perf CPU Sampling

22 Jun 2014

top(1) shows a process is burning CPU – what do you do next? Depending on the application and context, you might use an application profiler to see why, or reads its logs, or even kill it. However, there's one answer that works for any application, even the kernel: use a *system profiler* like Linux `perf` (aka perf_events).

`perf` isn't some random tool: it's part of the Linux kernel, and is actively developed and enhanced. It is also powerful: it can instrument hardware counters, static tracepoints, and dynamic tracepoints.

In this post I'll restrain myself to one feature: CPU sampling. Lets pretend this is our target:

```
top - 04:38:41 up 29 days, 9 min,  2 users,  load average: 2.82, 3.26, 1.67
Tasks: 133 total,   9 running, 123 sleeping,   0 stopped,   1 zombie
Cpu(s): 28.0%us, 72.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7629464k total,  1481328k used,  6148136k free,   285412k buffers
Swap:        0k total,        0k used,        0k free,   566712k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6336 root      20   0  117m  976  488 R   25  0.0   0:01.35 fio
 6338 root      20   0  117m  972  484 R   25  0.0   0:01.35 fio
 6339 root      20   0  117m  976  488 R   25  0.0   0:01.35 fio
 6340 root      20   0  117m  976  488 R   25  0.0   0:01.33 fio
 6342 root      20   0  117m  976  488 R   25  0.0   0:01.33 fio
 6335 root      20   0  117m  972  484 R   25  0.0   0:01.32 fio
 6341 root      20   0  117m  972  484 R   25  0.0   0:01.33 fio
 6337 root      20   0  117m  960  472 R   24  0.0   0:01.31 fio
 2337 bgregg-t  20   0  149m 5608 4436 S    0  0.1   3:42.93 postgres
[...]
```

This top(1) output shows several `fio` processes eating 25% CPU each. Why? What are they doing?

1. Check perf is installed

With no arguments, it should print a help message:

```
# perf

 usage: perf [--version] [--help] COMMAND [ARGS]

 The most commonly used perf commands are:
[...]
```

If it's not there, you may find it can be added from the linux-tools-common package. It's also under tools/perf in the Linux kernel source.

2. Profile CPUs

```
# sudo perf record -F 99 -a -g -- sleep 20
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.560 MB perf.data (~24472 samples) ]
```

Options are:

- `-F 99`: sample at 99 Hertz (samples per second). I'll sometimes sample faster than this (up to 999 Hertz), but that also costs overhead. 99 Hertz should be negligible. Also, the value '99' and not '100' is to avoid lockstep sampling, which can produce skewed results.
- `-a`: samples on all CPUs. Without it, it only samples a supplied command or PID.
- `-g`: include stack traces.
- `--`: skips providing a -g argument (in newer `perf` versions, -g can pick the stack unwinding method).
- `sleep 20`: a dummy command, used to set the duration of our sampling.

As `perf` tells you, it writes a perf.data file.

3. Read profile

```
# sudo perf report -n --stdio
[...]
# Overhead  Samples     Command       Shared Object                                Symbol
# ........ ..........  ........  ..................  ..............................
#
    20.97%        208       fio  [kernel.kallsyms]   [k] hypercall_page
                  |
                  --- hypercall_page
                      check_events
                      |
                      |--63.94%-- 0x7fff695c398f
                      |
                      |--18.27%-- 0x7f0c5b72bd2d
                      |
                      --17.79%-- 0x7f0c5b72c46d

    14.21%        141       fio  [kernel.kallsyms]   [k] copy_user_generic_string
                  |
                  --- copy_user_generic_string
                      do_generic_file_read.constprop.33
                      generic_file_aio_read
                      do_sync_read
                      vfs_read
                      sys_read
                      system_call_fastpath
                      0x7f0c5b72bd2d

    10.79%        107       fio  [vdso]              [.] 0x7fff695c398f
                  |
                  --- 0x7fff695c398f
                      clock_gettime
[...]
```

You can just run `perf report` for the interactive text user-interface, and drill down stacks using the arrow keys. I find that mode laborious, and usually use this `--stdio` mode instead. `-n` prints sample counts.

The percentages show the breakdowns at each level. Multiply them to see the percentage for each leaf. For example, 0x7fff695c398f (whatever that is) was sampled 20.97% x 63.94% of the time (= %13.40). If you want `perf` to do the multiplications for you, and always show the absolute percentages, use `-g graph`.

The code `perf` is showing may be alien to you, but it shouldn't take long to learn at least the "hottest" (most frequently sampled) stacks. Often the function names are enough of a clue. `clock_gettime` is probably ... getting the time. Can `fio` avoid doing that, or do it differently, to eliminate this overhead? (yes, in this case.)

Common Issues

Hexidecimal numbers are printed if perf_events can't translate the symbols, which can happen with stripped binaries, or JIT'd code. For the former, look for dbgsym packages (debug symbols), or recompile and don't strip applications, and also make sure the kernel has CONFIG_KALLSYMS. For the latter, that's a bigger topic I'll write about another time (perf's JIT support).

Incomplete stacks usually mean -fomit-frame-pointer was used – a compiler optimization that makes little positive difference in the real world, but breaks stack profilers. Always compile with -fno-omit-frame-pointer. More recent `perf` has a `-g dwarf` option, to use the alternate libunwind/dwarf method for retrieving stacks.

While it can be a bit of work to get full stacks with symbols, a partially working profile can be enough to solve some problems. For example, I may not be able to see Java methods in the JVM, but I can see JVM system library usage, kernel CPU usage, and GC.

Flame Graphs

If the output of `perf report` is too long to read quickly, you can reprocess the perf.data file using `perf script` and visualize it using perf Flame Graphs. I use them all the time.

But wait, there's more!

I wanted to write a simple, short example of `perf`, that wasn't long and intimidating. There's a **lot** more to perf_events: see my perf examples page and the official perf wiki.

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).

About this blog

Perf Methods
USE Method
TSA Method
Off-CPU Analysis
Active Bench.
Flame Graphs
Heat Maps
Frequency Trails
Colony Graphs
perf Examples
eBPF Tools
DTrace Tools
DTraceToolkit
DtkshDemos
Guessing Game
Specials
Books
Other Sites