Netflix
Instance Performance Analysis
Requirements

Brendan Gregg
Senior Performance Architect
Performance Engineering Team
bgregg@netflix.com @brendangregg
Monitoring companies are selling faster horses

I want to buy a car
Server/Instance Analysis Potential

In the last 10 years...

- More Linux
- More Linux metrics
- Better visualizations
- Containers

Conditions ripe for innovation: where is our Henry Ford?
This Talk

• **Instance analysis**: system resources, kernel, processes
  – For customers: what you can ask for
  – For vendors: our desirables & requirements
  – What we are building (and open sourcing) at Netflix to modernize instance performance analysis (Vector, …)
• Over 60M subscribers
• FreeBSD CDN for content delivery
• Massive AWS EC2 Linux cloud
• Many monitoring/analysis tools
• Awesome place to work
Agenda

1. Desirables
2. Undesirables
3. Requirements
4. Methodologies
5. Our Tools
1. Desirables
Line Graphs

CPU Utilization

- sys
- user
Historical Data
Summary Statistics
Histograms

... or a density plot
Heat Maps

Latency Heat Map

time 54s, range 840-880us, count: 34
Disk: I/O operations per second broken down by latency
Frequency Trails
## Waterfall Charts

<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Type</th>
<th>Size</th>
<th>Time</th>
<th>Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobs.netflix.com</td>
<td>CET</td>
<td>text/html</td>
<td>2.8 KB</td>
<td>101 ms</td>
<td></td>
</tr>
<tr>
<td>fonts.css</td>
<td>CET</td>
<td>text/html</td>
<td>428 B</td>
<td>228 ms</td>
<td></td>
</tr>
<tr>
<td>modernizr-daa757b7ded78...</td>
<td>CET</td>
<td>application</td>
<td>180 B</td>
<td>71 ms</td>
<td></td>
</tr>
<tr>
<td>application-2347b199ff55...</td>
<td>CET</td>
<td>application</td>
<td>182 B</td>
<td>133 ms</td>
<td></td>
</tr>
<tr>
<td>application-574fde8512795...</td>
<td>CET</td>
<td>text/css</td>
<td>181 B</td>
<td>122 ms</td>
<td></td>
</tr>
<tr>
<td>6196183C85D2A944.css</td>
<td>CET</td>
<td>text/css</td>
<td>181 B</td>
<td>54 ms</td>
<td></td>
</tr>
<tr>
<td>ga.js</td>
<td>CET</td>
<td>text/javascript</td>
<td>35 B</td>
<td>44 ms</td>
<td></td>
</tr>
<tr>
<td>mixpanel-2.2.min.js</td>
<td>CET</td>
<td>application</td>
<td>253 B</td>
<td>44 ms</td>
<td></td>
</tr>
<tr>
<td>in.js?async=true&amp;_t=1433626</td>
<td>CET</td>
<td>text/javascript</td>
<td>2.1 KB</td>
<td>50 ms</td>
<td></td>
</tr>
<tr>
<td>social-663f3d36b9048c89...</td>
<td>CET</td>
<td>application</td>
<td>43.9 KB</td>
<td>339 ms</td>
<td></td>
</tr>
<tr>
<td>data/application/x-...</td>
<td>CET</td>
<td>application</td>
<td>0 B</td>
<td>11 ms</td>
<td></td>
</tr>
<tr>
<td>data/application/x-...</td>
<td>CET</td>
<td>application</td>
<td>0 B</td>
<td>14 ms</td>
<td></td>
</tr>
<tr>
<td>data/application/x-...</td>
<td>CET</td>
<td>application</td>
<td>0 B</td>
<td>16 ms</td>
<td></td>
</tr>
<tr>
<td>netflix-logo-13f5a3e6f08502...</td>
<td>CET</td>
<td>image/png</td>
<td>1.5 KB</td>
<td>192 ms</td>
<td></td>
</tr>
<tr>
<td>slide_4-9f2f38f6c46b0fc7f3...</td>
<td>CET</td>
<td>image/jpeg</td>
<td>181 B</td>
<td>203 ms</td>
<td></td>
</tr>
<tr>
<td>slide_3-fb8c724c480343da...</td>
<td>CET</td>
<td>image/jpeg</td>
<td>175 B</td>
<td>600 ms</td>
<td></td>
</tr>
<tr>
<td>slide_2-18c972f59a98e97...</td>
<td>CET</td>
<td>image/jpeg</td>
<td>138 B</td>
<td>562 ms</td>
<td></td>
</tr>
<tr>
<td>slide_1-59f183cb0b603dbf...</td>
<td>CET</td>
<td>image/jpeg</td>
<td>79.9 KB</td>
<td>474 ms</td>
<td></td>
</tr>
<tr>
<td>_utm.gif?utmww=5.6.4&amp;utms...</td>
<td>CET</td>
<td>image/gif</td>
<td>70 B</td>
<td>144 ms</td>
<td></td>
</tr>
<tr>
<td>_utm.gif?utmww=5.6.4&amp;utms...</td>
<td>CET</td>
<td>image/gif</td>
<td>65 B</td>
<td>144 ms</td>
<td></td>
</tr>
<tr>
<td>?data=eyJidR...mixpanel-2...</td>
<td>CET</td>
<td>application</td>
<td>415 B</td>
<td>129 ms</td>
<td></td>
</tr>
</tbody>
</table>

33 requests | 533 KB transferred | Finish: 1.52 s | DOMContentLoaded: 823 ms | Load: 1.43 s
Directed Graphs
Flame Graphs
# Flame Charts

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>1800</th>
<th>1900</th>
<th>2000</th>
<th>2100</th>
<th>2200</th>
<th>2300</th>
<th>2400</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>949.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>66.9</td>
<td></td>
</tr>
</tbody>
</table>

**Main Thread**

- **Evaluate Script** (https://js...ff55ce4516f0d8684102c.js:1)
- **Parse HTML**
- **Event (DOMContentLoaded)**
- **Function Ca...102c.js:1**

---

### Diagram Details:

- **Evaluate Script**
- **Parse HTML**
- **DOMContentLoaded**
- **Function Call**

---

**Timeline Details:**

- Time durations labeled for each section.
- Event stages and function calls highlighted in the diagram.
… Without Running All These
Other Desirables

- Safe for production use
- Easy to use: self service
- [Near] Real Time
- Ad hoc / custom instrumentation
- Complete documentation
- Graph labels and units
- Open source
- Community
2. Undesirables
Tachometers

...especially with arbitrary color highlighting
Pie Charts

...for real-time metrics
Doughnuts

...like pie charts but worse
Traffic Lights

RED == BAD (usually)

GREEN == GOOD (hopefully)

...when used for subjective metrics

These can be used for objective metrics

For subjective metrics (e.g., IOPS/latency) try weather icons instead
3. Requirements
Acceptable T&Cs

• Probably acceptable:

XXX, Inc. shall have a royalty-free, worldwide, transferable, and perpetual license to use or incorporate into the Service any suggestions, ideas, enhancement requests, feedback, or other information provided by you or any Authorized User relating to the Service.

• Probably not acceptable:

By submitting any Ideas, Customer and Authorized Users agree that: ... (iii) all right, title and interest in and to the Ideas, including all associated IP Rights, shall be, and hereby are, assigned to [us]

• Check with your legal team
Acceptable Technical Debt

• It must be worth the …
  • Extra complexity when debugging
  • Time to explain to others
  • Production reliability risk
  • Security risk
• There is no such thing as a free trial
Known Overhead

• Overhead must be known to be managed
  – T&Cs should not prohibit its measurement or publication

• Sources of overhead:
  – CPU cycles
  – File system I/O
  – Network I/O
  – Installed software size

• We will measure it
Low Overhead

• Overhead should also be the lowest possible
  – 1% CPU overhead means 1% more instances, and $$$

• Things we try to avoid
  – Tracing every function/method call
  – Needless kernel/user data transfers
  – strace (ptrace), tcpdump, libpcap, …

• Event logging doesn't scale
Scalable

• Can the product scale to (say) 100,000 instances?
  – Atlas, our cloud-wide analysis tool, can
  – We tend to kill other monitoring tools that attempt this

• Real-time dashboards showing all instances:
  – How does that work? Can it scale to 1k? … 100k?
  – Adrian Cockcroft's spigo can simulate protocols at scale

• High overhead might be worth it: on-demand only
An instance analysis solution must provide actionable information that helps us improve performance.
4. Methodologies
Methodologies pose the questions for metrics to answer.

Good monitoring/analysis tools should support performance analysis methodologies.
Drunk Man *Anti*-Method

• Tune things at random until the problem goes away
Workload Characterization

Study the workload applied:

1. Who
2. Why
3. What
4. How
Workload Characterization

Eg, for CPUs:

1. **Who**: which PIDs, programs, users
2. **Why**: code paths, context
3. **What**: CPU instructions, cycles
4. **How**: changing over time
CPUs

Who

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>VIRT</th>
<th>RES</th>
<th>CPU%</th>
<th>MEM%</th>
<th>TIME+</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>2763</td>
<td>root</td>
<td>72384</td>
<td>2949</td>
<td>1.1%</td>
<td>0.1%</td>
<td>0.00</td>
<td>ab -k -c 100 -n 1</td>
</tr>
<tr>
<td>28004</td>
<td>root</td>
<td>22229</td>
<td>2049</td>
<td>14.4%</td>
<td>2.7</td>
<td>2:03:60</td>
<td>/usr/lib/jvm/java</td>
</tr>
<tr>
<td>28173</td>
<td>root</td>
<td>63488</td>
<td>4992</td>
<td>95.0%</td>
<td>0.1%</td>
<td>0:02:68</td>
<td>htop</td>
</tr>
<tr>
<td>28170</td>
<td>root</td>
<td>24660</td>
<td>2176</td>
<td>3.0%</td>
<td>0.0%</td>
<td>0:00:62</td>
<td>htop</td>
</tr>
<tr>
<td>2730</td>
<td>bgrep-gte</td>
<td>2629</td>
<td>58668</td>
<td>0.0%</td>
<td>0.8</td>
<td>2:31:25</td>
<td>/apps/epic/perl/b</td>
</tr>
<tr>
<td>2752</td>
<td>bgrep-gte</td>
<td>1531</td>
<td>10386</td>
<td>0.0%</td>
<td>0.1</td>
<td>1:40:36</td>
<td>postgres; bgrep-</td>
</tr>
<tr>
<td>29800</td>
<td>root</td>
<td>23336</td>
<td>2049</td>
<td>2.7</td>
<td>0:00:26</td>
<td>/usr/lib/jvm/java</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>24320</td>
<td>2256</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:01:29</td>
<td>/sbin/init</td>
</tr>
<tr>
<td>341</td>
<td>root</td>
<td>17236</td>
<td>632</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:04</td>
<td>upstart-udev-bridge</td>
</tr>
<tr>
<td>346</td>
<td>root</td>
<td>21600</td>
<td>1304</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:00</td>
<td>/sbin/udevd -- daemon</td>
</tr>
<tr>
<td>357</td>
<td>root</td>
<td>25944</td>
<td>1164</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:21</td>
<td>dbus-daemon -- syslog</td>
</tr>
<tr>
<td>408</td>
<td>root</td>
<td>21464</td>
<td>792</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:00</td>
<td>/sbin/udevd -- daemon</td>
</tr>
<tr>
<td>549</td>
<td>root</td>
<td>15192</td>
<td>392</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:00</td>
<td>upstart-socket-br</td>
</tr>
<tr>
<td>612</td>
<td>root</td>
<td>7268</td>
<td>1828</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:24</td>
<td>dhclient3 -e IF_M</td>
</tr>
<tr>
<td>644</td>
<td>root</td>
<td>50630</td>
<td>2928</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:00</td>
<td>/usr/sbin/sshd -d</td>
</tr>
<tr>
<td>772</td>
<td>root</td>
<td>14508</td>
<td>956</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:00</td>
<td>/sbin/getty -8 38</td>
</tr>
<tr>
<td>777</td>
<td>root</td>
<td>14508</td>
<td>952</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:00</td>
<td>/sbin/getty -8 38</td>
</tr>
<tr>
<td>785</td>
<td>root</td>
<td>14508</td>
<td>952</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0:00:00</td>
<td>/sbin/getty -8 38</td>
</tr>
</tbody>
</table>

Why

![Flame Graph]

How

![CPU Utilization Graph]

What

```
root@lguv-gregg:~: perf stat -a -d sleep 10

Performance counter stats for 'system wide':
39996.38868 task-clock (msec) # 3.999 CPUs ut
1,026,540 context-switches # 0.826 M/sec
193,563 cpu-migrations # 0.085 M/sec
4,835 page-faults # 0.012 K/sec
83,859,543,001 cycles # 2.897 GHz
61,028,919,136 stalled-cycles-frontend # 72.78% frontend
50,812,852,642 stalled-cycles-backend # 68.59% backend
52,969,864,055 instructions # 0.63 insns per
10,223,584,755 branches # 255.613 M/sec
376,529,869 branch-misses # 3.68% of all
0 L1-cache-loads # 0.000 K/sec
1,339,958,792 L1-cache-load-misses # 0.000% of all
762,761,193 LLC-loads # 19.071 M/sec
```
CPUs

Who

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>VIRT</th>
<th>RES</th>
<th>CPU%</th>
<th>MEM%</th>
<th>TIME+</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>2783</td>
<td>root</td>
<td>2759</td>
<td>2849</td>
<td>117.2</td>
<td>2.7</td>
<td>2580.5</td>
<td>/usr/lib/jvm/java</td>
</tr>
<tr>
<td>28084</td>
<td>root</td>
<td>2242</td>
<td>2849</td>
<td>144.4</td>
<td>2.7</td>
<td>2580.5</td>
<td>/usr/lib/jvm/java</td>
</tr>
<tr>
<td>28173</td>
<td>root</td>
<td>63488</td>
<td>4992</td>
<td>95.0</td>
<td>0.1</td>
<td>0:02.68</td>
<td>ab -k -c 100 -n 1</td>
</tr>
<tr>
<td>28170</td>
<td>root</td>
<td>24660</td>
<td>2176</td>
<td>3.0</td>
<td>0.0</td>
<td>0:00.62</td>
<td>htop</td>
</tr>
<tr>
<td>2730</td>
<td>bgregg-te</td>
<td>5865</td>
<td>0.0</td>
<td>0.0</td>
<td>2.0</td>
<td>31.25</td>
<td>/apps/epic/perf/b</td>
</tr>
<tr>
<td>2732</td>
<td>bgregg-te</td>
<td>3371</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2.36</td>
<td>postgres; bgregg-</td>
</tr>
<tr>
<td>29800</td>
<td>root</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>spin/init</td>
</tr>
<tr>
<td>29800</td>
<td>root</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>spin/udev-bridge</td>
</tr>
<tr>
<td>314</td>
<td>root</td>
<td>21464</td>
<td>792</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0000</td>
<td>/sbin/udevd --da</td>
</tr>
<tr>
<td>549</td>
<td>root</td>
<td>19192</td>
<td>392</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0000</td>
<td>/sbin/udevd --da</td>
</tr>
<tr>
<td>612</td>
<td>root</td>
<td>7268</td>
<td>1828</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2460</td>
<td>dchlient3-e-IF_M</td>
</tr>
<tr>
<td>644</td>
<td>root</td>
<td>50830</td>
<td>2920</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0000</td>
<td>/usr/sbin/sshd -c</td>
</tr>
<tr>
<td>772</td>
<td>root</td>
<td>14508</td>
<td>956</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0000</td>
<td>/sbin/getty -8 38</td>
</tr>
<tr>
<td>777</td>
<td>root</td>
<td>14508</td>
<td>952</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0000</td>
<td>/sbin/getty -8 38</td>
</tr>
<tr>
<td>785</td>
<td>root</td>
<td>14508</td>
<td>957</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0000</td>
<td>/sbin/getty -8 38</td>
</tr>
</tbody>
</table>

Why

- perf record -g
- flame graphs

How

- top, htop
- monitoring

What

- perf stat -a -d
- sleep 10
Most Monitoring Products Today

**Who**

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>VIRT</th>
<th>RES</th>
<th>CPU%</th>
<th>MEM%</th>
<th>TIME+</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>21563</td>
<td>root</td>
<td>3253</td>
<td>2048</td>
<td>117</td>
<td>2.7</td>
<td>21:31:36</td>
<td>/usr/lib/jvm/jre/bin/java</td>
</tr>
<tr>
<td>28804</td>
<td>root</td>
<td>2222</td>
<td>2048</td>
<td>104</td>
<td>2.7</td>
<td>2:03:60</td>
<td>/usr/lib/jvm/jre/bin/java</td>
</tr>
<tr>
<td>28173</td>
<td>root</td>
<td>63488</td>
<td>4992</td>
<td>95.8</td>
<td>0.1</td>
<td>0:02:68</td>
<td>ab -k c 100 -n 1</td>
</tr>
<tr>
<td>28170</td>
<td>root</td>
<td>24660</td>
<td>2176</td>
<td>3.0</td>
<td>0.0</td>
<td>0:00:62</td>
<td>htop</td>
</tr>
<tr>
<td>2730</td>
<td>bgregg</td>
<td>202M</td>
<td>58668</td>
<td>0.0</td>
<td>0.8</td>
<td>2h31:25</td>
<td>/apps/epic/perl/bin/bregg</td>
</tr>
<tr>
<td>2752</td>
<td>15M</td>
<td>18208</td>
<td>0.1</td>
<td>14:46:27</td>
<td>postgres; bregg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>298000</td>
<td>root</td>
<td>3</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/bin/init</td>
</tr>
<tr>
<td>347</td>
<td>root</td>
<td>1</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/start-udev-bridge</td>
</tr>
<tr>
<td>346</td>
<td>root</td>
<td>1</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/bin/udevd --daemon</td>
</tr>
<tr>
<td>357</td>
<td>root</td>
<td>1</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/bin/udevd --daemon</td>
</tr>
<tr>
<td>480</td>
<td>root</td>
<td>21464</td>
<td>792</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/sbin/dasd-daemon --system</td>
</tr>
<tr>
<td>549</td>
<td>root</td>
<td>15192</td>
<td>392</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/sbin/udevd --daemon</td>
</tr>
<tr>
<td>612</td>
<td>root</td>
<td>7268</td>
<td>1828</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/sbin/udevd --daemon</td>
</tr>
<tr>
<td>644</td>
<td>root</td>
<td>50038</td>
<td>3292</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/sbin/shd --daemon</td>
</tr>
<tr>
<td>772</td>
<td>root</td>
<td>14508</td>
<td>956</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/sbin/getty --daemon</td>
</tr>
<tr>
<td>777</td>
<td>root</td>
<td>14508</td>
<td>952</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/sbin/getty --daemon</td>
</tr>
<tr>
<td>785</td>
<td>root</td>
<td>14508</td>
<td>952</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>/sbin/getty --daemon</td>
</tr>
</tbody>
</table>

**Why**

```
perf record -g flame Graphs
```

**How**

```
perf stat -a -d
```

**What**

```
monitoring
```
The USE Method

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

- Saturation is queue length or queued time
- Start by drawing a functional (block) diagram of your system / software / environment
USE Method for Hardware

Include busses & interconnects!
# 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;idl&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 popu; 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 schedulenames.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;pgscank&quot; + &quot;pgscand&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 capacity</td>
<td>errors</td>
<td>dmesg for physical failures; dynamic tracing, eg, SystemTap uprobes for failed malloc()s</td>
</tr>
<tr>
<td>Network Interfaces</td>
<td>utilization</td>
<td>sar -n DEV 1,&quot;txKB/s&quot;/&quot;rxKB/s&quot;/max &quot;txKB/s&quot;/max &quot;txKB/s&quot;/max; ip -s link, RX/TX tput / max bandwidth; /proc/net/dev, &quot;bytes&quot; RX/TX tput/max; nicstat &quot;%Util&quot; [6]</td>
</tr>
<tr>
<td>Network Interfaces</td>
<td>saturation</td>
<td>ifconfig, &quot;overruns&quot;, &quot;dropped&quot;; netstat -s, &quot;segments transmitted&quot;; sar -n EDEV, *drop and *fifo metrics; /proc/net/dev, RX/TX &quot;drop&quot;; nicstat &quot;Sat&quot; [6]; dynamic tracing for other TCP/IP stack queueing [7]</td>
</tr>
<tr>
<td>Network Interfaces</td>
<td>errors</td>
<td>ifconfig, &quot;errors&quot;, &quot;dropped&quot;; netstat -i, &quot;RX-ERR&quot;/&quot;TX-ERR&quot;; ip -s link, &quot;errors&quot;; sar -n EDEV, &quot;rxerr/s&quot;/&quot;txerr/s&quot;; /proc/net/dev, &quot;errs&quot;, &quot;drop&quot;; extra counters may be under /sys/class/net/...; dynamic tracing of driver function returns [8]</td>
</tr>
</tbody>
</table>
Most Monitoring Products Today

- Showing what **is** and **is not** commonly measured
- Score: 8 out of 33 (24%)
- We can do better…
Other Methodologies

• There are many more:
  – Drill-Down Analysis Method
  – Time Division Method
  – Stack Profile Method
  – Off-CPU Analysis
  – …
  – I've covered these in previous talks & books
5. Our Tools

Atlas

Vector

Netflix Open Source Software
BaseAMI

- Many sources for instance metrics & analysis
  - Atlas, Vector, sar, perf-tools (ftrace, perf_events), …
- Currently not using 3rd party monitoring vendor tools

Linux (usually Ubuntu)

- Optional Apache, memcached, Node.js, ...
- Atlas, S3 log rotation, sar, ftrace, perf, stap, perf-tools
- Vector, pcp

Java (JDK 7 or 8)

- GC and thread dump logging
- Tomcat
  - Application war files, platform, base servelet
  - hystrix, metrics (Servo), health check
Netflix Atlas

Graphs and metrics for Netflix Atlas, showing data for nf.app / nf.cluster / nf.asg with Expressions such as apiproxy and nf.app:apiproxy. Metrics include CPU usage with names like CpuRawIdle, CpuRawInterrupt, CpuRawKernel, CpuRawNice, CpuRawSystem, and CpuRawUser. The graph displays metrics over time for specific nodes and statistics like avg, min, max, total, current. The region is set to us-east-1.
Netflix Vector

Select Instance

Select Metrics

Flame Graphs

Near real-time, per-second metrics
Java CPU Flame Graphs

Brendan’s patched OpenJDK, Mixed Mode CPU Flame Graph: vert.x

Function: io.netty.channel.nio.NioEventLoop::run (2,885 samples, 94.87%)
Java CPU Flame Graphs

Brendan's patched OpenJDK, Mixed Mode CPU Flame Graph: vert.x

Needs -XX:+PreserveFramePointer and perf-map-agent

Kernel

Java

JVM
$ sar -n DEV

Linux 3.13.0-49-generic (prod0141)  06/06/2015  _x86_64_  (16 CPU)

<table>
<thead>
<tr>
<th>Time</th>
<th>Interface</th>
<th>rxpck/s</th>
<th>txpck/s</th>
<th>rxkB/s</th>
<th>txkB/s</th>
<th>rxcmp/s</th>
<th>txcmp/s</th>
<th>rxmcst/s</th>
<th>%ifutil</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:00:01 AM</td>
<td>IFACE</td>
<td>rxpck/s</td>
<td>txpck/s</td>
<td>rxkB/s</td>
<td>txkB/s</td>
<td>rxcmp/s</td>
<td>txcmp/s</td>
<td>rxmcst/s</td>
<td>%ifutil</td>
</tr>
<tr>
<td>12:05:01 AM</td>
<td>eth0</td>
<td>4824.26</td>
<td>3941.37</td>
<td>919.57</td>
<td>15706.14</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12:05:01 AM</td>
<td>lo</td>
<td>23913.29</td>
<td>23913.29</td>
<td>17677.23</td>
<td>17677.23</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12:15:01 AM</td>
<td>eth0</td>
<td>4507.22</td>
<td>3749.46</td>
<td>909.03</td>
<td>12481.74</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12:15:01 AM</td>
<td>lo</td>
<td>23456.94</td>
<td>23456.94</td>
<td>14424.28</td>
<td>14424.28</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12:25:01 AM</td>
<td>eth0</td>
<td>10372.37</td>
<td>9990.59</td>
<td>1219.22</td>
<td>27788.19</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12:25:01 AM</td>
<td>lo</td>
<td>25725.15</td>
<td>25725.15</td>
<td>29372.20</td>
<td>29372.20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12:35:01 AM</td>
<td>eth0</td>
<td>4729.53</td>
<td>3899.14</td>
<td>914.74</td>
<td>12773.97</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12:35:01 AM</td>
<td>lo</td>
<td>23943.61</td>
<td>23943.61</td>
<td>14740.62</td>
<td>14740.62</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

• System Activity Reporter. Archive of metrics, eg:

• Metrics are also in Atlas and Vector
• Linux sar is well designed: units, groups
perf-tools

- Some front-ends to Linux ftrace & perf_events
  - Advanced, custom kernel observability when needed (rare)
  - [https://github.com/brendangregg/perf-tools](https://github.com/brendangregg/perf-tools)
  - Unsupported hacks: see WARNINGs
- ftrace
  - First added to Linux 2.6.27
  - A collection of capabilities, used via /sys/kernel/debug/tracing/
- perf_events
  - First added to Linux 2.6.31
  - Tracer/profiler multi-tool, used via "perf" command
perf-tools: funccount

- Eg, count a kernel function call rate:

```
# ./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

- Other perf-tools can then instrument these in more detail
perf-tools (so far...)
eBPF

- Currently being integrated. Efficient (JIT) in-kernel maps.
- Measure latency, heat maps, ...
eBPF

eBPF will make a profound difference to monitoring on Linux systems

There will be an arms race to support it, post Linux 4.1+
If it's not on your roadmap, it should be
Summary
Requirements

- Acceptable T&Cs
- Acceptable technical debt
- Known overhead
- Low overhead
- Scalable
- Useful
Methodologies

Support for:
• Workload Characterization
• The USE Method
• ...

Not starting with metrics in search of uses
Desirables
With full eBPF support

Linux has awesome instrumentation: use it!
Links & References

- **Netflix Vector**
  - [https://github.com/netflix/vector](https://github.com/netflix/vector)

- **Netflix Atlas**

- **Heat Maps**
  - [http://www.brendangregg.com/heatmaps.html](http://www.brendangregg.com/heatmaps.html)

- **Flame Graphs**
  - [http://www.brendangregg.com/flamegraphs.html](http://www.brendangregg.com/flamegraphs.html)

- **Frequency Trails**: [http://www.brendangregg.com/frequencytrails.html](http://www.brendangregg.com/frequencytrails.html)

- **Methodology**
  - [http://www.brendangregg.com/methodology.html](http://www.brendangregg.com/methodology.html)

- **perf-tools**: [https://github.com/brendangregg/perf-tools](https://github.com/brendangregg/perf-tools)


- **Images**:
  - horse: Microsoft Powerpoint clip art
  - gauge: [https://github.com/thlorenz/d3-gauge](https://github.com/thlorenz/d3-gauge)
  - eBPF ponycorn: Deirdré Straughan & General Zoi’s Pony Creator
Thanks

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