

bpftrace - A strong linux trace tool

Qiao Zhao

Agenda

- Background
- Bpftrace synopsis
- Demos - trace daily work

Bpfttrace background

- Bpfttrace (DTrace 2.0) for Linux 2018
 - Bpfttrace announce at Oct, 2018. Use bpfttrace feature since 4.9 kernel.
 - Language: similar to dtrace, awk, c(systemtap).
 - Using:
 - Trace
 - Performance
 - Troubleshooting

Bpftrace synopsis - probes

| | |
|------------|------------------------------|
| kprobe | Kernel function start |
| kretprobe | Kernel function return |
| uprobe | User-level function start |
| uretprobe | User-level function return |
| tracepoint | Kernel static tracepoints |
| usdt | User-level static tracepoint |
| profile | Timed sampling |
| interval | Timed output |
| software | Kernel software events |
| hardware | Processor-level events |

Bpftrace synopsis - built-in variables and functions

| Variable | Description |
|-------------|-----------------------------|
| pid | Process ID |
| comm | Process or command name |
| nsecs | Current time in nanoseconds |
| kstack | Kernel stack trace |
| ustack | User-level stack trace |
| arg0...argN | Function arguments |
| args | Tracepoint arguments |
| retval | Function return value |
| name | Full probe name |

| Function | Description |
|------------------------------|----------------------------|
| printf(...) | Print formatted string |
| time(...) | Print formatted time |
| system(...) | Run shell command |
| @ = count() | Count events |
| @ = hist(x) | Power-of-2 histogram for x |
| @ = lhist(x, min, max, step) | Linear histogram for x |

Bpftrace synopsis - one line command

```
# bpftrace -e 'BEGIN { printf("hello world\n"); }'
```

```
# bpftrace -e 'kprobe:do_nanosleep { printf("%s sleep by %d\n", comm, tid); }'
```

```
# bpftrace -e 'uprobe:/lib64/libc.so.6:fopen { printf("fopen: %s\n", str(arg0)); }'
```

```
# bpftrace -e 'uretprobe:/bin/bash:readline { printf("readline: \"%s\"\n", str(retval)); }'
```

```
# bpftrace -e 'tracepoint:sched:sched_switch { @[kstack] = count(); }'
```

```
# bpftrace -e 'kprobe:tcp_sendmsg { @size = hist(arg2); } interval:s:10 { exit(); }'
```

```
# bpftrace -e 'software:page-faults:100 { @[comm] = count(); }'
```

More: <https://github.com/iovisor/bpftrace/blob/master/docs/>

Bpfttrace synopsis - provided tools

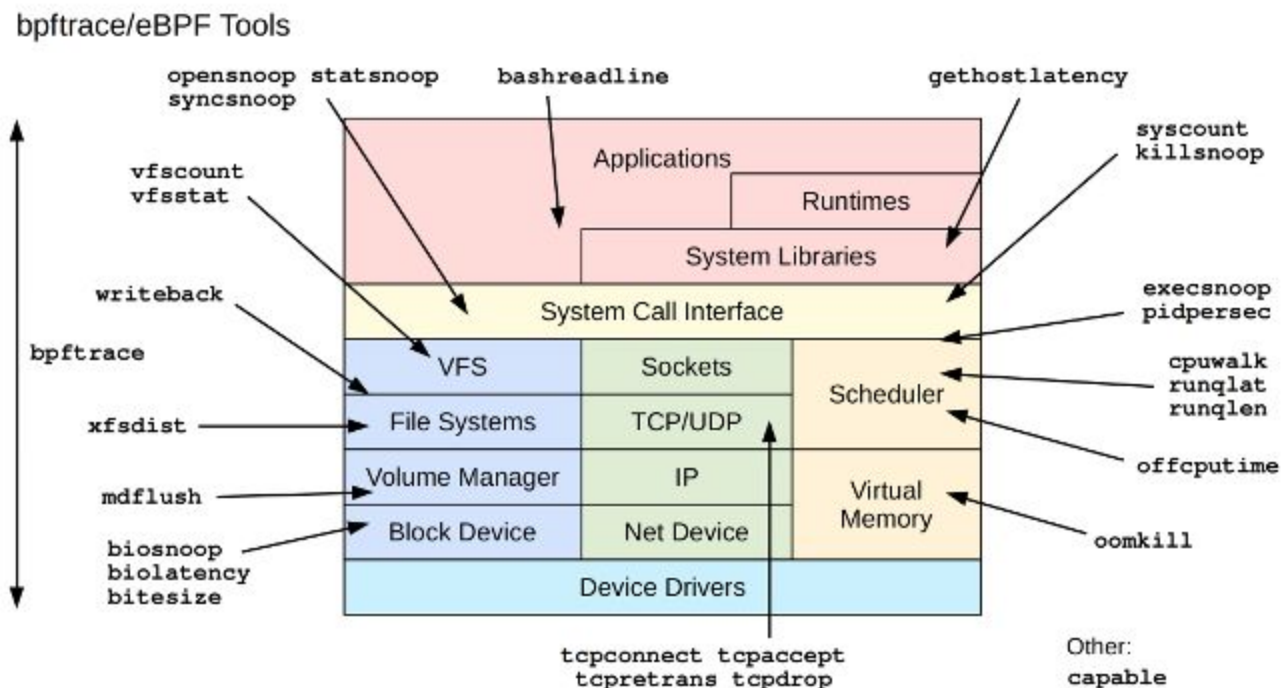


Diagram by Brendan Gregg, early 2019. <https://github.com/iovisor/bpfttrace>

Basic demos - k/uprobe; k/uretprobe

```
# bpftrace -e 'kprobe:_do_fork { @[comm] = count(); }'
```

Attaching 1 probe...

@[ThreadPoolForeg]: 1

@[kthreadd]: 1

@[chrome]: 1

```
# bpftrace -e 'uretprobe:/bin/bash:readline { printf("readline: \"%s\\n\"", str(retval)); }'
```

Attaching 1 probe...

readline: "cd Git/linux/"

readline: "cat /proc/cmdline"

...

Basic demos - count syscall

```
# bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @[comm] = count() }'
```

Attaching 1 probe...

^C

@[gssproxy]: 1

@[sssd]: 1

@[sedispatch]: 1

@[CacheThread_Blo]: 3

@[Chrome_SyncThre]: 4

@[sudo]: 5

@[Chrome_HistoryT]: 9

Basic demos - count syscall

```
# cat syscount.bt
```

```
#!/usr/bin/env bpftrace
BEGIN
{
    printf("Counting syscalls... Hit Ctrl-C to end.\n");
}
tracepoint:raw_syscalls:sys_enter
{
    @syscall[args->id] = count();
    @process[comm] = count();
}
END
{
    printf("\nTop 10 syscalls IDs:\n");
    print(@syscall, 10);
    clear(@syscall);
    printf("\nTop 10 processes:\n");
    print(@process, 10);
    clear(@process);
}
```

```
# bpftrace syscount.bt
```

Attaching 3 probes...

Counting syscalls... Hit Ctrl-C to end.

^C

Top 10 syscalls IDs:

@syscall[38]: 180

@syscall[20]: 289

...

@syscall[232]: 524

@syscall[7]: 648

@syscall[0]: 738

@syscall[16]: 1095

@syscall[47]: 1691

Top 10 processes:

@process[clock-applet]: 115

@process[gdbus]: 116

@process[Timer]: 125

...

@process[Web Content]: 491

@process[chrome]: 552

@process[Xorg]: 1863

@process[mate-multiloader]: 2019

Basic demos - kernel stack

```
# bpftrace -e 'k:tcp_sendmsg { @[kstack] = count(); }'
```

Attaching 1 probes...

```
@[
  tcp_sendmsg+1
  sock_sendmsg+65
  sock_write_iter+143
  new_sync_write+301
  vfs_write+182
  ksys_write+95
  do_syscall_64+95
  entry_SYSCALL_64_after_hwframe+68
]: 18
```

```
@[
  tcp_sendmsg+1
  sock_sendmsg+65
  __sys_sendto+238
  __x64_sys_sendto+37
  do_syscall_64+95
  entry_SYSCALL_64_after_hwframe+68
]: 59
```

Basic demos - tcp_sendmsg performance

```
# bpftrace -e 'k:tcp_sendmsg { @size = hist(arg2); } interval:s:10 { exit(); }'
```

Attaching 2 probes...

@size:

| | | |
|------------|---|--|
| [2, 4) | 4 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ |
| [4, 8) | 0 | |
| [8, 16) | 2 | @@@@@@@@@@@@@@@@ |
| [16, 32) | 2 | @@@@@@@@@@@@@@@@ |
| [32, 64) | 4 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ |
| [64, 128) | 3 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@ |
| [128, 256) | 5 | @@ |
| [256, 512) | 4 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ |
| [512, 1K) | 7 | @@ |
| [1K, 2K) | 1 | @@@@@@ |
| [2K, 4K) | 5 | @@ |

The Largest is 2k to 4k range, this is using a kprobe('k') on tcp_sendmsg(), and saving a histogram of arg2 (size) to a BPF map named "@size" (the name is unimportant). An interval event fires after 10 seconds and exits, at which point all BPF maps are printed. (Play the video in the background)

```
> int tcp_sendmsg(struct sock *sk, struct msghdr *msg, size_t size)
```

Basic demos - process debugging

```
# bpftrace -e 'kprobe:vfs_read /pid == 3412/ { @start[tid] = nsecs; } kretprobe:vfs_read /@start[tid]/ {  
@ns = hist(nsecs - @start[tid]); delete(@start[tid]); } interval:s:30 { exit(); }'
```

Attaching 3 probes...

@ns:

| | | | |
|------------|-----|--|--|
| [1K, 2K) | 4 | @@ | |
| [2K, 4K) | 23 | @@@@@@@@@@@@ | |
| [4K, 8K) | 48 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ | |
| [8K, 16K) | 104 | @@ | |
| [16K, 32K) | 3 | @ | |
| [32K, 64K) | 1 | | |

Process 3412 are firefox, bpftrace one-liners can easy to get the threads latency for vfs_read. Once you have performance issue, bpftrace can help. It can be of more use in helping to eliminate latency outliers.

Example of using - debug disk open/read

```
# bpftrace -e 'kprobe:vfs_read /pid == 19063/ { @start[tid] = nsecs; } kretprobe:vfs_read /@start[tid]/  
{ @ns = hist(nsecs - @start[tid]); delete(@start[tid]); } interval:s:30 { exit(); }'
```

Attaching 3 probes... [pid 19063 are command "dd if=/dev/random of=/tmp/t.img"]

@ns:

| | | |
|--------------|-----|--|
| [2K, 4K) | 2 | |
| [4K, 8K) | 18 | @ |
| [8K, 16K) | 20 | @ |
| [16K, 32K) | 543 | @@ |
| [32K, 64K) | 349 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@ |
| [64K, 128K) | 872 | @@ |
| [128K, 256K) | 195 | @@@@@@@@@@@@@@@@ |
| [256K, 512K) | 83 | @@@@ |
| [512K, 1M) | 10 | |
| [1M, 2M) | 2 | |
| [2M, 4M) | 2 | |
| [4M, 8M) | 0 | |
| [8M, 16M) | 0 | |
| [16M, 32M) | 0 | |
| [32M, 64M) | 2 | |
| [64M, 128M) | 2 | |
| [128M, 256M) | 5 | |
| [256M, 512M) | 91 | @@@@ |

@start[19063]: 34841858126989

Basic demos - include source file like c language

Use “.bt” file: (use kernel function)

```
# include <linux/path.h>
# include <linux/dcache.h>
kprobe:vfs_open
{
    printf("open path: %s\n", str(((path *)arg0)->dentry->d_name.name));
}
```

bpftrace -v path.bt

Attaching kprobe:vfs_open

Running...

open path: interrupts

open path: stat

open path: smp_affinity

C struct navigation (similar with systemtap)

Reference

<https://lwn.net/Articles/767956/>

<https://github.com/iovisor/bpftrace>

<https://github.com/iovisor/bpftrace/tree/master/tools>

<https://tracingsummit.org/wiki/TracingSummit2018#Schedule>

Thanks

Qiao Zhao
qiaozqjhsy@gmail.com
Red Hat