This article is not for beginners

BPF has evolved

Many docs were true in 2015 but not today

... not the best start

Newcomers keep re-posting old info as new

... out of date

This talk is Jun 2021
BPF is no longer an acronym.

BPF is a bytecode and execution environment.
How to get quick and easy BPF performance wins
Think like a sysadmin
Not like a programmer

```c
8    # define BPF program
9    prog = ""
10   int hello(void *ctx) {
11       bpf_trace_printk("Hello, World!
12         return 0;
13   }
14   ""
15
16   # load BPF program
17   b = BPF(text=prog)
18   b.attach_kprobe(event=b.get_syscall_fnname("clone"), fn_name=
```
Think like a **sysadmin**

Get it installed everywhere and use it.

```bash
# apt-get install bcc-tools
# PATH=$PATH:/usr/share/bcc/tools
# execsnoop
# opensnoop
# tcplife
# ext4slower
# biosnoop
[...]```
Think like a **sysadmin**

Get it installed everywhere and use it.

```
# apt-get install bcc-tools
# PATH=$PATH:/usr/share/bcc/tools
# execsnoop Anything periodic running? crontab?
# opensnoop Any misconfigurations? File not found?
# tcplife Any unexpected TCP sessions?
# ext4slower Any file system I/O slower than 10ms?
# biosnoop Any unusual disk access patterns? Outliers?
[...]```
# iostat -xz 1
Linux 4.15.0-1052-aws (cass-xxx) 12/04/2019 _x86_64_ (8 CPU)

<table>
<thead>
<tr>
<th>avg-cpu: %user</th>
<th>%nice</th>
<th>%system</th>
<th>%iowait</th>
<th>%steal</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.67</td>
<td>0.13</td>
<td>0.25</td>
<td>0.00</td>
<td>0.00</td>
<td>94.95</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>rrqm/s</th>
<th>wrqm/s</th>
<th>r/s</th>
<th>w/s</th>
<th>rkB/s</th>
<th>wkB/s</th>
<th>avgrq-sz</th>
<th>avgqu-sz</th>
<th>await</th>
<th>r-await</th>
<th>w-await</th>
<th>svctm</th>
<th>%util</th>
</tr>
</thead>
<tbody>
<tr>
<td>xvda</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>8.00</td>
<td>0.00</td>
<td>16.00</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>
</tr>
<tr>
<td>xvdb</td>
<td>0.00</td>
<td>0.00</td>
<td>32.00</td>
<td>0.00</td>
<td>152.00</td>
<td>0.00</td>
<td>9.50</td>
<td>0.16</td>
<td>5.00</td>
<td>5.00</td>
<td>0.00</td>
<td>5.00</td>
<td>16.00</td>
</tr>
<tr>
<td>xvdc</td>
<td>0.00</td>
<td>0.00</td>
<td>36.00</td>
<td>1.00</td>
<td>192.00</td>
<td>4.00</td>
<td>10.59</td>
<td>0.19</td>
<td>5.19</td>
<td>5.33</td>
<td>0.00</td>
<td>5.19</td>
<td>19.20</td>
</tr>
<tr>
<td>xvdd</td>
<td>4.00</td>
<td>0.00</td>
<td>37.00</td>
<td>0.00</td>
<td>181.00</td>
<td>0.00</td>
<td>9.78</td>
<td>0.18</td>
<td>4.97</td>
<td>4.97</td>
<td>0.00</td>
<td>4.54</td>
<td>16.80</td>
</tr>
<tr>
<td>xvde</td>
<td>0.00</td>
<td>0.00</td>
<td>27.00</td>
<td>0.00</td>
<td>156.00</td>
<td>0.00</td>
<td>11.56</td>
<td>0.18</td>
<td>6.67</td>
<td>6.67</td>
<td>0.00</td>
<td>6.67</td>
<td>18.00</td>
</tr>
<tr>
<td>xvdf</td>
<td>0.00</td>
<td>0.00</td>
<td>35.00</td>
<td>0.00</td>
<td>164.00</td>
<td>0.00</td>
<td>9.37</td>
<td>0.19</td>
<td>5.37</td>
<td>5.37</td>
<td>0.00</td>
<td>5.37</td>
<td>18.80</td>
</tr>
<tr>
<td>xvdg</td>
<td>0.00</td>
<td>0.00</td>
<td>25.00</td>
<td>1.00</td>
<td>136.00</td>
<td>4.00</td>
<td>10.77</td>
<td>0.14</td>
<td>5.23</td>
<td>5.44</td>
<td>0.00</td>
<td>5.23</td>
<td>13.60</td>
</tr>
<tr>
<td>md0</td>
<td>0.00</td>
<td>0.00</td>
<td>195.00</td>
<td>2.00</td>
<td>965.00</td>
<td>8.00</td>
<td>9.88</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>
</tr>
</tbody>
</table>

 [...]
### Case Study: BCC biosnoop, cont.

# /usr/share/bcc/tools/biosnoop

<table>
<thead>
<tr>
<th>TIME(s)</th>
<th>COMM</th>
<th>PID</th>
<th>DISK</th>
<th>T SECTOR</th>
<th>BYTES</th>
<th>LAT(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>perl</td>
<td>7755</td>
<td>xvdc</td>
<td>R 610822184</td>
<td>4096</td>
<td>8.57</td>
</tr>
<tr>
<td>0.000812</td>
<td>biosnoop</td>
<td>32196</td>
<td>xvda</td>
<td>R 269480</td>
<td>4096</td>
<td>0.43</td>
</tr>
<tr>
<td>0.006197</td>
<td>perl</td>
<td>3285</td>
<td>xvde</td>
<td>R 610737856</td>
<td>4096</td>
<td>6.10</td>
</tr>
<tr>
<td>0.006390</td>
<td>perl</td>
<td>23937</td>
<td>xvde</td>
<td>R 377704624</td>
<td>4096</td>
<td>0.10</td>
</tr>
<tr>
<td>0.015040</td>
<td>perl</td>
<td>7755</td>
<td>xvdb</td>
<td>R 732825200</td>
<td>4096</td>
<td>8.51</td>
</tr>
<tr>
<td>0.022842</td>
<td>perl</td>
<td>3285</td>
<td>xvdc</td>
<td>R 732953880</td>
<td>4096</td>
<td>7.72</td>
</tr>
<tr>
<td>0.023019</td>
<td>perl</td>
<td>23937</td>
<td>xvdb</td>
<td>R 732907064</td>
<td>4096</td>
<td>0.09</td>
</tr>
<tr>
<td>0.034443</td>
<td>perl</td>
<td>7755</td>
<td>xvdg</td>
<td>R 732998328</td>
<td>4096</td>
<td>11.28</td>
</tr>
<tr>
<td>0.039648</td>
<td>perl</td>
<td>23937</td>
<td>xvdg</td>
<td>R 733127392</td>
<td>4096</td>
<td>5.08</td>
</tr>
<tr>
<td>0.039863</td>
<td>perl</td>
<td>31913</td>
<td>xvdg</td>
<td>R 732868048</td>
<td>4096</td>
<td>0.10</td>
</tr>
<tr>
<td>0.049431</td>
<td>perl</td>
<td>3285</td>
<td>xvdg</td>
<td>R 732906896</td>
<td>4096</td>
<td>9.45</td>
</tr>
<tr>
<td>0.058521</td>
<td>perl</td>
<td>27565</td>
<td>xvdg</td>
<td>R 610744920</td>
<td>4096</td>
<td>8.99</td>
</tr>
<tr>
<td>0.070843</td>
<td>perl</td>
<td>27565</td>
<td>xvdg</td>
<td>R 377706520</td>
<td>4096</td>
<td>12.26</td>
</tr>
<tr>
<td>0.080564</td>
<td>perl</td>
<td>31913</td>
<td>xvdg</td>
<td>R 610951744</td>
<td>4096</td>
<td>9.62</td>
</tr>
<tr>
<td>0.080804</td>
<td>perl</td>
<td>7755</td>
<td>xvdc</td>
<td>R 732858664</td>
<td>4096</td>
<td>0.14</td>
</tr>
<tr>
<td>0.086932</td>
<td>perl</td>
<td>27565</td>
<td>xvdg</td>
<td>R 732937416</td>
<td>4096</td>
<td>6.01</td>
</tr>
<tr>
<td>0.087093</td>
<td>perl</td>
<td>27565</td>
<td>xvde</td>
<td>R 610853240</td>
<td>4096</td>
<td>0.09</td>
</tr>
</tbody>
</table>
## Case Study: BCC biosnoop, cont.

<table>
<thead>
<tr>
<th>#</th>
<th>User</th>
<th>PID</th>
<th>PPID</th>
<th>User</th>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>3285</td>
<td>3274</td>
<td>1</td>
<td>14:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>7755</td>
<td>7748</td>
<td>1</td>
<td>04:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>11366</td>
<td>11359</td>
<td>1</td>
<td>10:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>15054</td>
<td>15049</td>
<td>2</td>
<td>16:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>19675</td>
<td>19670</td>
<td>1</td>
<td>06:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>23937</td>
<td>23930</td>
<td>1</td>
<td>12:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>27565</td>
<td>27561</td>
<td>2</td>
<td>18:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>28232</td>
<td>28223</td>
<td>1</td>
<td>02:16</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
<tr>
<td>root</td>
<td>31913</td>
<td>31907</td>
<td>1</td>
<td>08:15</td>
<td>?</td>
<td>/usr/bin/perl /apps/...ec2rotatelogs.pl</td>
</tr>
</tbody>
</table>

[...]

---

**Note:** The commands shown are examples and may not reflect the exact output of the `ps -ef | grep perl` command.
Many more tools to try!

bcc tools
bpftool trace tools
(from my book, open source)

Solve >90% of perf issues with canned observability (tracing) tools

This is BPF observability in one pic
Print on your office/home wall, use as a checklist

Suspected disk issue?
Try these first:
Print on your office/home wall, use as a checklist

Suspected disk issue? Try these first:

Then walk up the stack: file system tools, VFS tools, syscall tools, etc.
The future of BPF perf observability ... is **GUIs**. The end user may not even know it’s BPF.

This GUI is in development by Susie Xia, Netflix
Example real-time BPF observability UI

Netflix Vector (now retired) uses this model
- Instance agent pcp pmdabcc
- https://github.com/Netflix/vector
Example real-time BPF observability UI #2

Client

GUI

Web Server

GUI
Webpage

bpftrace tools

Server

Instance agent

bpf-trace

Application

Server

Instance agent

bpf-trace

Application

Netflix FlameCommander UI
• (not yet open source)
Think like a sysadmin

Please try to use bcc/bpftrace tools as-is and fetch regular updates

Many tools are sandcastles, and require frequent rebuilding to match kernel changes
Fortunately many companies and engineers maintain these versions

- Facebook
- Netflix
- Isovalent
- (many more)
Sysadmins sometimes program

shell scripting
awk
sed one-liners

bpftrace tools
bpftrace one-liners
Think like a **programmer** if

You have a real-world problem that tools don’t solve
You’re a BPF-based startup
You’re debugging your own code*
You’re doing networking/security/etc.
You really want to learn BPF internals

* JIT-ed runtimes like Java are currently complex to trace
Recommended tracing front-ends

I want to run some tools

- bcc, bpftrace

I want to hack up some new tools

- bpftrace

I want to spend weeks developing a BPF product

- bcc libbpf C, bcc Python (maybe), gobpf, libbbpf-rs

New, lightweight, CO-RE & BTF based
Requires LLVM; becoming obsolete / special-use only
Recommended tracing front-ends

I want to run some tools

- bcc, bpftrace

I want to hack up some new tools

- bpftrace

I want to spend weeks developing a BPF product

- bcc libbpf C, bcc Python (maybe), gobpf, libbbpf-rs

Unix analogies

- /usr/bin/*

New, lightweight, CO-RE & BTF based

Requires LLVM; becoming obsolete / special-use only

C, C++

bash, awk
# readahead.bt

Attaching 5 probes...

^C

Readahead unused pages: 128

Readahead used page age (ms):

@age_ms:

<table>
<thead>
<tr>
<th>Range</th>
<th>Age (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>2455</td>
</tr>
<tr>
<td>[2, 4)</td>
<td>8424</td>
</tr>
<tr>
<td>[4, 8)</td>
<td>4417</td>
</tr>
<tr>
<td>[8, 16)</td>
<td>7680</td>
</tr>
<tr>
<td>[16, 32)</td>
<td>4352</td>
</tr>
<tr>
<td>[32, 64)</td>
<td>0</td>
</tr>
<tr>
<td>[64, 128)</td>
<td>0</td>
</tr>
<tr>
<td>[128, 256)</td>
<td>384</td>
</tr>
</tbody>
</table>
#!/usr/local/bin/bpftrace

kprobe:__do_page_cache_readahead    { @in_readahead[tid] = 1; }
kretprobe:__do_page_cache_readahead { @in_readahead[tid] = 0; }
kretprobe:__page_cache_alloc       @in_readahead[tid]/
    { @birth[retval] = nsecs;
      @rapages++;
    }
kprobe:mark_page_accessed         @birth[arg0]/
    { @age_ms = hist((nsecs - @birth[arg0]) / 1000000);
      delete(@birth[arg0]);
      @rapages--;
    }

END
    { printf("\nReadahead unused pages: %d\n", @rapages); printf("\nReadahead used page age (ms):\n"); print(@age_ms); clear(@age_ms);
      clear(@birth); clear(@in_readahead); clear(@rapages); }

Fits on one slide!
BCC libbpf tool example

```bash
# ./opensnoop
PID    COMM              FD ERR PATH
27974  opensnoop         28   0 /etc/localtime
1482   redis-server      7   0 /proc/1482/stat
[...]

# ldd opensnoop
   linux-vdso.so.1 (0x00007ffdddf3f1000)
   libelf.so.1 => /usr/lib/x86_64-linux-gnu/libelf.so.1 (0x00007f9fb7836000)
   libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f9fb7619000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9fb7228000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f9fb7c76000)

# ls -lh opensnoop opensnoop.stripped
-rwxr-xr-x 1 root root 645K Feb 28 23:18 opensnoop
-rwxr-xr-x 1 root root 151K Feb 28 23:33 opensnoop.stripped

151 Kbytes for a stand-alone BPF program!
(Note: A static bpftrace/BTF + scripts will also have a small average tool size)
CONFIG_DEBUG_INFO_BTF=y

E.g., Ubuntu 20.10, Fedora 30, and RHEL 8.2 have it
BPF Future: Event-based Applications

- User-mode Applications
- Kernel-mode Applications (BPF)
- Scheduler
- Kernel Events
- Hardware Events (incl. clock)
# A New Type of Software

<table>
<thead>
<tr>
<th></th>
<th>Execution model</th>
<th>User defined</th>
<th>Compilation</th>
<th>Security</th>
<th>Failure mode</th>
<th>Resource access</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td>task</td>
<td>yes</td>
<td>any</td>
<td>user based</td>
<td>abort</td>
<td>syscall, fault</td>
</tr>
<tr>
<td><strong>Kernel</strong></td>
<td>task</td>
<td>no</td>
<td>static</td>
<td>none</td>
<td>panic</td>
<td>direct</td>
</tr>
<tr>
<td><strong>BPF</strong></td>
<td>event</td>
<td>yes</td>
<td>JIT, CO-RE</td>
<td>verified, JIT</td>
<td>error message</td>
<td>restricted helpers</td>
</tr>
</tbody>
</table>
Take Away

To get started with **BPF performance wins**, think like a **sysadmin**:

1. Install BCC & bpftrace tools
2. Run them
3. Get some wins
References

This is also where I recommend you go to learn more:

- https://github.com/iovisor/bcc/blob/master/docs/tutorial.md
- https://github.com/iovisor/bpftrace/blob/master/docs/tutorial_one_liners.md
- BPF Performance Tools, Addison Wesley 2020
- Systems Performance 2nd Edition, Addison Wesley 2021
- https://ebpf.io/what-is-ebpf
Thanks

**BPF**: Alexei Starovoitov (Facebook), Daniel Borkmann (Isovalent), David S. Miller (Red Hat), Jakub Kicinski (Facebook), Yonghong Song (Facebook), Martin KaFai Lau (Facebook), John Fastabend (Isovalent), Quentin Monnet (Isovalent), Jesper Dangaard Brouer (Red Hat), Andrey Ignatov (Facebook), and Stanislav Fomichev (Google), Linus Torvalds, and many more in the BPF community

**BCC**: Brenden Blanco (VMware), Yonghong Song, Sasha Goldsthein (Google), Teng Qin (Facebook), Paul Chaignon (Isovalent), Vicent Martí (PlanetScale), and many more in the BCC community

**bpftrace**: Alastair Robertson (Yellowbrick Data), Dan Xu (Facebook), Bas Smit, Mary Marchini (Netflix), Masanori Misono, Jiri Olsa, Viktor Malík, Dale Hamel, Willian Gaspar, Augusto Mecking Caringi, and many more in the bpftrace community