Linux 4.x Performance
Using BPF Superpowers

Brendan Gregg
Senior Performance Architect

NETFLIX
Ten years ago, I gave a talk here about DTrace tools...
Superpowers are coming to Linux

Solve performance issues that were previously impossible

For example, full off-CPU analysis…
Figure 6.1. Process State Transition Diagram
Ideal Thread States

A starting point for deeper analysis
Linux Thread States

Based on:
- TASK_RUNNING
- TASK_INTERRUPTIBLE
- TASK_UNINTERRUPTIBLE

Still a useful starting point
Linux On-CPU Analysis

• I'll start with on-CPU analysis:

  running
  (on-CPU)

  CPU Flame Graph

  • Split into user/kernel states using /proc, mpstat(1), ...

  • perf_events ("perf") to analyze further:
    – User & kernel stack sampling (as a CPU flame graph)
    – CPI
    – Should be easy, but…
Broken Stacks

Missing Java stacks
"[unknown]"
Java Mixed-Mode CPU Flame Graph

- Fixed!
  - Java -XX:+PreserveFramePointer
  - Java perf-map-agent
  - Linux perf_events

Java

JVM

Kernel

GC
Also, CPI Flame Graph

Cycles Per Instruction
- **red** == instruction heavy
- **blue** == cycle heavy
  (likely mem stalls)

Zoomed:
In Linux, the state isn't helpful, but the code path is.

Off-CPU analysis by measuring blocked time with stack traces.
Off-CPU Time Flame Graph

From http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html
Off-CPU Time (zoomed): tar(1)

Currently kernel stacks only; user stacks will add more context
Off-CPU Time: more states

Flame graph quantifies total time spent in states
CPU + Off-CPU == See Everything?
Off-CPU Time (zoomed): gzip(1)

Off-CPU doesn't always make sense: what is gzip blocked on?
gzip(1) is blocked on tar(1)!

```
tar cf - * | gzip > out.tar.gz
```

Can't we associate off-CPU with wakeup stacks?
Wakeup stacks are associated and merged in-kernel using BPF.

We couldn't do this before.
Haven't Solved Everything Yet...

- One wakeup stack is often not enough...
- Who woke the waker?
Chain Graphs
Merging multiple wakeup stacks in kernel using BPF

With enough stacks, all paths lead to metal
Solve Everything

CPU + off-CPU analysis can solve most issues

Flame graph (profiling) types:

1. CPU
2. CPI
3. Off-CPU time
4. Wakeup time
5. Off-wake time
6. Chain

BPF makes this all more practical

different off-CPU analysis views, with more context and increasing measurement cost
2. BPF

"One of the more interesting features in this cycle is the ability to attach eBPF programs (user-defined, sandboxed bytecode executed by the kernel) to kprobes. This allows user-defined instrumentation on a live kernel image that can never crash, hang or interfere with the kernel negatively."

– Ingo Molnár (Linux developer)

Source: https://lkml.org/lkml/2015/4/14/232
2. BPF

"crazy stuff"
– Alexei Starovoitov (eBPF lead)

Source: http://www.slideshare.net/AlexeiStarovoitov/bpf-inkernel-virtual-machine
BPF

- eBPF == enhanced Berkeley Packet Filter; now just BPF
- Integrated into Linux (in stages: 3.15, 3.19, 4.1, 4.5, …)
- Uses
  - virtual networking
  - tracing
  - "crazy stuff"
- Front-ends
  - samples/bpf (raw)
  - bcc: Python, C
  - Linux perf_events
BPF for Tracing

- Can do per-event output and in-kernel summary statistics (histograms, etc).

![Diagram showing the process of BPF for Tracing]
Old way: TCP Retransmits

- tcpdump of all send & receive, dump to FS, post-process
- Overheads adds up on 10GbE+
New way: BPF TCP Retransmits

- Just trace the retransmit functions
- Negligible overhead

**tcpretrans (bcc)**

1. Config BPF & kprobe
2. read, print

**Kernel**

- send
- receive
- tcp_retransmit_skb()
# ./tcpretrans

<table>
<thead>
<tr>
<th>TIME</th>
<th>PID</th>
<th>IP</th>
<th>LADDR:LPOR</th>
<th>T&gt;</th>
<th>RADDR:RPORT</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:55:05</td>
<td>0</td>
<td>4 10.153.223.157:22</td>
<td>R&gt; 69.53.245.40:34619</td>
<td>ESTABLISHED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01:55:05</td>
<td>0</td>
<td>4 10.153.223.157:22</td>
<td>R&gt; 69.53.245.40:34619</td>
<td>ESTABLISHED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

includes kernel state
Old: Off-CPU Time Stack Profiling

- `perf_events` tracing of sched events, post-process
- Despite buffering, usually high cost (>1M events/sec)

```
perf record
1. async read
2. dump

perf inject
1. read
2. rewrite

perf report/script
read, process, print

Kernel
buffer
scheduler
file system (or pipe)
disks
```

New: BPF Off-CPU Time Stacks

- Measure off-CPU time, add to map with key = stack, value = total time. Async read map.

```
offcpmtime (bcc)
1. Config BPF & kprobe
2. async read stacks
3. symbol translate
4. print
```

```
Kernel
```
```
scheduler
finish_task_switch()
```

```
BPF
```
```
maps
```

Diagram:
- offcpmtime (bcc) interacts with the kernel's scheduler.
- The BPF program operates in the kernel space and interacts with the maps for storing and retrieving data.
Stack Trace Hack

• For my offcputime tool, I wrote a BPF stack walker:

```c
static u64 get_frame(u64 *bp) {
    if (*bp) {
        // The following stack walker is x86_64 specific
        u64 ret = 0;
        if (bpf_probe_read(&ret, sizeof(ret), (void *)(*bp+8)))
            return 0;
        if (bpf_probe_read(bp, sizeof(*bp), (void *)*bp))
            *bp = 0;
        if (ret < __START_KERNEL_map)
            return 0;
        return ret;
    }
    return 0;
}
```
"Crazy Stuff"

• ... using unrolled loops & goto:

```c
bp = ctx->bp;
// unrolled loop, 10 (MAXDEPTH) frames deep:
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;
if (!key.ret[depth++] = get_frame(&bp)) goto out;

out:
```
BPF Stack Traces

- Proper BPF stack support just landed in net-next:

```
Date: Sat, 20 Feb 2016 00:25:05 -0500 (EST)
Subject: Re: [PATCH net-next 0/3] bpf_get_stackid() and stack_trace map
From: David Miller <>

From: Alexei Starovoitov <ast@fb.com>
Date: Wed, 17 Feb 2016 19:58:56 -0800

> This patch set introduces new map type to store stack traces and
> corresponding bpf_get_stackid() helper.

... Series applied, thanks Alexei.
```

- Allows more than just chain graphs
memleak

- Real-time memory growth and leak analysis:

```python
# ./memleak.py -o 10 60 1
Attaching to kmalloc and kfree, Ctrl+C to quit.
[01:27:34] Top 10 stacks with outstanding allocations:
  72 bytes in 1 allocations from stack
    alloc_fdtable [kernel] (ffffffff8121960f)
    expand_files [kernel] (ffffffff8121986b)
    sys_dup2 [kernel] (ffffffff8121a68d)
[...]
  2048 bytes in 1 allocations from stack
    alloc_fdtable [kernel] (ffffffff812195da)
    expand_files [kernel] (ffffffff8121986b)
    sys_dup2 [kernel] (ffffffff8121a68d) ]
```

- Uses my stack hack, but will switch to BPF stacks soon
- By Sasha Goldshtein. Another bcc tool.
3. bcc

- BPF Compiler Collection
  - [https://github.com/iovisor/bcc](https://github.com/iovisor/bcc)
- Python front-end, C instrumentation
- Currently beta – in development!
- Some example tracing tools…
execsnoop

• Trace new processes:

```
# ./execsnoop

<table>
<thead>
<tr>
<th>PCOMM</th>
<th>PID</th>
<th>RET</th>
<th>ARGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>15887</td>
<td>0</td>
<td>/usr/bin/man ls</td>
</tr>
<tr>
<td>preconv</td>
<td>15894</td>
<td>0</td>
<td>/usr/bin/preconv -e UTF-8</td>
</tr>
<tr>
<td>man</td>
<td>15896</td>
<td>0</td>
<td>/usr/bin/tbl</td>
</tr>
<tr>
<td>man</td>
<td>15897</td>
<td>0</td>
<td>/usr/bin/nroff -mandoc -rLL=169n -rLT=169n -Tutf8</td>
</tr>
<tr>
<td>man</td>
<td>15898</td>
<td>0</td>
<td>/usr/bin/pager -s</td>
</tr>
<tr>
<td>nroff</td>
<td>15900</td>
<td>0</td>
<td>/usr/bin/locale charmap</td>
</tr>
<tr>
<td>nroff</td>
<td>15901</td>
<td>0</td>
<td>/usr/bin/groff -mtty-char -Tutf8 -mandoc -rLL=169n ...</td>
</tr>
<tr>
<td>groff</td>
<td>15902</td>
<td>0</td>
<td>/usr/bin/troff -mtty-char -mandoc -rLL=169n -rLT=169 ...</td>
</tr>
<tr>
<td>groff</td>
<td>15903</td>
<td>0</td>
<td>/usr/bin/grotty</td>
</tr>
</tbody>
</table>
```
biolatency

- Block device (disk) I/O latency distribution:

```
# ./biolatency -mT 1 5
Tracing block device I/O... Hit Ctrl-C to end.

06:20:16

msecs : count  distribution
0 -> 1 : 36  |**************************************|
2 -> 3 : 1   |*                                     |
4 -> 7 : 3   |***                                   |
8 -> 15 : 17  |*****************                     |
16 -> 31 : 33  |**********************************    |
32 -> 63 : 7   |*******                               |
64 -> 127 : 6   |******                                |
```
ext4slower

- ext4 file system I/O, slower than a threshold:

```bash
# ./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>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>15008</td>
<td>0</td>
<td>19.31</td>
<td>acpi_listen</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>6123</td>
<td>0</td>
<td>17.23</td>
<td>add-apt-repository</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>6280</td>
<td>0</td>
<td>18.40</td>
<td>addpart</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>27696</td>
<td>0</td>
<td>2.16</td>
<td>addr2line</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>58080</td>
<td>0</td>
<td>10.11</td>
<td>ag</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>906</td>
<td>0</td>
<td>6.30</td>
<td>ec2-meta-data</td>
</tr>
</tbody>
</table>
```

[...]
bashreadline

- Trace bash interactive commands system-wide:

```bash
# ./bashreadline

<table>
<thead>
<tr>
<th>TIME</th>
<th>PID</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>05:28:25</td>
<td>21176</td>
<td>ls -l</td>
</tr>
<tr>
<td>05:28:28</td>
<td>21176</td>
<td>date</td>
</tr>
<tr>
<td>05:28:35</td>
<td>21176</td>
<td>echo hello world</td>
</tr>
<tr>
<td>05:28:43</td>
<td>21176</td>
<td>foo this command failed</td>
</tr>
<tr>
<td>05:28:45</td>
<td>21176</td>
<td>df -h</td>
</tr>
<tr>
<td>05:29:04</td>
<td>3059</td>
<td>echo another shell</td>
</tr>
<tr>
<td>05:29:13</td>
<td>21176</td>
<td>echo first shell again</td>
</tr>
</tbody>
</table>
```
gethostlatency

- Show latency for `getaddrinfo/gethostbyname` calls:

```bash
# ./gethostlatency
TIME      PID    COMM       LATms  HOST
06:10:24  28011  wget       90.00  www.iovisor.org
06:10:28  28127  wget       0.00   www.iovisor.org
06:10:41  28404  wget       9.00   www.netflix.com
06:10:48  28544  curl       35.00  www.netflix.com.au
06:11:10  29054  curl       31.00  www.plumgrid.com
06:11:16  29195  curl       3.00   www.facebook.com
06:11:25  29404  curl       72.00  foo
06:11:28  29475  curl       1.00   foo
```
• Trace custom events. Ad hoc analysis multitool:

```
# trace 'sys_read (arg3 > 20000) "read %d bytes", arg3'

<table>
<thead>
<tr>
<th>TIME</th>
<th>PID</th>
<th>COMM</th>
<th>FUNC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>05:18:23</td>
<td>4490</td>
<td>dd</td>
<td>sys_read</td>
<td>read 1048576 bytes</td>
</tr>
<tr>
<td>05:18:23</td>
<td>4490</td>
<td>dd</td>
<td>sys_read</td>
<td>read 1048576 bytes</td>
</tr>
<tr>
<td>05:18:23</td>
<td>4490</td>
<td>dd</td>
<td>sys_read</td>
<td>read 1048576 bytes</td>
</tr>
<tr>
<td>05:18:23</td>
<td>4490</td>
<td>dd</td>
<td>sys_read</td>
<td>read 1048576 bytes</td>
</tr>
<tr>
<td>^C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
4. Future Work

- All event sources
- Language improvements
- More tools: eg, TCP
- GUI support
Linux Event Sources

Dynamic Tracing

Tracepoints
- ext4:
  Operating System
- Applications
- System Libraries
- System Call Interface
- VFS
- Sockets
- TCP/UDP
- File Systems
- IP
- Volume Manager
- Ethernet
- Block Device Interface
- Device Drivers
- jbd2:
- block:
- scsi:
- skb:
- net:
- cs:
- migrations
- cpu-clock
- page-faults
- minor-faults
- major-faults
- syscalls:
- sched:
- task:
- signal:
- timer:
- workqueue:
- sock:
- CPU Interconnect

Software Events
- jbd2:
- block:
- scsi:
- skb:
- net:
- cs:
- migrations
- cpu-clock
- page-faults
- minor-faults
- major-faults

PMCs
- XXX: todo
cycles
instructions
branch-*
L1-*
LLC-*

uprobes
done

kprobes
done

CPU 1

Memory Bus

DRAM

mem-load
mem-store

http://www.brendangregg.com/perf.html 2015
BPF/bcc Language Improvements

```c
static int trace_event(struct pt_regs *ctx, struct sock *sk, int type)
{
    if (sk == NULL)
        return 0;
    u32 pid = bpf_get_current_pid_tgid();
    struct sock *skp = NULL;
    bpf_probe_read(&skp, sizeof(skp), &sk);
    // pull in details
    u16 family = 0, lport = 0, dport = 0;
    char state = 0;
    bpf_probe_read(&family, sizeof(family), &skp->__sk_common.skc_family);
    bpf_probe_read(&lport, sizeof(lport), &skp->__sk_common.skc_num);
    bpf_probe_read(&dport, sizeof(dport), &skp->__sk_common.skc_dport);
    bpf_probe_read(&state, sizeof(state), (void *)&skp->__sk_common.skc_state);
    if (family == AF_INET) {
        struct ipv4_data_t data4 = {.pid = pid, .ip = 4, .type = type};
        bpf_probe_read(&data4.saddr, sizeof(u32),
                       &skp->__sk_common.skc_rcv_saddr);
        bpf_probe_read(&data4.daddr, sizeof(u32),
                       &skp->__sk_common.skc_daddr);
        data4.lport = lport;
        data4.dport = dport;
        data4.state = state;
        ipv4_events.perf_submit(ctx, &data4, sizeof(data4));
    }
}
```
More Tools

- eg, netstat(8)...

```bash
$ netstat -s
Ip:
  7962754 total packets received
  8 with invalid addresses
  0 forwarded
  0 incoming packets discarded
  7962746 incoming packets delivered
  8019427 requests sent out
Tcp:
  17337 active connections openings
  39515 passive connection openings
  8953 failed connection attempts
  240214 connection resets received
  3 connections established
  7198375 segments received
  7504939 segments sent out
  62696 segments retransmitted
  10 bad segments received.
  1072 resets sent
  InCsumErrors: 5
Udp:
  759925 packets received
  3412 packets to unknown port received.
  0 packet receive errors
  784370 packets sent
TcpExt:
  858 invalid SYN cookies received
  8951 resets received for embryonic SYN_RECV sockets
  14 packets pruned from receive queue because of socket buffer overrun
  6177 TCP sockets finished time out in fast timer
  293 packets rejects in established connections because of timestamp
  733028 delayed acks sent
  89 delayed acks further delayed because of locked socket
Quick ack mode was activated 13214 times
  336520 packets directly queued to recvmsg prequeue.
  43964 packets directly received from backlog
  11406012 packets directly received from prequeue
  1039165 packets header predicted
  7066 packets header predicted and directly queued to user
  1428960 acknowledgments not containing data received
  1006791 predicted acknowledgments
  1 times recovered from packet loss due to fast retransmit
  5044 times recovered from packet loss due to SACK data
  2 bad SACKs received
  Detected reordering 4 times using SACK
  Detected reordering 11 times using time stamp
  13 congestion windows fully recovered
  11 congestion windows partially recovered using Hoo heuristic
TCPDSACKundo: 39
  2384 congestion windows recovered after partial ack
  228 timeouts after SACK recovery
  100 timeouts in loss state
  5018 fast retransmits
  39 forward retransmits
  783 retransmits in slow start
  32455 other TCP timeouts
TCPLoseProbes: 30233
TCPLoseProbeRecovery: 19070
  992 sack retransmits failed
  18 times receiver scheduled too late for direct processing
  705 packets collapsed in receive queue due to low socket buffer
  13658 DSACKs sent for old packets
  8 DSACKs sent for out of order packets
  13595 DSACKs received
  33 DSACKs for out of order packets received
  32 connections reset due to unexpected data
  108 connections reset due to early user close
  1608 connections aborted due to timeout
TCPDSACKDiscord: 4
TCPDSACKIgnoredOld: 1
TCPDSACKIgnoredNewOld: 8649
TCPCuriousRTOs: 445
TCPBackShiftFailback: 8588
TCPRecOoSe: 95854
TCPOffQueue: 24741
TCPFilterQueue: 8
TCPChallengeACK: 1441
TCPReTrans: 5
TCPCuriousRtxMsgQueues: 1
TCP учеты: 4823
IpExt:
  InOctets: 1561561375
  OutOctets: 1509416943
  InOctetsPkt: 8201572
  InECPKts: 2
  InECTPKts: 3844
  InCEPKts: 306
```
Better TCP Tools

- TCP retransmit by type and time
- Congestion algorithm metrics
- etc.
GUI Support

- eg, Netflix Vector: open source instance analyzer:
Summary

• BPF in Linux 4.x makes many new things possible
  – Stack-based thread state analysis (solve all issues!)
  – Real-time memory growth/leak detection
  – Better TCP metrics
  – etc...

• Get involved: see iovisor/bcc

• So far just a preview of things to come
Links

- iovisor bcc:
  - https://github.com/iovisor/bcc

- BPF Off-CPU, Wakeup, Off-Wake & Chain Graphs:
  - http://www.brendangregg.com/blog/2016-02-05/ebpf-chaingraph-prototype.html

- Linux Performance:

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

- Flame Graphs:
  - http://www.brendangregg.com/flamegraphs.html

- Netflix Tech Blog on Vector:

- Wordcloud: https://www.jasondavies.com/wordcloud/
• Questions?
• http://slideshare.net/brendangregg
• http://www.brendangregg.com
• bgregg@netflix.com
• @brendangregg

Thanks to Alexei Starovoitov (Facebook), Brenden Blanco (PLUMgrid), Daniel Borkmann (Cisco), Wang Nan (Huawei), Sasha Goldshtein (Sela), and other BPF and bcc contributors!