perf sched for Linux CPU scheduler analysis

16 Mar 2017

Linux perf gained a new CPU scheduler analysis view in Linux 4.10: `perf sched timehist`. As I haven't talked about perf sched before, I'll summarize its capabilities here. If you're in a hurry, it may be helpful to just browse the following screenshots so that you are aware of what is available. (I've also added this content to my perf examples page.)

`perf sched` uses a dump-and-post-process approach for analyzing scheduler events, which can be a problem as these events can be very frequent – millions per second – costing CPU, memory, and disk overhead to record. I've recently been writing scheduler analysis tools using eBPF/bcc (including runqlat), which lets me greatly reduce overhead by using in-kernel summaries. But there are cases where you might want to capture every event using `perf sched` instead, despite the higher overhead. Imagine having five minutes to analyze a bad cloud instance before it is auto-terminated, and you want to capture everything for later analysis.

I'll start by recording one second of events:

```
# perf sched record -- sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.886 MB perf.data (13502 samples) ]
```

That's 1.9 Mbytes for *one second*, including 13,502 samples. The size and rate will be relative to your workload and number of CPUs (this example is an 8 CPU server running a software build). How this is written to the file system has been optimized: it only woke up one time to read the event buffers and write them to disk, which reduces overhead. That said, there are still significant overheads with instrumenting all scheduler events and writing event data to the file system. These events:

```
# perf script --header
# ========
# captured on: Sun Feb 26 19:40:00 2017
# hostname : bgregg-xenial
# os release : 4.10-virtual
# perf version : 4.10
# arch : x86_64
# nrcpus online : 8
# nrcpus avail : 8
# cpudesc : Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz
# cpuid : GenuineIntel,6,62,4
# total memory : 15401700 kB
# cmdline : /usr/bin/perf sched record -- sleep 1
# event : name = sched:sched_switch, , id = { 2752, 2753, 2754, 2755, 2756, 2757, 2758, 2759...
# event : name = sched:sched_stat_wait, , id = { 2760, 2761, 2762, 2763, 2764, 2765, 2766, 2...
# event : name = sched:sched_stat_sleep, , id = { 2768, 2769, 2770, 2771, 2772, 2773, 2774, ...
# event : name = sched:sched_stat_iowait, , id = { 2776, 2777, 2778, 2779, 2780, 2781, 2782,...
# event : name = sched:sched_stat_runtime, , id = { 2784, 2785, 2786, 2787, 2788, 2789, 2790...
# event : name = sched:sched_process_fork, , id = { 2792, 2793, 2794, 2795, 2796, 2797, 2798...
# event : name = sched:sched_wakeup, , id = { 2800, 2801, 2802, 2803, 2804, 2805, 2806, 2807...
# event : name = sched:sched_wakeup_new, , id = { 2808, 2809, 2810, 2811, 2812, 2813, 2814, ...
# event : name = sched:sched_migrate_task, , id = { 2816, 2817, 2818, 2819, 2820, 2821, 2822...
# HEADER_CPU_TOPOLOGY info available, use -I to display
# HEADER_NUMA_TOPOLOGY info available, use -I to display
# pmu mappings: breakpoint = 5, power = 7, software = 1, tracepoint = 2, msr = 6
# HEADER_CACHE info available, use -I to display
# missing features: HEADER_BRANCH_STACK HEADER_GROUP_DESC HEADER_AUXTRACE HEADER_STAT
# ========
#
    perf 16984 [005] 991962.879966:   sched:sched_wakeup: comm=perf pid=16999 prio=120 target_cpu
[...]
```

The captured trace file can be reported in a number of ways, summarized by the help message:

```
# perf sched -h

 Usage: perf sched [] {record|latency|map|replay|script|timehist}

    -D, --dump-raw-trace  dump raw trace in ASCII
    -f, --force           don't complain, do it
    -i, --input     input file name
    -v, --verbose         be more verbose (show symbol address, etc)
```

perf sched latency will summarize scheduler latencies by task, including average and maximum delay:

```
# perf sched latency

 -----------------------------------------------------------------------------------------------------------------
  Task                  |   Runtime ms  | Switches | Average delay ms | Maximum delay ms | Maximum delay at       |
 -----------------------------------------------------------------------------------------------------------------
  cat:(6)               |     12.002 ms |        6 | avg:   17.541 ms | max:   29.702 ms | max at: 991962.948070 s
  ar:17043              |      3.191 ms |        1 | avg:   13.638 ms | max:   13.638 ms | max at: 991963.048070 s
  rm:(10)               |     20.955 ms |       10 | avg:   11.212 ms | max:   19.598 ms | max at: 991963.404069 s
  objdump:(6)           |     35.870 ms |        8 | avg:   10.969 ms | max:   16.509 ms | max at: 991963.424443 s
  :17008:17008          |    462.213 ms |       50 | avg:   10.464 ms | max:   35.999 ms | max at: 991963.120069 s
  grep:(7)              |     21.655 ms |       11 | avg:    9.465 ms | max:   24.502 ms | max at: 991963.464082 s
  fixdep:(6)            |     81.066 ms |        8 | avg:    9.023 ms | max:   19.521 ms | max at: 991963.120068 s
  mv:(10)               |     30.249 ms |       14 | avg:    8.380 ms | max:   21.688 ms | max at: 991963.200073 s
  ld:(3)                |     14.353 ms |        6 | avg:    7.376 ms | max:   15.498 ms | max at: 991963.452070 s
  recordmcount:(7)      |     14.629 ms |        9 | avg:    7.155 ms | max:   18.964 ms | max at: 991963.292100 s
  svstat:17067          |      1.862 ms |        1 | avg:    6.142 ms | max:    6.142 ms | max at: 991963.280069 s
  cc1:(21)              |   6013.457 ms |     1138 | avg:    5.305 ms | max:   44.001 ms | max at: 991963.436070 s
  gcc:(18)              |     43.596 ms |       40 | avg:    3.905 ms | max:   26.994 ms | max at: 991963.380069 s
  ps:17073              |     27.158 ms |        4 | avg:    3.751 ms | max:    8.000 ms | max at: 991963.332070 s
...]
```

To shed some light as to how this is instrumented and calculated, I'll show the events that led to the top event's "Maximum delay at" of 29.702 ms. Here are the raw events from `perf sched script`:

```
      sh 17028 [001] 991962.918368:   sched:sched_wakeup_new: comm=sh pid=17030 prio=120 target_cpu=002
[...]
     cc1 16819 [002] 991962.948070:       sched:sched_switch: prev_comm=cc1 prev_pid=16819 prev_prio=120
                                                  prev_state=R ==> next_comm=sh next_pid=17030 next_prio=1
[...]
```

The time from the wakeup (991962.918368, which is in seconds) to the context switch (991962.948070) is 29.702 ms. This process is listed as "sh" (shell) in the raw events, but execs "cat" soon after, so is shown as "cat" in the `perf sched latency` output.

perf sched map shows all CPUs and context-switch events, with columns representing what each CPU was doing and when. It's the kind of data you see visualized in scheduler analysis GUIs (including `perf timechart`, with the layout rotated 90 degrees). Example output:

```
# perf sched map
                     *A0           991962.879971 secs A0 => perf:16999
                      A0     *B0   991962.880070 secs B0 => cc1:16863
          *C0         A0      B0   991962.880070 secs C0 => :17023:17023
  *D0      C0         A0      B0   991962.880078 secs D0 => ksoftirqd/0:6
   D0      C0 *E0     A0      B0   991962.880081 secs E0 => ksoftirqd/3:28
   D0      C0 *F0     A0      B0   991962.880093 secs F0 => :17022:17022
  *G0      C0  F0     A0      B0   991962.880108 secs G0 => :17016:17016
   G0      C0  F0    *H0      B0   991962.880256 secs H0 => migration/5:39
   G0      C0  F0    *I0      B0   991962.880276 secs I0 => perf:16984
   G0      C0  F0    *J0      B0   991962.880687 secs J0 => cc1:16996
   G0      C0 *K0     J0      B0   991962.881839 secs K0 => cc1:16945
   G0      C0  K0     J0 *L0   B0   991962.881841 secs L0 => :17020:17020
   G0      C0  K0     J0 *M0   B0   991962.882289 secs M0 => make:16637
   G0      C0  K0     J0 *N0   B0   991962.883102 secs N0 => make:16545
   G0     *O0  K0     J0  N0   B0   991962.883880 secs O0 => cc1:16819
   G0 *A0  O0  K0     J0  N0   B0   991962.884069 secs
   G0  A0  O0  K0 *P0  J0  N0   B0   991962.884076 secs P0 => rcu_sched:7
   G0  A0  O0  K0 *Q0  J0  N0   B0   991962.884084 secs Q0 => cc1:16831
   G0  A0  O0  K0  Q0  J0 *R0   B0   991962.884843 secs R0 => cc1:16825
   G0 *S0  O0  K0  Q0  J0  R0   B0   991962.885636 secs S0 => cc1:16900
   G0  S0  O0 *T0  Q0  J0  R0   B0   991962.886893 secs T0 => :17014:17014
   G0  S0  O0 *K0  Q0  J0  R0   B0   991962.886917 secs
[...]
```

This is an 8 CPU system, and you can see the 8 columns for each CPU starting from the left. Some CPU columns begin blank, as we've yet to trace an event on that CPU at the start of the profile. They quickly become populated.

The two character codes you see ("A0", "C0") are identifiers for tasks, which are mapped on the right ("=>"). This is more compact than using process (task) IDs. The "*" shows which CPU had the context switch event, and the new event that was running. For example, the very last line shows that at 991962.886917 (seconds) CPU 4 context-switched to K0 (a "cc1" process, PID 16945).

That example was from a busy system. Here's an idle system:

```
# perf sched map
                     *A0           993552.887633 secs A0 => perf:26596
  *.                  A0           993552.887781 secs .  => swapper:0
   .                 *B0           993552.887843 secs B0 => migration/5:39
   .                 *.            993552.887858 secs
   .                  .  *A0       993552.887861 secs
   .                 *C0  A0       993552.887903 secs C0 => bash:26622
   .                 *.   A0       993552.888020 secs
   .          *D0     .   A0       993552.888074 secs D0 => rcu_sched:7
   .          *.      .   A0       993552.888082 secs
   .           .     *C0  A0       993552.888143 secs
   .      *.   .      C0  A0       993552.888173 secs
   .       .   .     *B0  A0       993552.888439 secs
   .       .   .     *.   A0       993552.888454 secs
   .      *C0  .      .   A0       993552.888457 secs
   .       C0  .      .  *.        993552.889257 secs
   .      *.   .      .   .        993552.889764 secs
   .       .  *E0     .   .        993552.889767 secs E0 => bash:7902
...]
```

Idle CPUs are shown as ".".

Remember to examine the timestamp column to make sense of this visualization (GUIs use that as a dimension, which is easier to comprehend, but here the numbers are just listed). It's also only showing context switch events, and not scheduler latency. The newer `timehist` command has a visualization (-V) that can include wakeup events.

perf sched timehist was added in Linux 4.10, and shows the scheduler latency by event, including the time the task was waiting to be woken up (`wait time`) and the scheduler latency after wakeup to running (`sch delay`). It's the scheduler latency that we're more interested in tuning. Example output:

```
# perf sched timehist
Samples do not have callchains.
           time    cpu  task name                       wait time  sch delay   run time
                        [tid/pid]                          (msec)     (msec)     (msec)
--------------- ------  ------------------------------  ---------  ---------  ---------
  991962.879971 [0005]  perf[16984]                         0.000      0.000      0.000
  991962.880070 [0007]  :17008[17008]                       0.000      0.000      0.000
  991962.880070 [0002]  cc1[16880]                          0.000      0.000      0.000
  991962.880078 [0000]  cc1[16881]                          0.000      0.000      0.000
  991962.880081 [0003]  cc1[16945]                          0.000      0.000      0.000
  991962.880093 [0003]  ksoftirqd/3[28]                     0.000      0.007      0.012
  991962.880108 [0000]  ksoftirqd/0[6]                      0.000      0.007      0.030
  991962.880256 [0005]  perf[16999]                         0.000      0.005      0.285
  991962.880276 [0005]  migration/5[39]                     0.000      0.007      0.019
  991962.880687 [0005]  perf[16984]                         0.304      0.000      0.411
  991962.881839 [0003]  cat[17022]                          0.000      0.000      1.746
  991962.881841 [0006]  cc1[16825]                          0.000      0.000      0.000
[...]
  991963.885740 [0001]  :17008[17008]                      25.613      0.000      0.057
  991963.886009 [0001]  sleep[16999]                     1000.104      0.006      0.269
  991963.886018 [0005]  cc1[17083]                         19.998      0.000      9.948
```

This output includes the `sleep` command run to set the duration of perf itself to one second. Note that `sleep`'s wait time is 1000.104 milliseconds because I had run "sleep 1": that's the time it was asleep waiting its timer wakeup event. Its scheduler latency was only 0.006 milliseconds, and its time on-CPU was 0.269 milliseconds.

There are a number of options to timehist, including -V to add a CPU visualization column, -M to add migration events, and -w for wakeup events. For example:

```
# perf sched timehist -MVw
Samples do not have callchains.
           time    cpu  012345678  task name           wait time  sch delay   run time
                                   [tid/pid]             (msec)     (msec)     (msec)
--------------- ------  ---------  ------------------  ---------  ---------  ---------
  991962.879966 [0005]             perf[16984]                                          awakened: perf[16999]
  991962.879971 [0005]       s     perf[16984]             0.000      0.000      0.000
  991962.880070 [0007]         s   :17008[17008]           0.000      0.000      0.000
  991962.880070 [0002]    s        cc1[16880]              0.000      0.000      0.000
  991962.880071 [0000]             cc1[16881]                                           awakened: ksoftirqd/0[6]
  991962.880073 [0003]             cc1[16945]                                           awakened: ksoftirqd/3[28]
  991962.880078 [0000]  s          cc1[16881]              0.000      0.000      0.000
  991962.880081 [0003]     s       cc1[16945]              0.000      0.000      0.000
  991962.880093 [0003]     s       ksoftirqd/3[28]         0.000      0.007      0.012
  991962.880108 [0000]  s          ksoftirqd/0[6]          0.000      0.007      0.030
  991962.880249 [0005]             perf[16999]                                          awakened: migration/5[39]
  991962.880256 [0005]       s     perf[16999]             0.000      0.005      0.285
  991962.880264 [0005]        m      migration/5[39]                                     migrated: perf[16999] cpu
  991962.880276 [0005]       s     migration/5[39]         0.000      0.007      0.019
  991962.880682 [0005]        m      perf[16984]                                       migrated: cc1[16996] cpu 0
  991962.880687 [0005]       s     perf[16984]             0.304      0.000      0.411
  991962.881834 [0003]             cat[17022]                                           awakened: :17020
...]
  991963.885734 [0001]             :17008[17008]                                        awakened: sleep[16999]
  991963.885740 [0001]   s         :17008[17008]          25.613      0.000      0.057
  991963.886005 [0001]             sleep[16999]                                         awakened: perf[16984]
  991963.886009 [0001]   s         sleep[16999]         1000.104      0.006      0.269
  991963.886018 [0005]       s     cc1[17083]              19.998      0.000      9.948
```

The CPU visualization column ("012345678") has "s" for context-switch events, and "m" for migration events, showing the CPU of the event.

The last events in that output include those related to the "sleep 1" command used to time perf. The wakeup happened at 991963.885734, and at 991963.885740 (6 microseconds later) CPU 1 begins to context-switch to the sleep process. The column for that event still shows ":17008[17008]" for what was on-CPU, but the target of the context switch (sleep) is not shown. It is in the raw events:

```
  :17008 17008 [001] 991963.885740:       sched:sched_switch: prev_comm=cc1 prev_pid=17008 prev_prio=120
                                                              prev_state=R ==> next_comm=sleep next_pid=16999 next_pr
```

The 991963.886005 event shows that the perf command received a wakeup while sleep was running (almost certainly sleep waking up its parent process because it terminated), and then we have the context switch on 991963.886009 where sleep stops running, and a summary is printed out: 1000.104 ms waiting (the "sleep 1"), with 0.006 ms scheduler latency, and 0.269 ms of CPU runtime.

Here I've decorated the timehist output with the details of the context switch destination in red:

```
  991963.885734 [0001]             :17008[17008]                                      awakened: sleep[16999]
  991963.885740 [0001]   s         :17008[17008]          25.613      0.000      0.057  next: sleep[16999]
  991963.886005 [0001]             sleep[16999]                                       awakened: perf[16984]
  991963.886009 [0001]   s         sleep[16999]         1000.104      0.006      0.269  next: cc1[17008]
  991963.886018 [0005]       s     cc1[17083]             19.998      0.000      9.948  next: perf[16984]
```

When sleep finished, a waiting "cc1" process then executed. perf ran on the following context switch, and is the last event in the profile (perf terminated). I've submitted a patch to add this info when a -n option is used.

perf sched script dumps all events (similar to `perf script`):

```
# perf sched script
    perf 16984 [005] 991962.879960: sched:sched_stat_runtime: comm=perf pid=16984 runtime=3901506 [ns] vruntime=165.
    perf 16984 [005] 991962.879966:       sched:sched_wakeup: comm=perf pid=16999 prio=120 target_cpu=005
    perf 16984 [005] 991962.879971:       sched:sched_switch: prev_comm=perf prev_pid=16984 prev_prio=120 prev_stat.
    perf 16999 [005] 991962.880058: sched:sched_stat_runtime: comm=perf pid=16999 runtime=98309 [ns] vruntime=16405.
     cc1 16881 [000] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=16881 runtime=3999231 [ns] vruntime=7897.
  :17024 17024 [004] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=17024 runtime=3866637 [ns] vruntime=7810.
     cc1 16900 [001] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=16900 runtime=3006028 [ns] vruntime=7772.
     cc1 16825 [006] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=16825 runtime=3999423 [ns] vruntime=7876.
```

Each of these events ("sched:sched_stat_runtime" etc) are tracepoints you can instrument directly using perf record. As I've shown earlier, this raw output can be useful for digging further than the summary commands.

That's it for now. Happy hunting.

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).