perf Hacktogram

10 Jul 2014

What is the distribution of sent packet sizes for my Linux system?

```
# ./perf-stat-hist net:net_dev_xmit len 10
Tracing net:net_dev_xmit, power-of-4, max 16384, for 10 seconds...

             Range           : Count    Distribution
             0               : 0        |                                        |
             1 -> 3          : 0        |                                        |
             4 -> 15         : 0        |                                        |
            16 -> 63         : 6        |#                                       |
            64 -> 255        : 385      |########################################|
           256 -> 1023       : 133      |##############                          |
          1024 -> 4095       : 155      |################                        |
          4096 -> 16383      : 0        |                                        |
         16384 ->            : 0        |                                        |
```

Great! So about half are between 64 and 255 bytes, and the rest are between 256 and 4095 bytes.

How about the requested size of read() syscalls?

```
# time ./perf-stat-hist syscalls:sys_enter_read count 10
Tracing syscalls:sys_enter_read, power-of-4, max 1048576, for 10 seconds...

             Range           : Count    Distribution
             0               : 0        |                                        |
             1 -> 3          : 1361     |#                                       |
             4 -> 15         : 2        |#                                       |
            16 -> 63         : 8        |#                                       |
            64 -> 255        : 60       |#                                       |
           256 -> 1023       : 1933859  |########################################|
          1024 -> 4095       : 59       |#                                       |
          4096 -> 16383      : 146      |#                                       |
         16384 -> 65535      : 21       |#                                       |
         65536 -> 262143     : 554007   |###########                             |
        262144 -> 1048575    : 0        |                                        |
       1048576 ->            : 0        |                                        |

real    0m10.056s
user    0m0.012s
sys     0m0.008s
```

Neat! The most common are in the 256 - 1023 byte range.

This time I added a `time` command, to show that extracting this information from the kernel cost little.

The script I'm using, perf-stat-hist, is demonstrating a custom distribution capability that is bread-and-butter for more advanced tracers like SystemTap, ktap, and DTrace. However, I'm not using those.

I'm using Linux perf_events on the 3.2 kernel. Aka, the `perf` command.

To do **in-kernel histograms**.

Stock, standard, perf_events.

Via user-space

Yes, for the current version of perf_events (3.16 and earlier) this is supposed to be impossible. perf_events can do in-kernel tracepoint counts, but anything beyond that requires dumping data to user-space for post-processing, like this:

```
# perf record -e 'syscalls:sys_enter_read' -a sleep 5
[ perf record: Woken up 25 times to write data ]
[ perf record: Captured and wrote 132.355 MB perf.data (~5782677 samples) ]
```

Now I have two problems. This perf.data file has over 5 million entries, which will cost some CPU to process. How much CPU just to read it? Lets dump it using `perf script` to /dev/null:

```
window1# time perf script > /dev/null
...hang...
window2# top
...hang...
```

Both windows have frozen. Now I have four problems.

When top finally runs, I can see what's wrong:

```
# top
top - 23:21:58 up 25 days,  2:56,  2 users,  load average: 1.68, 1.42, 1.09
Tasks: 142 total,   2 running, 136 sleeping,   0 stopped,   4 zombie
Cpu(s): 18.9%us, 54.1%sy,  0.0%ni, 27.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3839240k total,  3813020k used,    26220k free,      448k buffers
Swap:        0k total,        0k used,        0k free,   167712k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
27328 root      20   0 3437m 3.3g  68m R  100 88.8   0:06.02 perf
   34 root      20   0     0    0    0 S   26  0.0   0:45.04 kswapd0
26289 root      20   0     0    0    0 S   10  0.0   0:00.04 kworker/u4:0
26290 root      20   0 17344  712  340 R    5  0.0   0:00.90 top
[...]
```

`perf` has not only eaten one CPU, but also exhausted the memory on this system.

perf_events *does* have an excellent architecture for passing data from the kernel to user-level programs, minimizing overhead and CPU costs. What I'm testing is an extreme case. In many other cases, a `perf report` and `perf script` cycle will work fine, and the overhead will be negligible. A perf-stat-hist script using that cycle would merely bucketize the `perf script` output and print a report: a trivial program.

I could reduce overheads further by reading the binary perf.data file directly, or even better, calling perf_event_open() and mmap() to read the binary buffer, and process data without a trip via the file system. But there's also another way...

The hacktogram

This is based on `perf stat`, which does efficient in-kernel counts. I gave a quick tour of basic perf Counting capabilities in my previous post.

`perf stat` lets you instrument the same tracepoint multiple times, with different filters. The trick is to use a tracepoint and filter pair for each histogram bucket. For example:

```
# perf stat -e syscalls:sys_enter_read --filter 'count < 1024' \
    -e syscalls:sys_enter_read --filter 'count >= 1024 && count < 1048576' \
    -e syscalls:sys_enter_read --filter 'count > 1048576' -a sleep 5

 Performance counter stats for 'system wide':

         1,522,160      syscalls:sys_enter_read                     [100.00%]
           401,805      syscalls:sys_enter_read                     [100.00%]
                18      syscalls:sys_enter_read

       5.001822069 seconds time elapsed
```

This shows that there were 1,522,160 read() syscalls requesting less than 1 Kbyte, 401,805 requesting between 1 Kbyte and 1 Mbyte, and 18 requesting over 1 Mbyte.

That's the approach I used in perf-stat-hist. Tracing the same tracepoint multiple times *does* incur additional overhead, so this approach is not ideal, and can slow my target by up to 50% when using over a dozen tracepoints (buckets). It's a hack.

As for the variable I'm using, in this case "count": those come from the tracepoint. See the end of my previous post on perf Counting, and the contents of the /sys/.../format file.

Ideal

What would be ideal is for `perf stat` to provide a histogram option. Eg:

```
# perf stat -e syscalls:sys_enter_read --hist "pow2 count"
```

For a power-of-2 histogram of the count variable.

I think it's likely perf_events will get this capability in the future, especially thanks to recent kernel developments (more on this soon). So my perf-stat-hist workaround has a limited lifespan.

For more on perf_events, see my perf_events examples page and the perf_events 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

Colony Graphs
perf Examples
eBPF Tools
DTrace Tools
DTraceToolkit
DtkshDemos
Guessing Game
Specials
Books
Other Sites