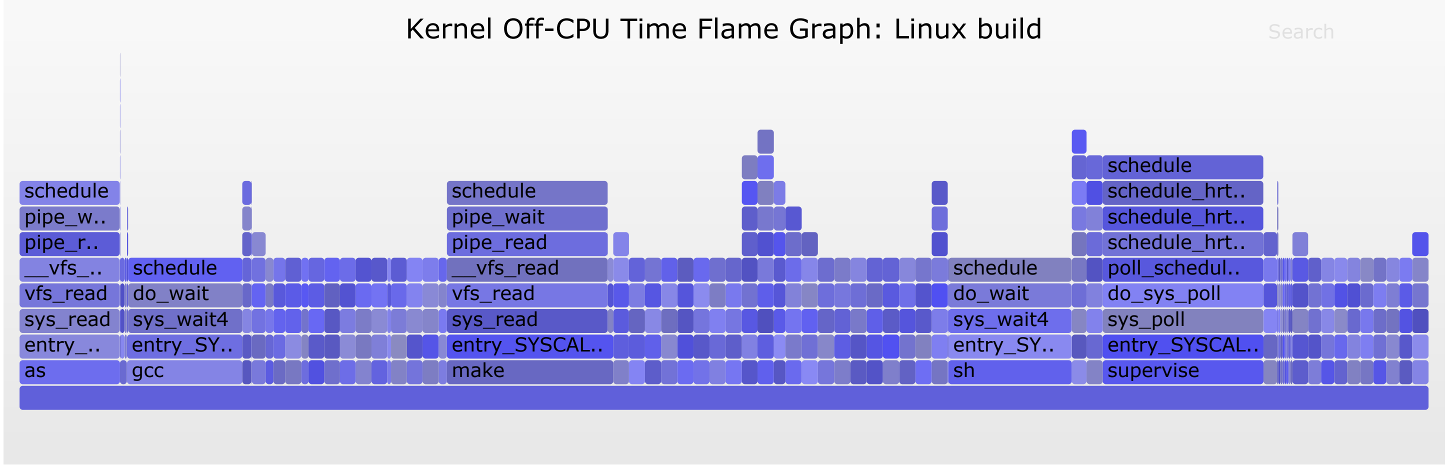
Linux eBPF Off-CPU Flame Graph

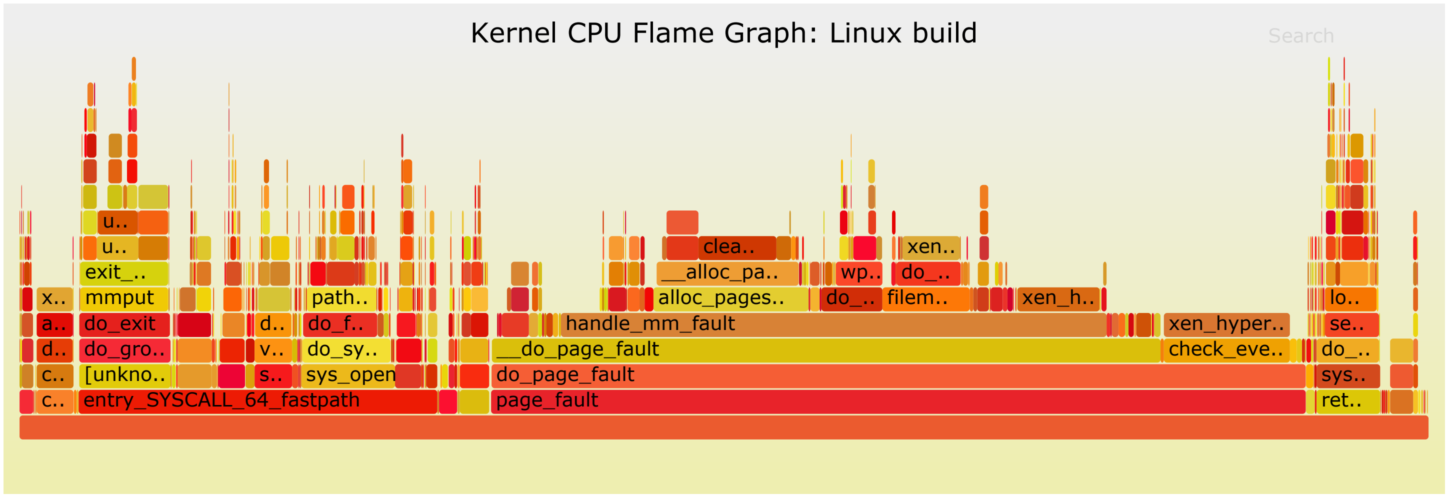
20 Jan 2016

CPU profiles, which can be visualized as a CPU flame graph, can solve a wide range of CPU usage issues. Off-CPU analysis, which can be visualized as Off-CPU time flame graphs, can begin to solve the rest. As I recently hacked stack traces for eBPF, I can now start to explore off-CPU profiling using core Linux capabilities.

Off-CPU Time Flame Graph

This is for a Linux kernel build, followed by a CPU profile flame graph:





Click to zoom, or browse the Off-CPU and CPU flame graph SVGs directly. To keep these examples simple, I'm only showing kernel time.

Seeing both on-CPU and off-CPU flame graphs shows you the full picture: what's consuming CPUs, what's blocked, and by how much. The x-axis for the CPU flame graph shows the sample population. The x-axis for the off-CPU time flame graph shows the time threads were blocked.

This example off-CPU time flame graph has the thread names at the bottom, then the stack traces. During the Linux kernel compile, we can see "as" and "make" were blocked in vfs_read(), likely reading from the filesystem. "gcc" and "sh" were waiting on child processes, as we'd expect. Plus there are many other threads waiting for work in poll routines. Since many threads can be blocked in parallel, they can add up to hundreds of seconds of blocked time for this 30 second profile.

Why eBPF Is So Important

Off-CPU profiling has been a prickly problem that can be explained with some math:

- A typical CPU profile using a system profiler may involve 99 stack trace samples per second, on all CPUs. If this were a 16 CPU instance, that's 1,584 samples per second. Linux perf_events writes every sample to a perf.data file (in groups for efficiency), and usually handles this rate with low overhead.
- An off-CPU profile may involve tracing the scheduler task switch routine, and recording timestamps, calculating off-CPU time, and saving this with stack traces. Scheduler events scale with load, so instead of maybe 1,584 stack samples per second, this can be over 100,000 or 1 million per second. Overhead can become a big problem fast, which can make this prohibitive for production use.

This is where the new eBPF functionality comes in: my offcputime script uses eBPF to sum off-CPU time by kernel stack trace in kernel context for efficiency. Only the summary is passed to user-level.

Dumping every event to a file (eg, perf_event's perf.data) will become problematic before it reaches 100,000 events per second, as it will likely cost noticeable CPU resources and start dropping events. By summarizing in kernel using eBPF, we can keep going, making this and a whole lot more practical. There's still overhead, and you should test and quantify before production use, but it should be the least possible.

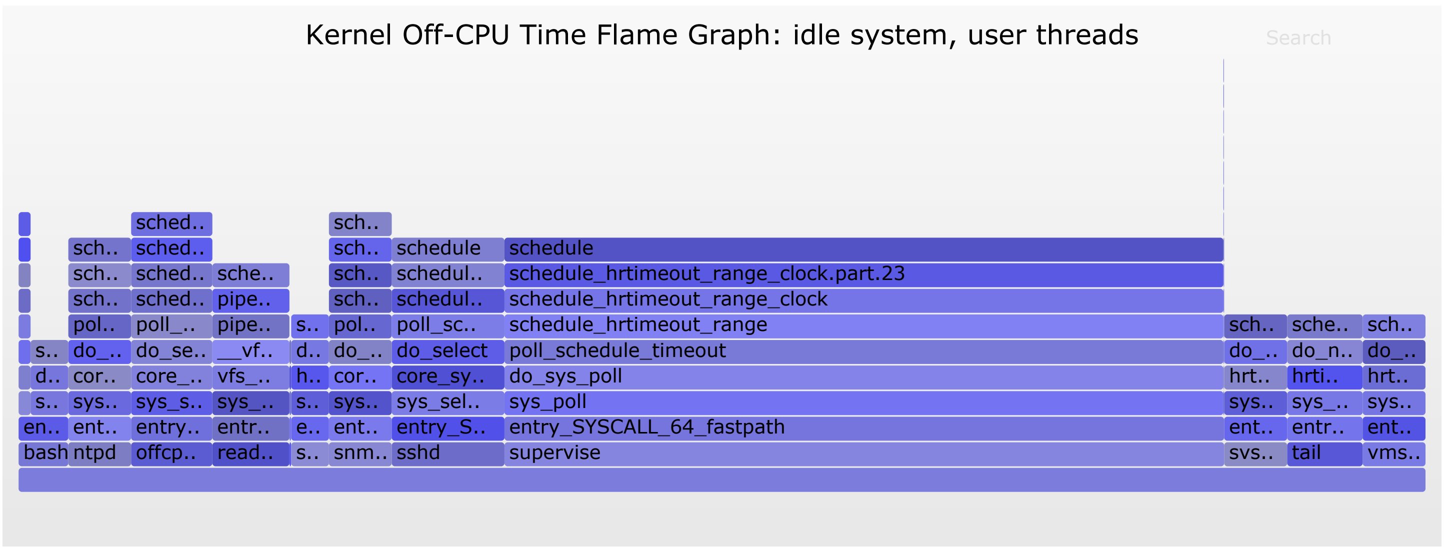
With my eBPF stack hack, I'm able to try out off-CPU flame graphs now, at least for kernel stacks, on x86_64, and up to 20 frames deep. In a future Linux version (4.5?), eBPF stack tracing should be supported properly, and not have these limitations.

offcputime

The off-CPU flame graph was generated using offcputime from bcc tools. It can emit folded output suitable for flame graph generation. Eg, for a 30 second profile:

```
# ./offcputime -f 30 > out.offcpustacks01
```

A -u option will print only user-mode stacks. For example, this off-CPU flame graph is for an idle system, where I ran "sleep 3" in one window, and "vmstat 1" in another SVG:



Can you find the sleep command? (Try the Search button in the top right.) The blocked time for "vmstat 1" can be seen on the right, which adds to 5 seconds (I stopped the profile between 5 and 6 seconds). Both sleep and vmstat are blocked on a timer, which makes sense.

The commands I used to create this were:

```
# ./offcputime -uf > out.offcpustacks02
^C
# ./flamegraph.pl --color=io --countname=us --width=900 \
    --title="Off-CPU Time Flame Graph: idle system" < out.offcpustacks02 > offcpu.svg
```

flamegraph.pl is from FlameGraph.

Other Tracers

I previously posted about using perf_events for off-CPU flame graphs, along with a warning about the prohibitive overhead, and a suggestion that this should be done using eBPF.

DTrace and SystemTap can also summarize in kernel, and can be used for off-CPU analysis. Yichun Zhang first published off-CPU time flame graphs in his Introduction to off-CPU Time Flame Graphs (PDF) talk, and has the SystemTap scripts in his nginx-systemtap-toolkit. Frits Hoogland more recently published stapflame as a proof of concept, which uses perf_events for CPU profiling and SystemTap for off-CPU time with Oracle context.

Future Work

eBPF needs stack trace support, for both kernel and user stacks, for generating complete off-CPU time flame graphs. Perhaps by the time you're reading this, it already exists (check the implementation of offcputime). Other eBPF features will help as well: once profiling (timed sampling) is available, eBPF could do in kernel aggregations of the CPU profile as well. It'll be great to have this in the core Linux kernel.

Update 2017: eBPF stack support arrived in Linux 4.8, and offcputime in bcc has been updated to use it.

More information about blocked stacks should also be included. Sometimes the off-CPU stack is a sufficient clue as to the source of latency, but in many cases it isn't: you're blocked on a file descriptor (sys_poll), a mutex, or a conditional variable. I'll explore this in an upcoming post.

Another area of future work is how to best show CPU and off-CPU flame graphs together. See my hot/cold flame graphs page for a summary of this work.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).