Linux Systems Performance

Brendan Gregg
Senior Performance Engineer

USENIX LISA 2019, Portland, Oct 28-30
Experience: A 3x Perf Difference
**mpstat**

load averages: serverA 90, serverB 17

---

**serverA# mpstat 10**
Linux 4.4.0-130-generic (serverA) 07/18/2019 _x86_64_ (48 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>10:07:55 PM</td>
<td>all</td>
<td>89.72</td>
<td>0.00</td>
<td>7.84</td>
<td>0.00</td>
<td>0.04</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.40</td>
</tr>
<tr>
<td>10:08:05 PM</td>
<td>all</td>
<td>88.60</td>
<td>0.00</td>
<td>9.18</td>
<td>0.00</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.17</td>
</tr>
<tr>
<td>10:08:15 PM</td>
<td>all</td>
<td>89.71</td>
<td>0.00</td>
<td>9.01</td>
<td>0.00</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.23</td>
</tr>
</tbody>
</table>

[...]

**Average:**
all 89.49 0.00 8.47 0.00 0.00 0.05 0.00 0.00 0.00 1.99

**serverB# mpstat 10**
Linux 4.19.26-nflx (serverB) 07/18/2019 _x86_64_ (64 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>09:56:11 PM</td>
<td>all</td>
<td>23.21</td>
<td>0.01</td>
<td>0.32</td>
<td>0.00</td>
<td>0.10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>76.37</td>
</tr>
<tr>
<td>09:56:21 PM</td>
<td>all</td>
<td>20.21</td>
<td>0.00</td>
<td>0.38</td>
<td>0.00</td>
<td>0.08</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>79.33</td>
</tr>
<tr>
<td>09:56:31 PM</td>
<td>all</td>
<td>21.58</td>
<td>0.00</td>
<td>0.39</td>
<td>0.00</td>
<td>0.10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>77.92</td>
</tr>
</tbody>
</table>

[...]

**Average:**
all 21.50 0.00 0.36 0.00 0.00 0.09 0.00 0.00 0.00 78.04
### serverA# ./pmcarch -p 4093 10

<table>
<thead>
<tr>
<th>K_CYCLES</th>
<th>K_INSTR</th>
<th>IPC</th>
<th>BR RETIRED</th>
<th>BR MISPRED</th>
<th>BMR%</th>
<th>LLCREF</th>
<th>LLCMISS</th>
<th>LLC%</th>
</tr>
</thead>
<tbody>
<tr>
<td>982412660</td>
<td>575706336</td>
<td>0.59</td>
<td>126424862460</td>
<td>2416880487</td>
<td>1.91</td>
<td>15724006692</td>
<td>10872315070</td>
<td>30.86</td>
</tr>
<tr>
<td>999621309</td>
<td>555043627</td>
<td>0.56</td>
<td>120449284756</td>
<td>2317302514</td>
<td>1.92</td>
<td>15378257714</td>
<td>11121882510</td>
<td>27.68</td>
</tr>
<tr>
<td>991146940</td>
<td>558145849</td>
<td>0.56</td>
<td>126350181501</td>
<td>2530383860</td>
<td>2.00</td>
<td>15965082710</td>
<td>11464682655</td>
<td>28.19</td>
</tr>
<tr>
<td>996314688</td>
<td>562276830</td>
<td>0.56</td>
<td>122215605985</td>
<td>2348638980</td>
<td>1.92</td>
<td>15558268345</td>
<td>10835594199</td>
<td>30.35</td>
</tr>
<tr>
<td>979890037</td>
<td>560268707</td>
<td>0.57</td>
<td>125609807909</td>
<td>2386085660</td>
<td>1.90</td>
<td>15828820588</td>
<td>11038597030</td>
<td>30.26</td>
</tr>
</tbody>
</table>

^C

### serverB# ./pmcarch -p 1928219 10

<table>
<thead>
<tr>
<th>K_CYCLES</th>
<th>K_INSTR</th>
<th>IPC</th>
<th>BR RETIRED</th>
<th>BR MISPRED</th>
<th>BMR%</th>
<th>LLCREF</th>
<th>LLCMISS</th>
<th>LLC%</th>
</tr>
</thead>
<tbody>
<tr>
<td>147523816</td>
<td>222396364</td>
<td>1.51</td>
<td>46053921119</td>
<td>641813770</td>
<td>1.39</td>
<td>8880477235</td>
<td>968809014</td>
<td>89.09</td>
</tr>
<tr>
<td>156634810</td>
<td>229801807</td>
<td>1.47</td>
<td>48236123575</td>
<td>653064504</td>
<td>1.35</td>
<td>9186609260</td>
<td>1183858023</td>
<td>87.11</td>
</tr>
<tr>
<td>152783226</td>
<td>237001219</td>
<td>1.55</td>
<td>49344315621</td>
<td>692819230</td>
<td>1.40</td>
<td>9314992450</td>
<td>879494418</td>
<td>90.56</td>
</tr>
<tr>
<td>140787179</td>
<td>213570329</td>
<td>1.52</td>
<td>44518363978</td>
<td>631588112</td>
<td>1.42</td>
<td>8675999448</td>
<td>712318917</td>
<td>91.79</td>
</tr>
<tr>
<td>136822760</td>
<td>219706637</td>
<td>1.61</td>
<td>45129020910</td>
<td>651436401</td>
<td>1.44</td>
<td>8689831639</td>
<td>617678747</td>
<td>92.89</td>
</tr>
</tbody>
</table>
### Server A

```bash
class stat -e cs -a -I 1000
#           time             counts unit events
1.000411740          2,063,105      cs
2.000977435          2,065,354      cs
3.001537756          1,527,297      cs
4.002028407            515,509      cs
5.002538455          2,447,126      cs
```

### Server B

```bash
class stat -e cs -p 1928219 -I 1000
#           time             counts unit events
1.001931945              1,172      cs
2.002664012              1,370      cs
3.003441563              1,034      cs
4.004140394              1,207      cs
5.004947675              1,053      cs
```

[...]
serverA# /usr/share/bcc/tools/cpudist -p 4093 10 1
Tracing on-CPU time... 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>3618650</td>
<td>****************************************</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>2704935</td>
<td>*****************************************</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>421179</td>
<td>****</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>99416</td>
<td>*</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>16951</td>
<td></td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>6355</td>
<td></td>
</tr>
</tbody>
</table>

[...]

serverB# /usr/share/bcc/tools/cpudist -p 1928219 10 1
Tracing on-CPU time... Hit Ctrl-C to end.

<table>
<thead>
<tr>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 -&gt; 511</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>156</td>
<td>*</td>
</tr>
<tr>
<td>1024 -&gt; 2047</td>
<td>238</td>
<td>**</td>
</tr>
<tr>
<td>2048 -&gt; 4095</td>
<td>4511</td>
<td>*****************************************</td>
</tr>
<tr>
<td>4096 -&gt; 8191</td>
<td>277</td>
<td>**</td>
</tr>
<tr>
<td>8192 -&gt; 16383</td>
<td>286</td>
<td>**</td>
</tr>
<tr>
<td>16384 -&gt; 32767</td>
<td>77</td>
<td></td>
</tr>
</tbody>
</table>

[...]
Systems Performance in 45 mins

• This is slides + discussion
• For more detail and stand-alone texts:
Agenda

1. Observability
2. Methodologies
3. Benchmarking
4. Profiling
5. Tracing
6. Tuning
1. Observability
How do you measure these?
Linux Observability Tools
Why Learn Tools?

• Most analysis at Netflix is via GUIs
• Benefits of command-line tools:
  - Helps you understand GUIs: they show the same metrics
  - Often documented, unlike GUI metrics
  - Often have useful options not exposed in GUIs
• Installing essential tools (something like):

$ sudo apt-get install sysstat bcc-tools bpftrace linux-tools-common \
  linux-tools-$\{uname -r\} iproute2 msr-tools
$ git clone https://github.com/brendangregg/msr-cloud-tools

These are crisis tools and should be installed by default
In a performance meltdown you may be unable to install them
One way to print *load averages*:

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

- A measure of resource demand: CPUs + disks
  - Includes TASK_UNINTERRUPTIBLE state to show all demand types
  - You can use BPF & off-CPU flame graphs to explain this state:
    http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html
  - PSI in Linux 4.20 shows CPU, I/O, and memory loads

- Exponentially-damped moving averages
  - With time constants of 1, 5, and 15 minutes. See historic trend.

- Load > # of CPUs, may mean CPU saturation

  Don’t spend more than 5 seconds studying these
System and per-process interval summary:

%CPU is summed across all CPUs
Can miss short-lived processes (atop won’t)
• Pros: configurable. Cons: misleading colors.
• dstat is similar, and now dead (May 2019); see pcp-dstat
vmstat

- Virtual memory statistics and more:

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

- USAGE: vmstat [interval [count]]
- First output line has *some* summary since boot values
- High level CPU summary
  - “r” is runnable tasks
$ iostat -xz 1

Linux 5.0.21 (c099.xxxx)     06/24/19   _x86_64_   (32 CPU)

<table>
<thead>
<tr>
<th>Device</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>rrrqm/s</th>
<th>wrqm/s</th>
<th>%rrqm</th>
<th>%wrqm</th>
</tr>
</thead>
<tbody>
<tr>
<td>sda</td>
<td>0.01</td>
<td>0.00</td>
<td>0.16</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>nvme3n1</td>
<td>19528.04</td>
<td>20.39</td>
<td>293152.56</td>
<td>14758.05</td>
<td>0.00</td>
<td>4.72</td>
<td>0.00</td>
<td>18.81</td>
</tr>
<tr>
<td>nvme1n1</td>
<td>18513.51</td>
<td>17.83</td>
<td>286402.15</td>
<td>13089.56</td>
<td>0.00</td>
<td>4.05</td>
<td>0.00</td>
<td>18.52</td>
</tr>
<tr>
<td>nvme0n1</td>
<td>16560.88</td>
<td>19.70</td>
<td>258184.52</td>
<td>14218.55</td>
<td>0.00</td>
<td>4.78</td>
<td>0.00</td>
<td>19.51</td>
</tr>
</tbody>
</table>

Workload

Very useful set of stats

Resulting Performance
- **Main memory usage:**

  ```bash
  $ free -m
  Mem:  total    used    free    shared  buff/cache  available
  23850   18248    592    3776    5008        1432
  Swap:  total    used    free
  31699   2021   29678
  ```

- Recently added “available” column
  - `buff/cache`: block device I/O cache + virtual page cache
  - `available`: memory likely available to apps
  - `free`: completely unused memory
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, "wow much syscall\n", 17wow much syscall ) = 17 <0.000048>
```

• Translates syscall arguments
• Not all kernel requests (e.g., page faults)
• Currently has massive overhead (ptrace based)
  – Can slow the target by > 100x. Skews measured time (-ttt, -T).

• perf trace will replace it: uses a ring buffer & BPF
tcpdump

- Sniff network packets for post analysis:

```
$ tcpdump -i eth0 -w /tmp/out.tcpdump
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C7985 packets captured
8996 packets received by filter
1010 packets dropped by kernel
# tcpdump -nr /tmp/out.tcpdump | head
reading from file /tmp/out.tcpdump, link-type EN10MB (Ethernet)
20:41:05.038437 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 18...
20:41:05.038533 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 48...
20:41:05.038584 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 96...
[...]
```

- Study packet sequences with timestamps (us)
- CPU overhead optimized (socket ring buffers), but can still be significant. *Use BPF in-kernel summaries instead.*
nstat

- Replacement for netstat from iproute2
- Various network protocol statistics:
  - `-s` won’t reset counters, otherwise intervals can be examined
  - `-d` for daemon mode
- Linux keeps adding more counters

```
$ nstat -s
#kernel
IpInReceives  31109659  0.0
IpInDelivers  31109371  0.0
IpOutRequests 33209552  0.0
[...]
TcpActiveOpens 508924  0.0
TcpPassiveOpens 388584  0.0
TcpAttemptFails  933  0.0
TcpEstabResets  1545  0.0
TcpInSegs  31099176  0.0
TcpOutSegs  56254112  0.0
TcpRetransSegs  3762  0.0
TcpOutRsts  3183  0.0
[...]
```
slabtop

- Kernel slab allocator memory usage:

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

  OBJS ACTIVE USE OBJ SIZE SLABS OBJ/SLAB CACHE SIZE NAME
3565575 3565575 100%  0.10K  91425       39   365700K buffer_head
314916 314066  99%    0.19K  14996       21   59984K dentry
184192 183751  99%    0.06K   2878       64    59984K dentry
138618 138618 100%    0.94K   4077       34  130464K xfs_inode
138602 138602 100%    0.21K   3746       37  29968K xfs_ili
102116  99012  96%    0.55K   3647       28  58352K radix_tree_node
  97482  49093  50%    0.09K   2321       42  9284K kmalloc-96
  22695  20777  91%    0.05K   267       85  1068K shared_policy_node
  21312  21312 100%    0.86K   576       37  18432K ext4_inode_cache
  16288  14601  89%    0.25K   509       32  4072K kmalloc-256
[...]
```
# 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 mincore(2) syscall. Used for database perf analysis.
docker stats

- Soft limits (cgroups) by container:

<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>58dcf8aed9a7</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>
</tbody>
</table>

- Stats are in /sys/fs/cgroups
- CPU shares and bursting breaks monitoring assumptions
showboost

• Determine current CPU clock rate

```
# showboost
Base CPU MHz : 2500
Set CPU MHz : 2500
Turbo MHz(s) : 3100 3200 3300 3500
Turbo Ratios : 124% 128% 132% 140%
CPU 0 summary every 1 seconds...

 TIME       C0_MCYC      C0_ACYC        UTIL  RATIO    MHz
23:39:07   1618910294   89419923        64%     5%    138
23:39:08   1774059258   97132588        70%     5%    136
23:39:09   2476365498   130869241       99%     5%    132
```

• Uses MSRs. Can also use PMCs for this.
• Also see turbostat.

https://github.com/brendangregg/msr-cloud-tools
Also: Static Performance Tuning Tools
Where do you start...and stop?

Workload Observability

Static Configuration
2. Methodologies
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
Methodologies

• Linux Performance Analysis in 60 seconds
• The USE method
• Workload characterization
• Many others:
  – Resource analysis
  – Workload analysis
  – Drill-down analysis
  – CPU profile method
  – Off-CPU analysis
  – Static performance tuning
  – 5 whys
...
Linux Perf Analysis in 60s

1. uptime  
2. dmesg -T | tail  
3. vmstat 1  
4. mpstat -P ALL 1  
5. pidstat 1  
6. iostat -xz 1  
7. free -m  
8. sar -n DEV 1  
9. sar -n TCP,ETCP 1  
10. top  

load averages  
kernel errors  
overall stats by time  
CPU balance  
process usage  
disk I/O  
memory usage  
network I/O  
TCP stats  
check overview

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 and software (cgroups)
Workload Characterization

Analyze workload characteristics, not resulting performance

For example, CPUs:

1. **Who**: which PIDs, programs, users
2. **Why**: code paths, context
3. **What**: CPU instructions, cycles
4. **How**: changing over time
3. Benchmarking
\~100\% of benchmarks are wrong

The energy needed to refute benchmarks is orders of magnitude bigger than to run them (so, no one does)
Benchmarking

• An experimental analysis activity
  – Try observational analysis first; benchmarks can perturb

• Benchmarking is error prone:
  – Testing the wrong target
    • eg, FS cache I/O instead of disk I/O
  – Choosing the wrong target
    • eg, disk I/O instead of FS cache I/O
  – Invalid results
    • eg, bugs
  – Misleading results:
    • you benchmark A, but actually measure B, and conclude you measured C

caution: benchmarking
Benchmark Examples

• Micro benchmarks:
  – File system maximum cached read operations/sec
  – Network maximum throughput

• Macro (application) benchmarks:
  – Simulated application max request rate

• Bad benchmarks:
  – `gitpid()` in a tight loop
  – Context switch timing
If your product’s chances of winning a benchmark are 50/50, you’ll usually lose

Benchmark paradox

caution: despair

Solution: Active Benchmarking

• Root cause analysis while the benchmark runs
  – Use the earlier observability tools
  – Identify the limiter (or suspect) and include it with the results

• For any given benchmark, ask: why not 10x?

• This takes time, but uncovers most mistakes
4. Profiling
Profiling

Can you do this?

“As an experiment to investigate the performance of the resulting TCP/IP implementation ... the 11/750 is CPU saturated, but the 11/780 has about 30% idle time. The time spent in the system processing the data is spread out among handling for the Ethernet (20%), IP packet processing (10%), TCP processing (30%), checksumming (25%), and user system call handling (15%), with no single part of the handling dominating the time in the system.”

– Bill Joy, 1981, TCP-IP Digest, Vol 1 #6

perf: CPU profiling

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

```
# 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

[...78,000 lines truncated...]
```
Full "perf report" Output
... as a Flame Graph
Flame Graphs

• Visualizes a collection of stack traces
  – **x-axis**: alphabetical stack sort, to maximize merging
  – **y-axis**: stack depth
  – **color**: random (default), or a dimension

• Perl + SVG + JavaScript
  – https://github.com/brendangregg/FlameGraph
  – Takes input from many different profilers
  – Multiple d3 versions are being developed

• References:
  – http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
  – http://queue.acm.org/detail.cfm?id=2927301
  – "The Flame Graph" CACM, June 2016
Linux CPU Flame Graphs

Linux 2.6+, via perf:

```bash
git clone --depth 1 https://github.com/brendangregg/FlameGraph
cd FlameGraph
perf record -F 99 -a -g -- sleep 30
perf script --header > out.perf01
./stackcollapse-perf.pl < out.perf01 | ./flamegraph.pl > perf.svg
```

Linux 4.9+, via BPF:

```bash
git clone --depth 1 https://github.com/brendangregg/FlameGraph
git clone --depth 1 https://github.com/iovisor/bcc
./bcc/tools/profile.py -dF 99 30 | ./FlameGraph/flamegraph.pl > perf.svg
```

- Most efficient: no perf.data file, summarizes in-kernel
FlameScope

- Analyze variance, perturbations

Subsecond-offset heat map

Flame graph

https://github.com/Netflix/flamescope
perf: 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]

[…]

  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]

• Measure instructions-per-cycle (IPC) and CPU stall types

• PMCs only enabled for some cloud instance types

My front-ends, incl. pmcarch:
https://github.com/brendangregg/pmc-cloud-tools
5. Tracing
Linux Tracing Events

Tracing Stack

- **add-on tools:** trace-cmd, perf-tools, bcc, bpftrace
- **front-end tools:** perf
- **tracing frameworks:** Ftrace, perf_events, BPF
- **back-end instrumentation:** tracepoints, kprobes, uprobes

**BPF** enables a new class of **custom, efficient**, and **production safe** performance analysis tools
Ftrace: perf-tools funccount

• Built-in kernel tracing capabilities, added by Steven Rostedt and others since Linux 2.6.27

```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>[...]</td>
<td></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_fsDestructor</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>
```

• Also see trace-cmd
perf: Tracing Tracepoints

• perf was introduced earlier; it is also a powerful tracer

```
# perf stat -e block:block_rq_complete -a sleep 10
Performance counter stats for 'system wide':

  91      block:block_rq_complete
```

```
# perf record -e block:block_rq_complete -a sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.428 MB perf.data (~18687 samples) ]
# perf script

    run 30339 [000] 2083345.722857: block:block_rq_complete: 202,1 W () 12986336 + 8 [0]
    run 30339 [000] 2083345.723180: block:block_rq_complete: 202,1 W () 12986528 + 8 [0]
  swapper     0 [000] 2083345.723489: block:block_rq_complete: 202,1 W () 12986496 + 8 [0]
  swapper     0 [000] 2083346.745840: block:block_rq_complete: 202,1 WS () 1053128 + 8 [0]
  supervise 30342 [000] 2083346.746571: block:block_rq_complete: 202,1 WS () 1053128 + 8 [0]
[...]
```

http://www.brendangregg.com/perf.html
https://perf.wiki.kernel.org/index.php/Main_Page
BCC/BPF: ext4slower

- ext4 operations slower than the threshold:

```
# ./ext4slower 1
Tracing ext4 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>06:49:17</td>
<td>bash</td>
<td>3616</td>
<td>R</td>
<td>128</td>
<td>0</td>
<td>7.75</td>
<td>cksum</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>39552</td>
<td>0</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>96</td>
<td>0</td>
<td>5.36</td>
<td>2to3-2.7</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>96</td>
<td>0</td>
<td>14.94</td>
<td>2to3-3.4</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>10320</td>
<td>0</td>
<td>6.82</td>
<td>411toppm</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>65536</td>
<td>0</td>
<td>4.01</td>
<td>a2p</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>55400</td>
<td>0</td>
<td>8.77</td>
<td>ab</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>36792</td>
<td>0</td>
<td>16.34</td>
<td>aclocal-1.14</td>
</tr>
<tr>
<td>[...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Better indicator of application pain than disk I/O
- Measures & filters in-kernel for efficiency using BPF

https://github.com/iovisor/bcc
bpftrace: one-liners

• Block I/O (disk) events by type; by size & comm:

```bash
# bpftrace -e 't:block:block_rq_issue { @[args->rwbs] = count(); }'
Attaching 1 probe...
^C
@[WS]: 2
@[RM]: 12
@[RA]: 1609
@[R]: 86421

# bpftrace -e 't:block:block_rq_issue { @bytes[comm] = hist(args->bytes); }'
Attaching 1 probe...
^C
@bytes[dmcrypt_write]:
[4K, 8K)       68 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[8K, 16K)      35 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[16K, 32K)      4 |@@@                                  |
[32K, 64K)      1 |                              |
[64K, 128K)     2 |@  https://github.com/iovisor/bpftrace
[...]```
BPF Perf Tools (2019)

BCC & bpftrace repos contain many of these. The book has them all.
Off-CPU Analysis

• Explain all blocking events. High-overhead: needs BPF.
6. Tuning
Ubuntu Bionic Tuning: Late 2019 (1/2)

- **CPU**
  
  ```bash
  schedtool -B PID
  disable Ubuntu apport (crash reporter)
  upgrade to Bionic (scheduling improvements)
  ```

- **Virtual Memory**
  
  ```bash
  vm.swappiness = 0 # from 60
  ```

- **Memory**
  
  ```bash
  echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
  kernel.numa_balancing = 0
  ```

- **File System**
  
  ```bash
  vm.dirty_ratio = 80 # from 40
  vm.dirty_background_ratio = 5 # from 10
  vm.dirty_expire_centisecs = 12000 # from 3000
  mount -o defaults,noatime,discard,nobarrier ...
  ```

- **Storage I/O**
  
  ```bash
  /sys/block/*/queue/rq_affinity 1 # or 2
  /sys/block/*/queue/scheduler kyber
  /sys/block/*/queue/nr_requests 256
  /sys/block/*/queue/read_ahead_kb 128
  mdadm -chunk=64 ...
  ```
• Networking
  
  net.core.default_qdisc = fq
  net.core.netdev_max_backlog = 5000
  net.core.rmem_max = 16777216
  net.core.somaxconn = 1024
  net.core.wmem_max = 16777216
  net.ipv4.ip_local_port_range = 10240 65535
  net.ipv4.tcp_abort_on_overflow = 1
  # maybe
  net.ipv4.tcp_congestion_control = bbr
  net.ipv4.tcp_max_syn_backlog = 8192
  net.ipv4.tcp_rmem = 4096 12582912 16777216
  # or 8388608 ...
  net.ipv4.tcp_slow_start_after_idle = 0
  net.ipv4.tcp_syn_retries = 2
  net.ipv4.tcp_tw_reuse = 1
  net.ipv4.tcp_wmem = 4096 12582912 16777216
  # or 8388608 ...

• Hypervisor
  
  echo tsc > /sys/devices/…/current_clocksource
  Plus use AWS Nitro

• Other
  
  net.core.bpf_jit_enable = 1
  sysct1 -w kernel.perf_event_max_stack=1000
Takeaways

Systems Performance is:
  Observability, Methodologies, Benchmarking, Profiling, Tracing, Tuning

Print out for your office wall:

1. uptime
2. dmesg -T | tail
3. vmstat 1
4. mpstat -P ALL 1
5. pidstat 1
6. iostat -xz 1
7. free -m
8. sar -n DEV 1
9. sar -n TCP,ETCP 1
10. top
Links

Netflix Tech Blog on Linux:

Linux Performance:

Linux perf:
- https://perf.wiki.kernel.org/index.php/Main_Page
- http://www.brendangregg.com/perf.html

Linux ftrace:
- https://www.kernel.org/doc/Documentation/trace/ftrace.txt
- https://github.com/brendangregg/perf-tools

Linux BPF:
- https://github.com/iovisor/bcc
- https://github.com/iovisor/bpftrace

Methodologies:

Flame Graphs & FlameScope:
- http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
- http://queue.acm.org/detail.cfm?id=2927301
- https://github.com/Netflix/flamescope

MSRs and PMCs:
- https://github.com/brendangregg/msr-cloud-tools
- https://github.com/brendangregg/pmc-cloud-tools

BPF Performance Tools
Thanks

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

Look out for 2nd Ed.