>/mnt/partial.0.0</td>
</tr>
</tbody>
</table>

[...]
```

- Currently syscalls and user-level processing only
  - I'm not sure it can be optimized enough for kernel tracing, unless it adopts eBPF for in-kernel processing & summaries
Present & Future

• Present:
  – ftrace & perf_events solving many needs today:
    • PMC profiling, stack profiling, tracepoint & dynamic tracing, …

• Expected Future:
  – eBPF for kernel summaries & advanced programs
  – eBPF perf integration to improve ease of use

• Possible Future:
  – eBPF high level language (ktap?)
  – ftrace/eBPF integration
  – Other tracer eBPF integration (SystemTap, LTTng, sysdig?)
  – One of the other tracers going mainline?
The Tracing Landscape, May 2015

Ease of use

Stage of Development

Scope & Capability

- sysdig
- perf
- ftrace
- dtrace4L.
- ktap
- stap
- eBPF

(my opinion)

(alpha) → (mature)

(brutal) → (less brutal)
In Summary…
Methodologies Summary

• Objectives:
  – Recognize the Streetlight Anti-Method
  – Perform the Workload Characterization Method
  – Perform the USE Method
  – Be aware of other methodologies

Try to start with the questions (methodology), to help guide your use of the tools
Tools Summary

- **Objectives:**
  - Perform the USE Method for resource utilization
  - Perform Workload Characterization for disks, network
  - Have exposure to various observability tools:
    - Basic: vmstat, iostat, mpstat, ps, top, …
    - Intermediate: tcpdump, netstat, nicstat, pidstat, sar, …
    - Advanced: ss, slaptop, perf_events, …
  - Perform Active Benchmarking
  - Understand tuning risks
  - Perform Static Performance Tuning

Print out the tools diagrams for your office wall
Profiling & Tracing Summary

• Objectives:
  – Profile CPU usage by stack sampling
  – Generate CPU flame graphs
  – Understand gotchas with stacks & symbols
  – Understand frameworks: tracepoints, kprobes, uprobes
  – Understand mainline tracers: ftrace, perf_events, eBPF
  – Awareness of other tracers: SystemTap, LTTng, ktap, sysdig
  – Awareness of what tracing can accomplish (eg, perf-tools)

I've hopefully turned some unknown unknowns into known unknowns
References & Links

– Systems Performance: Enterprise and the Cloud, Prentice Hall, 2013
– http://www.brendangregg.com/linuxperf.html incl. tools diagrams as PNGs
– 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/ Ftrace: The hidden light switch
– 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