Container Performance Analysis

Brendan Gregg

Sr. Performance Architect, Netflix
Take Aways

Identify bottlenecks:
1. In the host vs container, using system metrics
2. In application code on containers, using CPU flame graphs
3. Deeper in the kernel, using tracing tools

Focus of this talk is how containers work in Linux (will demo on 4.9)
I will include some Docker specifics, and start with a Netflix summary (Titus)
1. Titus
Containers at Netflix

Summary slides from the Titus team
Titus

- Cloud runtime platform for container jobs
- Scheduling
  - Service & batch job management
  - Advanced resource management across elastic shared resource pool
- Container Execution
  - Docker and AWS EC2 Integration
    - Adds VPC, security groups, EC2 metadata, IAM roles, S3 logs, …
  - Integration with Netflix infrastructure
Current Titus Scale

- Deployed across multiple AWS accounts & three regions
- Over 2,500 instances (Mostly M4.4xls & R3.8xls)
- Over a week period launched over 1,000,000 containers
Titus Use Cases

- **Service**
  - Stream Processing (Flink)
  - UI Services (Node.JS single core)
  - Internal dashboards

- **Batch**
  - Algorithm training, personalization & recommendations
  - Adhoc reporting
  - Continuous integration builds

- **Queued worker model**
  - Media encoding
Container Performance @Netflix

- Ability to **scale** and **balance** workloads with EC2 and Titus
  - Can already solve many perf issues

- Performance needs:
  - **Application analysis**: using CPU flame graphs with containers
  - **Host tuning**: file system, networking, sysctl's, …
  - **Container analysis and tuning**: cgroups, GPUs, …
  - **Capacity planning**: reduce over provisioning
2. Container Background
And Strategy
Namespaces

Restricting visibility

Namespaces:
- cgroup
- ipc
- mnt
- net
- pid
- user
- uts

PID namespaces

Host

PID 1

1237

... 1 (1238)

2 (1241)

PID namespace 1

Kernel
Control Groups

Restricting usage

cgroups:
- blkio
- cpu,cpuacct
- cpuset
- devices
- hugetlb
- memory
- net_cls,net_prio
- pids
- …

CPU cgroups

container 1

cpu cgroup 1

container 2

container 3

… 2

… 3

CPUs

…
Linux Containers

Container = combination of namespaces & cgroups
cgroup v1

cpu,cpuacct:
- **cap CPU usage** (hard limit). e.g. 1.5 CPUs.
- **CPU shares**. e.g. 100 shares.
- usage statistics (cpuacct)

memory:
- **limit** and **kmem limit** (maximum bytes)
- **OOM control**: enable/disable
- usage statistics

blkio (block I/O):
- **weights** (like shares)
- **IOPS/tput** caps per storage device
- statistics

Docker:
- **--cpus (1.13)**
- **--cpu-shares**
- **--memory --kernel-memory**
- **--oom-kill-disable**
CPU Shares

Container's CPU limit = 100% \times \frac{\text{container's shares}}{\text{total busy shares}}

This lets a container use other tenant's idle CPU (aka "bursting"), when available.

Container's minimum CPU limit = 100\% \times \frac{\text{container's shares}}{\text{total allocated shares}}

Can make analysis tricky. Why did perf regress? Less bursting available?
cgroup v2

- Major rewrite has been happening: cgroups v2
  - Supports nested groups, better organization and consistency
  - Some already merged, some not yet (e.g. CPU)
- See docs/talks by maintainer Tejun Heo (Facebook)
- References:
  - https://www.kernel.org/doc/Documentation/cgroup-v2.txt
  - https://lwn.net/Articles/679786/
Container OS Configuration

File systems
• Containers may be setup with aufs/overlay on top of another FS
• See "in practice" pages and their performance sections from https://docs.docker.com/engine/userguide/storagedriver/

Networking
• With Docker, can be bridge, host, or overlay networks
• Overlay networks have come with significant performance cost
Analysis Strategy

Performance analysis with containers:
- One kernel
- Two perspectives
- Namespaces
- cgroups

Methodologies:
- USE Method
- Workload characterization
- Checklists
- Event tracing
USE Method

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

For example, CPUs:
- Utilization: time busy
- Saturation: run queue length or latency
- Errors: ECC errors, etc.

Can be applied to hardware resources and software resources (cgroups)
3. Host Tools
And Container Awareness

... if you have host access
Host Analysis Challenges

- PIDs in host don't match those seen in containers
- Symbol files aren't where tools expect them
- The kernel currently doesn't have a container ID
I'll demo CLI tools

It's the lowest common denominator

You may usually use GUIs (like we do). They source the same metrics.
A refresher of basics... Not container specific.

This will, however, solve many issues!

Containers are often not the problem.
Linux Perf Tools

Where can we begin?
Host Perf Analysis in 60s

1. uptime                                ➤ load averages
2. dmesg | tail                          ➤ kernel errors
3. vmstat 1                              ➤ overall stats by time
4. mpstat -P ALL 1                       ➤ CPU balance
5. pidstat 1                             ➤ process usage
6. iostat -xz 1                          ➤ disk I/O
7. free -m                                ➤ memory usage
8. sar -n DEV 1                          ➤ network I/O
9. sar -n TCP,ETCP 1                     ➤ TCP stats
10. top                                  ➤ check overview

## USE Method: Host Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Utilization</th>
<th>Saturation</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td><code>mpstat -P ALL 1, sum non-idle fields</code></td>
<td><code>vmstat 1,&quot;r&quot;</code></td>
<td><code>perf</code></td>
</tr>
<tr>
<td>Memory Capacity</td>
<td><code>free -m, &quot;used&quot;/&quot;total&quot;</code></td>
<td>`vmstat 1,&quot;si&quot;+&quot;so&quot;; dmesg</td>
<td>grep killed`</td>
</tr>
<tr>
<td>Storage I/O</td>
<td><code>iostat --xz 1, &quot;%util&quot;</code></td>
<td><code>iostat --xnz 1, &quot;avgqu-sz&quot; &gt; 1</code></td>
<td><code>/sys/.../ioerr_cnt; smartctl</code></td>
</tr>
<tr>
<td>Network</td>
<td><code>nicstat,&quot;%Util&quot;</code></td>
<td><code>ifconfig,&quot;overrunns&quot;; netstat --s &quot;retrans...&quot;</code></td>
<td><code>ifconfig, &quot;errors&quot;</code></td>
</tr>
</tbody>
</table>

These should be in your monitoring GUI. Can do other resources too (busses, ...)

[Image: dockericon.png]
Event Tracing: e.g. iosnoop

Disk I/O events with latency (from perf-tools; also in bcc/BPF as biosnoop)

```bash
# ./iosnoop
Tracing block I/O... Ctrl-C to end.
```

<table>
<thead>
<tr>
<th>COMM</th>
<th>PID</th>
<th>TYPE</th>
<th>DEV</th>
<th>BLOCK</th>
<th>BYTES</th>
<th>LATms</th>
</tr>
</thead>
<tbody>
<tr>
<td>supervise</td>
<td>1809</td>
<td>W</td>
<td>202,1</td>
<td>17039968</td>
<td>4096</td>
<td>1.32</td>
</tr>
<tr>
<td>supervise</td>
<td>1809</td>
<td>W</td>
<td>202,1</td>
<td>17039976</td>
<td>4096</td>
<td>1.30</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8457608</td>
<td>4096</td>
<td>7.53</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8470336</td>
<td>4096</td>
<td>14.90</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8470368</td>
<td>4096</td>
<td>0.27</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8470784</td>
<td>4096</td>
<td>7.74</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8470360</td>
<td>4096</td>
<td>0.25</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8469968</td>
<td>4096</td>
<td>0.24</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8470240</td>
<td>4096</td>
<td>0.24</td>
</tr>
<tr>
<td>tar</td>
<td>14794</td>
<td>RM</td>
<td>202,1</td>
<td>8470392</td>
<td>4096</td>
<td>0.23</td>
</tr>
</tbody>
</table>
Event Tracing: e.g. zfsslower

```
# /usr/share/bcc/tools/zfsslower 1
Tracing ZFS operations slower than 1 ms

<table>
<thead>
<tr>
<th>TIME</th>
<th>COMM</th>
<th>PID</th>
<th>T</th>
<th>BYTES</th>
<th>OFF_KB</th>
<th>LAT(ms)</th>
<th>FILENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>23:44:40</td>
<td>java</td>
<td>31386</td>
<td>O</td>
<td>0</td>
<td>0</td>
<td>8.02</td>
<td>solrFeatures.txt</td>
</tr>
<tr>
<td>23:44:53</td>
<td>java</td>
<td>31386</td>
<td>W</td>
<td>8190</td>
<td>1812222</td>
<td>36.24</td>
<td>solrFeatures.txt</td>
</tr>
<tr>
<td>23:44:59</td>
<td>java</td>
<td>31386</td>
<td>W</td>
<td>8192</td>
<td>1826302</td>
<td>20.28</td>
<td>solrFeatures.txt</td>
</tr>
<tr>
<td>23:44:59</td>
<td>java</td>
<td>31386</td>
<td>W</td>
<td>8191</td>
<td>1826846</td>
<td>28.15</td>
<td>solrFeatures.txt</td>
</tr>
<tr>
<td>23:45:00</td>
<td>java</td>
<td>31386</td>
<td>W</td>
<td>8192</td>
<td>1831015</td>
<td>32.17</td>
<td>solrFeatures.txt</td>
</tr>
<tr>
<td>23:45:15</td>
<td>java</td>
<td>31386</td>
<td>O</td>
<td>0</td>
<td>0</td>
<td>27.44</td>
<td>solrFeatures.txt</td>
</tr>
<tr>
<td>23:45:56</td>
<td>dockerd</td>
<td>3599</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.03</td>
<td>.tmp-a66ce9aad...</td>
</tr>
<tr>
<td>23:46:16</td>
<td>java</td>
<td>31386</td>
<td>W</td>
<td>31</td>
<td>0</td>
<td>36.28</td>
<td>solrFeatures.txt</td>
</tr>
</tbody>
</table>
```

- This is from our production Titus system (Docker).
- File system latency is a better pain indicator than disk latency.
- zfsslower (and btrfs*, etc) are in bcc/BPF. Can exonerate FS/disks.
Latency Histograms: e.g. btrfsdist

```bash
# ./btrfsdist
Tracing btrfs operation latency... Hit Ctrl-C to end.
^C
```

<table>
<thead>
<tr>
<th>operation = 'read'</th>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-1</td>
<td>192529</td>
<td>**************************************************</td>
</tr>
<tr>
<td></td>
<td>2-3</td>
<td>72337</td>
<td>*****************</td>
</tr>
<tr>
<td></td>
<td>4-7</td>
<td>5620</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>8-15</td>
<td>1026</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16-31</td>
<td>369</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32-63</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64-127</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>128-255</td>
<td>975</td>
<td></td>
</tr>
<tr>
<td></td>
<td>256-511</td>
<td>524</td>
<td></td>
</tr>
<tr>
<td></td>
<td>512-1023</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1024-2047</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2048-4095</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4096-8191</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
```

probably cache reads

probably cache misses (flash reads)
Latency Histograms: e.g. btrfsdist

- From a test Titus system (Docker).
- Histograms show modes, outliers. Also in bcc/BPF (with other FSes).
- Latency heat maps: http://queue.acm.org/detail.cfm?id=1809426

```
operation = 'write'

<table>
<thead>
<tr>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>276</td>
<td></td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>32125</td>
<td>***********</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>111253</td>
<td>*****************************************</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>59154</td>
<td>*******************</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>5463</td>
<td>*</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>612</td>
<td></td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```
3.2. Host Containers & cgroups

Inspecting containers from the host
Worth checking namespace config before analysis:

<table>
<thead>
<tr>
<th>CONTAINER</th>
<th>NAME</th>
<th>PID</th>
<th>PATH</th>
<th>CGROUP</th>
<th>IPC</th>
<th>MNT</th>
<th>NET</th>
<th>PID</th>
<th>USER</th>
<th>UTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>titusagent-mainvpc-m</td>
<td>1</td>
<td>systemctl</td>
<td>4026531835</td>
<td>4026531839</td>
<td>4026531840</td>
<td>4026532533</td>
<td>4026531836</td>
<td>4026531837</td>
<td>4026531838</td>
</tr>
<tr>
<td>b27909cd6dd1</td>
<td>Titus-1435830-worker</td>
<td>37280</td>
<td>svscanboot</td>
<td>4026531835</td>
<td>4026533387</td>
<td>4026533385</td>
<td>4026532931</td>
<td>4026533388</td>
<td>4026531837</td>
<td>4026533386</td>
</tr>
<tr>
<td>dcf3e506de45</td>
<td>Titus-1392192-worker</td>
<td>27992</td>
<td>/apps/spaas/spaa</td>
<td>4026531835</td>
<td>4026533354</td>
<td>4026533352</td>
<td>4026532991</td>
<td>4026533355</td>
<td>4026531837</td>
<td>4026533353</td>
</tr>
<tr>
<td>370a3f041f36</td>
<td>Titus-1243558-worker</td>
<td>98602</td>
<td>/apps/spaas/spaa</td>
<td>4026531835</td>
<td>4026533290</td>
<td>4026533288</td>
<td>4026533223</td>
<td>4026533291</td>
<td>4026531837</td>
<td>4026533289</td>
</tr>
<tr>
<td>af7549c76d9a</td>
<td>Titus-1243553-worker</td>
<td>97972</td>
<td>/apps/spaas/spaa</td>
<td>4026531835</td>
<td>4026533216</td>
<td>4026533214</td>
<td>4026533149</td>
<td>4026533217</td>
<td>4026531837</td>
<td>4026533215</td>
</tr>
<tr>
<td>dc27769a9b9c</td>
<td>Titus-1243546-worker</td>
<td>97356</td>
<td>/apps/spaas/spaa</td>
<td>4026531835</td>
<td>4026533142</td>
<td>4026533140</td>
<td>4026533075</td>
<td>4026533143</td>
<td>4026531837</td>
<td>4026533141</td>
</tr>
<tr>
<td>e1bd6e189dc3</td>
<td>Titus-1243517-worker</td>
<td>96733</td>
<td>/apps/spaas/spaa</td>
<td>4026531835</td>
<td>4026533068</td>
<td>4026533066</td>
<td>4026533001</td>
<td>4026533069</td>
<td>4026531837</td>
<td>4026533067</td>
</tr>
<tr>
<td>ab45227dceea</td>
<td>Titus-1243516-worker</td>
<td>96173</td>
<td>/apps/spaas/spaa</td>
<td>4026531835</td>
<td>4026532920</td>
<td>4026532918</td>
<td>4026532830</td>
<td>4026532921</td>
<td>4026531837</td>
<td>4026532919</td>
</tr>
</tbody>
</table>

- A POC "docker ps --namespaces" tool. NS shared with root in red.
- https://github.com/docker/docker/issues/32501
# systemd-cgtop

A "top" for cgroups:

<table>
<thead>
<tr>
<th>Control Group</th>
<th>Tasks</th>
<th>%CPU</th>
<th>Memory</th>
<th>Input/s</th>
<th>Output/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>-</td>
<td>798.2</td>
<td>45.9G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/docker</td>
<td>1082</td>
<td>790.1</td>
<td>42.1G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/docker/dfc3a...9d28fc4a1c72bbafe4a24834</td>
<td>200</td>
<td>610.5</td>
<td>24.0G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/docker/370a3...e64ca01198f1e843ade7ce21</td>
<td>170</td>
<td>174.0</td>
<td>3.0G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice</td>
<td>748</td>
<td>5.3</td>
<td>4.1G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/daemontools.service</td>
<td>422</td>
<td>4.0</td>
<td>2.8G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/docker/dc277...42ab0603b7da2ac8af67996b</td>
<td>160</td>
<td>2.5</td>
<td>2.3G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice</td>
<td>5</td>
<td>2.0</td>
<td>34.5M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice/user-0.slice</td>
<td>5</td>
<td>2.0</td>
<td>15.7M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice/u.../slice/session-c26.scope</td>
<td>3</td>
<td>2.0</td>
<td>13.3M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/docker/ab452...c946f8447f2a4184f3ccff2a</td>
<td>174</td>
<td>1.0</td>
<td>6.3G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/docker/e18bd...26ffdd7368b870aa3d1deb7a</td>
<td>156</td>
<td>0.8</td>
<td>2.9G</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

[...]

<table>
<thead>
<tr>
<th>CONTAINER</th>
<th>CPU %</th>
<th>MEM USAGE / LIMIT</th>
<th>MEM %</th>
<th>NET I/O</th>
<th>BLOCK I/O</th>
<th>PIDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>353426a09db1</td>
<td>526.81%</td>
<td>4.061 GiB / 8.5 GiB</td>
<td>47.78%</td>
<td>0 B / 0 B</td>
<td>2.818 MB / 0 B</td>
<td>247</td>
</tr>
<tr>
<td>6bf166a66e08</td>
<td>303.82%</td>
<td>3.448 GiB / 8.5 GiB</td>
<td>40.57%</td>
<td>0 B / 0 B</td>
<td>2.032 MB / 0 B</td>
<td>267</td>
</tr>
<tr>
<td>58dcf8aed0a7</td>
<td>41.01%</td>
<td>1.322 GiB / 2.5 GiB</td>
<td>52.89%</td>
<td>0 B / 0 B</td>
<td>0 B / 0 B</td>
<td>229</td>
</tr>
<tr>
<td>61061566ffe5</td>
<td>85.92%</td>
<td>220.9 MiB / 3.023 GiB</td>
<td>7.14%</td>
<td>0 B / 0 B</td>
<td>43.4 MB / 0 B</td>
<td>61</td>
</tr>
<tr>
<td>bdc721460293</td>
<td>2.69%</td>
<td>1.204 GiB / 3.906 GiB</td>
<td>30.82%</td>
<td>0 B / 0 B</td>
<td>4.35 MB / 0 B</td>
<td>66</td>
</tr>
<tr>
<td>6c80ed61ae63</td>
<td>477.45%</td>
<td>557.7 MiB / 8 GiB</td>
<td>6.81%</td>
<td>0 B / 0 B</td>
<td>9.257 MB / 0 B</td>
<td>19</td>
</tr>
<tr>
<td>337292fb5b64</td>
<td>89.05%</td>
<td>766.2 MiB / 8 GiB</td>
<td>9.35%</td>
<td>0 B / 0 B</td>
<td>5.493 MB / 0 B</td>
<td>19</td>
</tr>
<tr>
<td>b652ede9a605</td>
<td>173.50%</td>
<td>689.2 MiB / 8 GiB</td>
<td>8.41%</td>
<td>0 B / 0 B</td>
<td>6.48 MB / 0 B</td>
<td>19</td>
</tr>
<tr>
<td>d7cd2599291f</td>
<td>504.28%</td>
<td>673.2 MiB / 8 GiB</td>
<td>8.22%</td>
<td>0 B / 0 B</td>
<td>12.58 MB / 0 B</td>
<td>19</td>
</tr>
<tr>
<td>05bf9f3e0d13</td>
<td>314.46%</td>
<td>711.6 MiB / 8 GiB</td>
<td>8.69%</td>
<td>0 B / 0 B</td>
<td>7.942 MB / 0 B</td>
<td>19</td>
</tr>
<tr>
<td>09082f005755</td>
<td>142.04%</td>
<td>693.9 MiB / 8 GiB</td>
<td>8.47%</td>
<td>0 B / 0 B</td>
<td>8.081 MB / 0 B</td>
<td>19</td>
</tr>
<tr>
<td>bd45a3e1ce16</td>
<td>190.26%</td>
<td>538.3 MiB / 8 GiB</td>
<td>6.57%</td>
<td>0 B / 0 B</td>
<td>10.6 MB / 0 B</td>
<td>19</td>
</tr>
</tbody>
</table>

Loris Degioanni demoed a similar sysdigcloud view yesterday (needs the sysdig kernel agent)
In the host, top shows all processes. **Currently doesn't show a container ID.**

```
# top - 22:46:53 up 36 days, 59 min, 1 user, load average: 5.77, 5.61, 5.63
Tasks: 1067 total, 1 running, 1046 sleeping, 0 stopped, 20 zombie
%Cpu(s): 34.8 us, 1.8 sy, 0.0 ni, 61.3 id, 0.0 wa, 0.0 hi, 1.9 si, 0.1 st
KiB Mem: 65958552 total, 12418448 free, 49247988 used, 4292116 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 13101316 avail Mem

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</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>28321</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>33.126g</td>
<td>0.023t</td>
<td>37564</td>
<td>S</td>
<td>621.1</td>
<td>38.2</td>
<td>35184:09</td>
<td>java</td>
</tr>
<tr>
<td>97712</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>11.445g</td>
<td>2.333g</td>
<td>37084</td>
<td>S</td>
<td>3.1</td>
<td>3.7</td>
<td>404:27.90</td>
<td>java</td>
</tr>
<tr>
<td>98306</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>12.149g</td>
<td>3.060g</td>
<td>36996</td>
<td>S</td>
<td>2.0</td>
<td>4.9</td>
<td>194:21.10</td>
<td>java</td>
</tr>
<tr>
<td>96511</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>15.567g</td>
<td>6.313g</td>
<td>37112</td>
<td>S</td>
<td>1.7</td>
<td>10.0</td>
<td>168:07.44</td>
<td>java</td>
</tr>
<tr>
<td>5283</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>1643676</td>
<td>100092</td>
<td>94184</td>
<td>S</td>
<td>1.0</td>
<td>0.2</td>
<td>401:36.16</td>
<td>mesos-slave</td>
</tr>
<tr>
<td>2079</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>9512</td>
<td>132</td>
<td>12</td>
<td>S</td>
<td>0.7</td>
<td>0.0</td>
<td>220:07.75</td>
<td>rngd</td>
</tr>
<tr>
<td>5272</td>
<td>titusag+</td>
<td>20</td>
<td>0</td>
<td>10.473g</td>
<td>1.611g</td>
<td>23488</td>
<td>S</td>
<td>0.7</td>
<td>2.6</td>
<td>1934:44</td>
<td>java</td>
</tr>
</tbody>
</table>

[...]

... remember, there is no container ID in the kernel yet.
htop can add a CGROUP field, but, can truncate important info:

<table>
<thead>
<tr>
<th>CGROUP</th>
<th>PID</th>
<th>USER</th>
<th>PRI</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S CPU%</th>
<th>MEM%</th>
<th>TIME+</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>:pids:/docker/ 28321 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>524. 38.2</td>
<td>672h</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9982 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>44.4 38.2</td>
<td>17h00:41</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9985 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 R 41.9</td>
<td>38.2</td>
<td>16h44:51</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9979 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>41.2 38.2</td>
<td>17h01:35</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9980 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>39.3 38.2</td>
<td>16h59:17</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9981 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>39.3 38.2</td>
<td>17h01:32</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9984 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>37.3 38.2</td>
<td>16h49:03</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9983 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 R 35.4 38.2</td>
<td>16h54:31</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9986 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>35.4 38.2</td>
<td>17h05:30</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:name=systemd:/user.slice/user-0.slice/session-c31.scope?</td>
<td>74066 root</td>
<td>20</td>
<td>0</td>
<td>27620</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 9998 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 R 28.3 38.2</td>
<td>11h38:03</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 10001 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>27.7 38.2</td>
<td>11h38:59</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:name=systemd:/system.slice/daemontools.service?</td>
<td>5272 titusagen 20 0 10.5G 1650M 23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:pids:/docker/ 10002 root</td>
<td>20</td>
<td>0</td>
<td>33.1G</td>
<td>24.0G</td>
<td>37564 S</td>
<td>25.1 38.2</td>
<td>11h40:37</td>
<td>/apps/java</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Can fix, but that would be Docker + cgroup-v1 specific. Still need a kernel CID.
Host PID -> Container ID

... who does that (CPU busy) PID 28321 belong to?

```
# grep 28321 /sys/fs/cgroup/cpu,cpuacct/docker/*/tasks | cut -d/ -f7
dcf3a506de453107715362f6c9ba9056f6cfc6e769d28fc4a1c72bbafe4a24834
```

- Only works for Docker, and that cgroup v1 layout. Some Linux commands:

```
# ls -l /proc/27992/ns/*
lrwxrwxrwx 1 root root 0 Apr 13 20:49 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 Apr 13 20:49 ipc   -> ipc:[4026533354]
lrwxrwxrwx 1 root root 0 Apr 13 20:49 mnt   -> mnt:[4026533352]
```

```
# cat /proc/27992/cgroup
11:freezer:/docker/dcf3a506de453107715362f6c9ba9056f6cfc6e769d28fc4a1c72bbafe4a24834
10:blkio:/docker/dcf3a506de453107715362f6c9ba9056f6cfc6e769d28fc4a1c72bbafe4a24834
9:perf_event:/docker/dcf3a506de453107715362f6c9ba9056f6cfc6e769d28fc4a1c72bbafe4a24834
```

[...]

[dockercon17 logo]
nsenter Wrapping

... what hostname is PID 28321 running on?

```bash
# nsenter -t 28321 -u hostname
titus-1392192-worker-14-16
```

- Can namespace enter:
  - `-m`: mount
  - `-u`: uts
  - `-i`: ipc
  - `-n`: net
  - `-p`: pid
  - `-U`: user
- Bypasses cgroup limits, and seccomp profile (allowing syscalls)
  - For Docker, you can enter the container more completely with: docker exec -it CID command
- Handy nsenter one-liners:
  - `nsenter -t PID -u hostname` container hostname
  - `nsenter -t PID -n netstat -i` container netstat
  - `nsenter -t PID -m -p df -h` container file system usage
  - `nsenter -t PID -p top` container top
nsenter: Host -> Container top

... Given PID 28321, running top for its container by entering its namespaces:

```
# nsenter -t 28321 -m -p top
```

```
top - 18:16:13 up 36 days, 20:28, 0 users, load average: 5.66, 5.29, 5.28
Tasks: 6 total, 1 running, 5 sleeping, 0 stopped, 0 zombie
%Cpu(s): 30.5 us, 1.7 sy, 0.0 ni, 65.9 id, 0.0 wa, 0.0 hi, 1.8 si, 0.1 st
KiB Mem: 65958552 total, 54664124 used, 11294428 free, 164232 buffers
KiB Swap: 0 total, 0 used, 0 free. 1592372 cached Mem
```

```
P ID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
301 root 20 0 33.127g 0.023t 37564 S 537.3 38.2 40269:41 java
1 root 20 0 21404 2236 1812 S 0.0 0.0 4:15.11 bash
8788 root 20 0 21464 1720 1348 R 0.0 0.0 0:00.00 top
```

Note that it is PID 301 in the container. Can also see this using:

```
# grep NSpid /proc/28321/status
NSpid: 28321 301
```
perf: CPU Profiling

Can run system-wide (-a), match a pid (-p), or cgroup (-G, if it works)

```bash
# perf record -F 49 -a -g -- sleep 30
# perf script
Failed to open /lib/x86_64-linux-gnu/libc-2.19.so, continuing without symbols
Failed to open /tmp/perf-28321.map, continuing without symbols
```

- Current symbol translation gotchas (up to 4.10-ish):
  - perf can't find /tmp/perf-PID.map files in the host, and the PID is different
  - perf can't find container binaries under host paths (what /usr/bin/java?)
- Can copy files to the host, map PIDs, then run perf script/report:
  - [http://batey.info/docker-jvm-flamegraphs.html](http://batey.info/docker-jvm-flamegraphs.html)
- Can nsenter (-m -u -i -n -p) a "power" shell, and then run "perf -p PID"
- perf should be fixed to be namespace aware (like bcc was, PR#1051)
**CPU Flame Graphs**

- See previous slide for getting perf symbols to work
- From the host, can study all containers, as well as container overheads

```
  git clone --depth 1 https://github.com/brendangregg/FlameGraph
  cd FlameGraph
  perf record -F 49 -a -g -- sleep 30
  perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > perf.svg
```

Kernel TCP/IP stack
Look in areas like this to find and quantify overhead (cgroup throttles, FS layers, networking, etc). It's likely small and hard to find.

Java, missing stacks (need -XX:+PreserveFramePointer)
# cd /sys/fs/cgroup/cpu,cpuacct/docker/02a7cf65f82e3f3e75283944caa4462e82f8f6ff5a7c9a...
# ls
cgroup.clone_children cpuacct.usage_all ccpuacct.usage_sys cpu.shares
cgroup.procs ccpuacct.usage_percpu ccpuacct.usage_user cpu.stat
cpuacct.stat ccpuacct.usage_percpu_sys cpu.cfs_period_us notify_on_release
cpuacct.usage ccpuacct.usage_percpu_user cpu.cfs_quota_us tasks

# cat cpuacct.usage
1615816262506

# cat cpu.stat
nr_periods 507
nr_throttled 74
throttled_time 3816445175

total time throttled (nanoseconds). saturation metric. average throttle time = throttled_time / nr_throttled

- https://blog.docker.com/2013/10/gathering-lxc-docker-containers-metrics/

Note: grep cgroup /proc/mounts to check where these are mounted

These metrics should be included in performance monitoring GUIs
Netflix Atlas

Cloud-wide monitoring of containers (and instances)

Fetches cgroup metrics via Intel snap

https://github.com/netflix/Atlas
Netflix Vector

Our per-instance analyzer
Has per-container metrics
https://github.com/Netflix/vector
Intel snap

A metric collector used by monitoring GUIs
https://github.com/intelsdi-x/snap

Has a Docker plugin to read cgroup stats

There's also a collectd plugin:
https://github.com/bobrik/collectd-docker
3.3. Let's Play a Game

Host or Container?
(or neither?)
Game Scenario 1

Container user claims they have a CPU performance issue
• Container has a CPU cap and CPU shares configured
• There is idle CPU on the host
• Other tenants are CPU busy
• /sys/fs/cgroup/.../cpu.stat \texttt{\rightarrow} throttled_time is increasing
• /proc/PID/status nonvoluntary_ctxt_switches is increasing
• Container CPU usage equals its cap (clue: this is not really a clue)
Game Scenario 2

Container user claims they have a CPU performance issue

• Container has a CPU cap and CPU shares configured
• There is no idle CPU on the host
• Other tenants are CPU busy
• `/sys/fs/cgroup/.../cpu.stat -> throttled_time is not increasing`
• `/proc/PID/status nonvoluntary_ctxt_switches is increasing`
Game Scenario 3

Container user claims they have a CPU performance issue
• Container has CPU shares configured
• There is no idle CPU on the host
• Other tenants are CPU busy
• /sys/fs/cgroup/.../cpu.stat -> throttled_time is not increasing
• /proc/PID/status nonvoluntary_ctxt_switches is not increasing much

Experiments to confirm conclusion?
Methodology: Reverse Diagnosis

Enumerate possible outcomes, and work backwards to the metrics needed for diagnosis.

E.g. CPU performance outcomes:

A. physical CPU throttled
B. cap throttled
C. shares throttled (assumes physical CPU limited as well)
D. not throttled
**CPU Bottleneck Identification**

- **throttled_time increasing?**
  - **Y** → cap throttled
  - **N**
    - **N**
      - **Y** → host has idle CPU?
        - **Y** (but dig further)
          - not throttled
        - **N**
          - **N**
            - **Y** → all other tenants idle?
              - **Y** → physical CPU throttled
              - **N**
                - share throttled
4. Guest Tools
And Container Awareness

... if you only have guest access
Guest Analysis Challenges

• Some resource metrics are for the container, some for the host. Confusing!
• May lack system capabilities or syscalls to run profilers and tracers
Can see host's CPU devices, but only container (pid namespace) processes:

**container# uptime**
20:17:19 up 45 days, 21:21, 0 users, load average: 5.08, 3.69, 2.22

**container# mpstat 1**
Linux 4.9.0 (02a7cf65f82e) 04/14/17 __x86_64__ (8 CPU)

<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>%gnice</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>20:17:26</td>
<td>all</td>
<td>51.00</td>
<td>0.00</td>
<td>12.28</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>36.72</td>
</tr>
<tr>
<td>20:17:27</td>
<td>all</td>
<td>50.88</td>
<td>0.00</td>
<td>12.31</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>36.81</td>
</tr>
</tbody>
</table>

^C

**Average:**
all  50.94 | 0.00 | 12.30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 36.76

**container# pidstat 1**
Linux 4.9.0 (02a7cf65f82e) 04/14/17 __x86_64__ (8 CPU)

<table>
<thead>
<tr>
<th>Time</th>
<th>UID</th>
<th>PID</th>
<th>%usr</th>
<th>%system</th>
<th>%guest</th>
<th>%CPU</th>
<th>CPU Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>20:17:33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20:17:34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 20:17:35 |     |     |      |         |        |      |             | [...]

Can see host's CPU devices, but only container (pid namespace) processes:

**load!**

**busy CPUs**

but this container is running nothing (we saw CPU usage from neighbors)
Memory

Can see host's memory:

<table>
<thead>
<tr>
<th>container# free -m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem:</td>
</tr>
<tr>
<td>total</td>
</tr>
<tr>
<td>15040</td>
</tr>
<tr>
<td>Swap:</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

container# perl -e '$a = "A" x 1_000_000_000'
Killed

host memory (this container is --memory=1g)

tries to consume ~2 Gbytes
Disks

Can see host's disk devices:

container# iostat -xz 1
avg-cpu: %user  %nice %system %iowait %steal %idle
         52.57  0.00  16.94  0.00  0.00  30.49

Device: rrqm/s  wrqm/s  r/s  w/s  rkB/s  wkB/s  avgrq-sz avgq-sz await r-await w-await svctm  %util
  xvdap1   0.00   7.00  0.00  2.00  0.00  36.00  36.00   0.00   2.00   0.00   2.00  0.00  36.00
  xvdb     0.00   0.00  200.00 0.00  3080.00  0.00  30.80   0.04   0.20   0.20   0.00  0.20  4.00
  xvdc     0.00   0.00  185.00 0.00  2840.00  0.00  30.70   0.04   0.24   0.24   0.00  0.24  4.40
  md0      0.00   0.00  385.00 0.00  5920.00  0.00  30.75   0.00   0.00   0.00   0.00  0.00  0.00

container# pidstat -d 1
Linux 4.9.0 (02a7cf65f82e) 04/18/17 _x86_64_ (8 CPU)

22:41:13      UID       PID  kB_rd/s  kB_wr/s  kB_ccwr/s  iodelay  Command
22:41:14      UID       PID  kB_rd/s  kB_wr/s  kB_ccwr/s  iodelay  Command
22:41:15      UID       PID  kB_rd/s  kB_wr/s  kB_ccwr/s  iodelay  Command

Can see host’s disk devices: host disk I/O but no container I/O
Network

Can't see host's network interfaces (network namespace):

```
container# sar -n DEV,TCP 1
Linux 4.9.0 (02a7cf65f82e) 04/14/17 _x86_64_ (8 CPU)

21:45:07        IFACE  rxpck/s  txpck/s  rxkB/s  txkB/s  rxcmp/s  txcmp/s  rxmcst/s  %ifutil
21:45:08        lo     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
21:45:08       eth0    0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00

21:45:08        active/s passive/s  iseg/s  oseg/s
21:45:08       0.00     0.00     0.00     0.00

21:45:08        IFACE  rxpck/s  txpck/s  rxkB/s  txkB/s  rxcmp/s  txcmp/s  rxmcst/s  %ifutil
21:45:09        lo     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
21:45:09      eth0     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00

21:45:08        active/s passive/s  iseg/s  oseg/s
21:45:09       0.00     0.00     0.00     0.00

[...]
```

host has heavy network I/O, container sees itself (idle)
Metrics Namespace

This confuses apps too: trying to bind on all CPUs, or using 25% of memory

- Including the JDK, which is unaware of container limits, covered yesterday by Fabiane Nardon

We could add a "metrics" namespace so the container only sees itself

- Or enhance existing namespaces to do this

If you add a metrics namespace, please consider adding an option for:

- `/proc/host/stats`: maps to host's `/proc/stats`, for CPU stats
- `/proc/host/diskstats`: maps to host's `/proc/diskstats`, for disk stats

As those host metrics can be useful, to identify/exonerate neighbor issues
perf: CPU Profiling

Needs capabilities to run from a container:

```
container# ./perf record -F 99 -a -g -- sleep 10
perf_event_open(..., PERF_FLAG_FD_CLOEXEC) failed with unexpected error 1 (Operation not permitted)
perf_event_open(..., 0) failed unexpectedly with error 1 (Operation not permitted)
Error: You may not have permission to collect system-wide stats.
```

Consider tweaking `/proc/sys/kernel/perf_event_paranoid`, which controls use of the performance events system by unprivileged users (without CAP_SYS_ADMIN).

The current value is 2:

- `-1`: Allow use of (almost) all events by all users
- `>= 0`: Disallow raw tracepoint access by users without CAP_IOC_LOCK
- `>= 1`: Disallow CPU event access by users without CAP_SYS_ADMIN
- `>= 2`: Disallow kernel profiling by users without CAP_SYS_ADMIN

Although, after setting `perf_event_paranoid` to `-1`, it prints the same error...
perf & Container Debugging

Debugging using strace from the host (as ptrace() is also blocked):

```
host# strace -fp 26450 ➔ bash PID, from which I then ran perf
[...]
[pid 27426] perf_event_open(0x2bfe498, -1, 0, -1, 0) = -1 EPERM (Operation not permitted)
[pid 27426] perf_event_open(0x2bfe498, -1, 0, -1, 0) = -1 EPERM (Operation not permitted)
[pid 27426] perf_event_open(0x2bfc1a8, -1, 0, -1, PERF_FLAG_FD_CLOEXEC) = -1 EPERM (Operation not permitted)
```

Many different ways to debug this.
https://docs.docker.com/engine/security/seccomp/#significant-syscalls-blocked-by-the-default-profile:

<table>
<thead>
<tr>
<th>syscall</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_by_handle_at</td>
<td>Cause of an old container breakout. Also gated by CAP_DAC_READ_SEARCH.</td>
</tr>
<tr>
<td>perf_event_open</td>
<td>Tracing/profiling syscall, which could leak a lot of information on the host.</td>
</tr>
<tr>
<td>personality</td>
<td>Prevent container from enabling BSD emulation. Not inherently dangerous, but poorly tested, potential for a lot of kernel vulns.</td>
</tr>
</tbody>
</table>
perf, cont.

- Can enable `perf_event_open()` with: `docker run --cap-add sys_admin`
  - Also need (for kernel symbols): `echo 0 > /proc/sys/kernel/kptr_restrict`
- `perf` then "works", and you can make **flame graphs**. But it sees all CPUs!?
  - `perf` needs to be "container aware", and only see the container's tasks.
    patch pending: https://lkml.org/lkml/2017/1/12/308
- Currently easier to run `perf` from the host (or secure "monitoring" container)
  - Via a secure monitoring agent, e.g. Netflix Vector -> CPU Flame Graph
  - See earlier slides for steps
5. Tracing
Advanced Analysis

... a few more examples
(iosnoop, zfsslower, and btrfsdist shown earlier)
Built-in Linux Tracers

Some front-ends:
• ftrace: [https://github.com/brendangregg/perf-tools](https://github.com/brendangregg/perf-tools)
• perf_events: used for **CPU flame graphs**
• eBPF (aka BPF): [https://github.com/iovisor/bcc](https://github.com/iovisor/bcc) (Linux 4.4+).
ftrace: Overlay FS Function Calls

Using ftrace via my perf-tools to count function calls in-kernel context:

```
# funccount '*ovl*'
Tracing "*ovl*"... Ctrl-C to end.
^C

<table>
<thead>
<tr>
<th>FUNC</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ovl_cache_free</td>
<td>3</td>
</tr>
<tr>
<td>ovl_xattr_get</td>
<td>3</td>
</tr>
<tr>
<td>[... ]</td>
<td></td>
</tr>
<tr>
<td>ovl_fill_merge</td>
<td>339</td>
</tr>
<tr>
<td>ovl_path_real</td>
<td>617</td>
</tr>
<tr>
<td>ovl_path_upper</td>
<td>777</td>
</tr>
<tr>
<td>ovl_update_time</td>
<td>777</td>
</tr>
<tr>
<td>ovl_permission</td>
<td>1408</td>
</tr>
<tr>
<td>ovl_d_real</td>
<td>1434</td>
</tr>
<tr>
<td>ovl_override_creds</td>
<td>1804</td>
</tr>
</tbody>
</table>

Ending tracing...
```

Each can be a target for further study with kprobes
ftrace: Overlay FS Function Tracing

Using kprobe (perf-tools) to trace ovl_fill_merge() args and stack trace

```bash
# kprobe -s 'p:ovl_fill_merge ctx=%di name=+0(%si):string'
Tracing kprobe ovl_fill_merge. Ctrl-C to end.

bash-16633 [000] d... 14390771.218973: ovl_fill_merge: (ovl_fill_merge+0x0/0x1f0 [overlay]) ctx=0xffffc90042477db0 name="iostat"

bash-16633 [000] d... 14390771.218981: <stack trace>
=> ovl_fill_merge
=> ext4.readdir
=> iterate_dir
=> ovl_dir_read_merged
=> ovl_iterate
=> iterate_dir
=> SyS_getdents
=> do_syscall_64
=> return_from_SYSCALL_64
[...]
```

Good for debugging, although dumping all events can cost too much overhead. ftrace has some solutions to this, BPF has more...
Enhanced BPF Tracing Internals

- **Observability Program**
  - BPF program
  - event config
  - per-event data
  - output
  - statistics

- **Kernel**
  - BPF
  - verifier
  - tracepoints
  - kprobes
  - uprobes
  - sampling, PMCs
  - perf_events

- Load
- Attach
- Async copy

Diagram illustrates the flow of data and operations between the BPF program and the kernel, highlighting key components and interactions.
This is an app in a Docker container on a system with idle CPU.

Tracing scheduler events can be costly (high rate), but this BPF program reduces cost by using in-kernel maps to summarize data, and only emits the "count" column to user space.
BPF: Scheduler Latency 2

```
host# runqlat -p 20228 10 1
Tracing run queue latency... Hit Ctrl-C to end.

<table>
<thead>
<tr>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>7</td>
<td>**</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>14</td>
<td>*****</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1024 -&gt; 2047</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2048 -&gt; 4095</td>
<td>5</td>
<td>**</td>
</tr>
<tr>
<td>4096 -&gt; 8191</td>
<td>6</td>
<td>**</td>
</tr>
<tr>
<td>8192 -&gt; 16383</td>
<td>28</td>
<td>***********</td>
</tr>
<tr>
<td>16384 -&gt; 32767</td>
<td>59</td>
<td>******************</td>
</tr>
<tr>
<td>32768 -&gt; 65535</td>
<td>99</td>
<td>******************</td>
</tr>
<tr>
<td>65536 -&gt; 131071</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>131072 -&gt; 262143</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>262144 -&gt; 524287</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```

Now other tenants are using more CPU, and this PID is throttled via CPU shares.

8 - 65ms delays
**BPF: Scheduler Latency 3**

```bash
host# runqlat --pidnss -m
Tracing run queue latency... Hit Ctrl-C to end.

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
BPF: Namespace-ing Tools

Walking from the task_struct to the PID namespace ID:

task_struct->nsproxy->pid_ns_for_children->ns.inum

• This is unstable, and could break between kernel versions. If it becomes a problem, we'll add a bpf_get_current_pidns()
• Does needs a *task, or bpf_get_current_task() (added in 4.8)
• Can also pull out cgroups, but gets trickier…
bcc (BPF) Perf Tools

- filetop
- filelife
- fileslow
- vfscount
- vfsstat

- cachestat
- cachetop
- dcstat
- dcsnoop
- mountsnoop

- trace
- argdist
- funcount
- funcslower
- funclatency
- stackcount
- profile

- mdflush

- btrfsdist
- btrfsllower
- ext4dist
- ext4slower
- xfsdist
- xfsllower
- zfssdist
- zfsslower

- biotop
- biosnoop
- biolatency
- bitezise

- opensnoop
- statsnoop
- syncsnoop

- c*
- java*
- node*
- php*
- python*
- ruby*

- ucalls
- uflow
- ugc
- uobjnew
- ustat
- uthreads

- mysqlqslower

- gethostlatency
- memleak
- sslsniff

- Other:
  - capable
  - syscount
  - killsnoop
  - execsnoop
  - pidpersec
  - cpu
dist
  - runqlat
  - runqlen
  - deadlock_detector
  - cpuunclaimed

- offcpu
  - uptime
  - wakeup
  - offwaketime

- softirqs
- oomkill
- memleak
- slb
- slratetop

- tcptop
- tcplife
- tcptracer
- tcpconnect
- tcpaccept
- tcpconnlat
- tcpretrans

- hardirqs
- ttysnoop

- dram
- llcstat
- profile

https://github.com/iovisor/bcc#tools
Docker Analysis & Debugging

If needed, dockerd can also be analyzed using:

- go execution tracer
- GODEBUG with gctrace and schedtrace
- gdb and Go runtime support
- perf profiling
- bcc/BPF and uprobes

Each has pros/cons. bcc/BPF can trace user & kernel events.
BPF: dockerd Go Function Counting

Counting dockerd Go calls in-kernel using BPF that match "*docker*get":

```
# funccount '/usr/bin/dockerd:*docker*get'
Tracing 463 functions for "'/usr/bin/dockerd:*docker*get"... Hit Ctrl-C to end.
^C
FUNC                                    COUNT
github.com/docker/docker/daemon.(*statsCollector).getSystemCPUUsage        3
github.com/docker/docker/daemon.(*Daemon).getNetworkSandboxID                3
github.com/docker/docker/daemon.(*Daemon).getNetworkStats                   3
github.com/docker/docker/daemon.(*statsCollector).getSystemCPUUsage.func1   3
github.com/docker/docker/pkg/ioutils.getBuffer                               6
github.com/docker/docker/vendor/golang.org/x/net/trace.getBucket            9
github.com/docker/docker/vendor/golang.org/x/net/trace.getFamily            9
github.com/docker/docker/vendor/google.golang.org/grpc.(*ClientConn).getTransport 10
github.com/docker/docker/vendor/github.com/golang/protobuf/proto.getbase    20
github.com/docker/docker/vendor/google.golang.org/grpc/transport.(*http2Client).getStream 30
Detaching...
```

```
# objdump -tTj .text /usr/bin/dockerd | wc -l
35859
```

35,859 functions can be traced!

Uses uprobes, and needs newer kernels. Warning: will cost overhead at high function rates.
BPF: dockerd Go Stack Tracing

Counting stack traces that led to this ioutils.getBuffer() call:

```
# stackcount 'p:/usr/bin/dockerd:/ioutils.getBuffer'
Tracing 1 functions for "p:/usr/bin/dockerd:/ioutils.getBuffer"... Hit Ctrl-C to end.
^C
 github.com/docker/docker/pkg/ioutils.getBuffer
 github.com/docker/docker/pkg/broadcaster.(*Unbuffered).Write
 bufio.(*Reader).writeBuf
 bufio.(*Reader).WriteTo
 io.copyBuffer
 io.Copy
 github.com/docker/docker/pkg/pools.Copy
 github.com/docker/docker/container/stream.(*Config).CopyToPipe.func1.1
 runtime.goexit
   dockerd [18176]
   110
Detaching...
```

means this stack was seen 110 times

Can also trace function arguments, and latency (with some work)

http://www.brendangregg.com/blog/2017-01-31/golang-bcc-bpf-function-tracing.html
Summary

Identify bottlenecks:

1. In the host vs container, using system metrics
2. In application code on containers, using CPU flame graphs
3. Deeper in the kernel, using tracing tools
References

- https://www.slideshare.net/aspyker/netflix-and-containers-titus
- https://docs.docker.com/engine/admin/runmetrics/#tips-for-high-performance-metric-collection
- https://blog.docker.com/2013/10/gathering-lxc-docker-containers-metrics/
- https://www.slideshare.net/jpetazzo/anatomy-of-a-container-namespaces-cgroups-some-filesystem-magic-linuxcon
- https://www.youtube.com/watch?v=sK5i-N34im8 Cgroups, namespaces, and beyond
- https://jvns.ca/blog/2016/10/10/what-even-is-a-container/
- https://blog.jessfraz.com/post/containers-zones-jails-vms/
- http://www.brendangregg.com/USEmethod/use-linux.html full USE method list
- http://queue.acm.org/detail.cfm?id=1809426 latency heat maps
Thank You!

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

Titus team: @aspyker @anwleung @fabiokung @tomaszbak1974 @amit_joshee @sargun @corindwyer ...

#dockercon