Linux Systems Performance

Brendan Gregg
Senior Performance Engineer

Auckland, Perth, Singapore, Hong Kong
What I’m currently working on
Application Request Time

- On-CPU Time
- Off-CPU Time
Application Request Time

On-CPU Time

Off-CPU Time

CPU Flame Graph

Off-CPU Flame Graph
Off-CPU stacks often end abruptly in libc
More than one way to walk a stack

Frame pointers
Last branch record (LBR)
Branch trace store (BTS)
DWARF
ORC
Application exception handler
...

We're currently rolling out our own build of libc with frame pointers (-fno-omit-frame-pointer)
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?
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
top

- System and per-process interval summary:

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

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

[...]
```

- %CPU is summed across all CPUs
- Can miss short-lived processes (atop won’t)
- dstat is similar, and now dead (May 2019); see pcp-dstat
vmstat

- Virtual memory statistics and more:

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

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

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

```
$ iostat -xz 1
Linux 5.0.21 (c099.xxxx)   06/24/19   _x86_64_   (32 CPU)
[...]
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm  ...
svd               0.01    0.00      0.16      0.00     0.00     0.00   0.00   0.00  /...
vme3n1           19528.04   20.39 293152.56  14758.05     0.00     4.72   0.00  18.81 \
vme1n1           18513.51   17.83 286402.15  13089.56     0.00     4.05   0.00  18.52 /...
vme0n1           16560.88   19.70 258184.52  14218.55     0.00     4.78   0.00  19.51 \
[...]
```

Workload

Very useful set of stats

Resulting Performance
free

• Main memory usage:

$ free -m

<table>
<thead>
<tr>
<th></th>
<th>total</th>
<th>used</th>
<th>free</th>
<th>shared</th>
<th>buff/cache</th>
<th>available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem:</td>
<td>23850</td>
<td>18248</td>
<td>592</td>
<td>3776</td>
<td>5008</td>
<td>1432</td>
</tr>
<tr>
<td>Swap:</td>
<td>31699</td>
<td>2021</td>
<td>29678</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• 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
[T...]
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
[T...]
```
### Kernel slab allocator memory usage:

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

<table>
<thead>
<tr>
<th>OBJS</th>
<th>ACTIVE</th>
<th>USE</th>
<th>OBJ</th>
<th>SIZE</th>
<th>SLABS</th>
<th>OBJ/SLAB</th>
<th>CACHE</th>
<th>SIZE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>3565575</td>
<td>3565575</td>
<td>100</td>
<td>0.10K</td>
<td>91425</td>
<td>39</td>
<td>365700K</td>
<td>buffer_head</td>
<td></td>
<td></td>
</tr>
<tr>
<td>314916</td>
<td>314066</td>
<td>99</td>
<td>0.19K</td>
<td>14996</td>
<td>21</td>
<td>59984K</td>
<td>dentry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>184192</td>
<td>183751</td>
<td>99</td>
<td>0.06K</td>
<td>2878</td>
<td>64</td>
<td>11512K</td>
<td>kmalloc-64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>138618</td>
<td>138618</td>
<td>100</td>
<td>0.94K</td>
<td>4077</td>
<td>34</td>
<td>130464K</td>
<td>xfs_inode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>138602</td>
<td>138602</td>
<td>100</td>
<td>0.21K</td>
<td>3746</td>
<td>37</td>
<td>29968K</td>
<td>xfs_ili</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102116</td>
<td>99012</td>
<td>96</td>
<td>0.55K</td>
<td>3647</td>
<td>28</td>
<td>58352K</td>
<td>radix_tree_node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97482</td>
<td>49093</td>
<td>50</td>
<td>0.09K</td>
<td>2321</td>
<td>42</td>
<td>9284K</td>
<td>kmalloc-96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22695</td>
<td>20777</td>
<td>91</td>
<td>0.05K</td>
<td>267</td>
<td>85</td>
<td>1068K</td>
<td>shared_policy_node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21312</td>
<td>21312</td>
<td>100</td>
<td>0.86K</td>
<td>576</td>
<td>37</td>
<td>18432K</td>
<td>ext4_inode_cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16288</td>
<td>14601</td>
<td>89</td>
<td>0.25K</td>
<td>509</td>
<td>32</td>
<td>4072K</td>
<td>kmalloc-256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
[...]

pcstat

- Show page cache residency by file:

```bash
# ./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>58dcf8aead9a7</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>337292fb5564</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
^C
```

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

https://github.com/brendangregg/msr-cloud-tools
Measures instructions-per-cycle (IPC) and other metrics

https://github.com/brendangregg/pmc-cloud-tools
Various thermal info is available in MSRs

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
• Drunk Man Anti-Method
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)

Start with the questions, then find the tools
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
Other Methodologies

- Resource analysis
- Workload analysis
- Drill-down analysis
- Off-CPU analysis
- Static performance tuning
- Performance mantras
- Scientific method
- 5 whys
- ...

All methodologies summarized:
http://www.brendangregg.com/methodology.html
3. Benchmarking
Benchmarking

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

• My favorite tools:
  – fio, lmbench, sysperf, iperf, netperf

• Benchmarking is error prone
  – ~100% of benchmarks are wrong
  – You benchmark A, but actually measure B, and conclude you measured C

caution: benchmarking
Solution: Active Benchmarking

- Root cause analysis while the benchmark runs
- 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:

```bash
# perf record -F 99 -ag -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2.745 MB perf.data (~119930 samples) ]
# perf report -n --stdio
1.40%  162   java [kernel.kallsyms] [k] _raw_spin_lock

  |   -- _raw_spin_lock
  |       --63.21%-- try_to_wake_up
  |           |   --63.91%-- default_wake_function
  |           |       |   --56.11%-- __wake_up_common
  |           |       |       |   __wake_up_locked
  |           |       |       |       |   ep_poll_callback
  |           |       |       |       |   __wake_up_common
  |           |       |       |       |   __wake_up_sync_key
  |           |       |       |       |                   |   --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://queue.acm.org/detail.cfm?id=2927301](http://queue.acm.org/detail.cfm?id=2927301)
  – "The Flame Graph" CACM, June 2016
Linux CPU Flame Graphs

Linux 2.6+, via perf:

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
cd FlameGraph
perf record -F 49 -a -g -- sleep 30
perf script --header > out.perf01
./stackcollapse-perf.pl < out.perf01 | ./flamegraph.pl > perf.svg
```

Linux 4.9+, via BPF:

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
git clone --depth 1 https://github.com/iovisor/bcc
./bcc/tools/profile.py -dF 49 30 | ./FlameGraph/flamegraph.pl > perf.svg
```

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

These files can be read using FlameScope
Mixed-Mode Flame Graphs
FlameScope

- Analyze variance, perturbations

https://github.com/Netflix/flamescope

1. Click start
2. Click end
3. See flame graph for that range

Subsecond-offset heat map

Flame graph

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

- Performance Monitoring Counters (PMCs):

```bash
$ perf list | grep -i hardware
  cpu-cycles OR cycles [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend [Hardware event]
  instructions [Hardware event]
  ...
  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 CPU operations, cycles, including stall cycles
- PMCs only enabled for some cloud instance types

My front-ends, incl. pmcarch:
https://github.com/brendangreggPMC-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>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>
```

• Also see trace-cmd
perf: Tracing Tracepoints

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

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

   91 block:block_rq_complete
```

```bash
# 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) ]
```

```bash
# 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 () 1052984 + 144 [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
         TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms)   FILENAME
06:49:17 bash           3616   R 128     0           7.75    cksum
06:49:17 cksum          3616   R 39552   0           1.34    [...
06:49:17 cksum          3616   R 96      0           5.36    2to3-2.7
06:49:17 cksum          3616   R 96      0          14.94    2to3-3.4
06:49:17 cksum          3616   R 10320   0           6.82    411toppm
06:49:17 cksum          3616   R 65536   0           4.01    a2p
06:49:17 cksum          3616   R 55400   0           8.77    ab
06:49:17 cksum          3616   R 36792   0          16.34    aclocal-1.14
[...]
```

- 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
```

```bash
# 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: Sep 2020 (1/2)

- **CPU**
  ```
  schedtool -B PID
  disable Ubuntu apport (crash reporter)
  upgrade to Bionic (scheduling improvements)
  ```
- **Virtual Memory**
  ```
  vm.swappiness = 0       # from 60
  ```
- **Memory**
  ```
  echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
  kernel.numa_balancing = 0
  ```
- **File System**
  ```
  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**
  ```
  /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 # may update to 1000
  net.core.rmem_max = 16777216
  net.core.somaxconn = 1024 # may update to 4096
  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
  sysctl -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:
• http://www.brendangregg.com/linuxperf.html

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:
• http://www.brendangregg.com/ebpf.html
• https://github.com/iovisor/bcc
• https://github.com/iovisor/bpftrace

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

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
Thanks

- Q&A in #qa-brendan
- http://slideshare.net/brendangregg
- http://www.brendangregg.com
- bgregg@netflix.com
- @brendangregg