Red Hat
Enterprise Linux

Rtla: an interface for osnoise/timerlat tracers

Daniel Bristot de Oliveira

Red Hat

State of art for testing/benchmark -rt kernel

- Nowadays, we use a set of **blackbox** tools that mimics a workload
  - Periodic like cyclictest
  - Polling like sysjitter/oslat

- If a bad value happens, you need to start getting the hands dirt
  - The developer needs to set up a trace session manually
    - Hard to do when you have someone else operating the machine
  - Manual interpretation of a lot of data
    - Speculation goes on (many times misleading)

- You never know if the problem you faced in the first place is the same you are seeing while tracing
  - That is especially hard when the target values are tight, and a lot of information is traced

- After 10+ years doing this, the trace became a mechanical thing:
  - Irq events, sched: events, compute deltas.

This topic is the continuation of a topic from last year, but progress was made!

osnoise and timerlat tracers

- osnoise and timerlat are kernel **tracers that also dispatches the workload**
  - The workload runs in the kernel:
    - osnoise: A busy loop kernel thread that reads time() in a loop
      - Reports problem when time()' - time() > threshold - aka noise.
    - timerlat: A periodic task that is awakened by an hrtimer
      - Reports **IRQ latency** and **Thread latency**

- The tracers provided a **new set of tracepoints** that automatize the trace:
  - osnoise:nmi_noise/irq_noise/softirq_noise/thread_noise:
    - Report the interference of these tasks to the tracer workload
    - Account the interference and report net values of it.
      - The osnoise: tracepoints works by hooking to existing events
      - Instead of tracing irq_entry & irq_exit, osnoise:irq_noise reports the delta

- The workload and the trace are synchronized
  - The workload can have atomic access to information collected by the trace
    - E.g., osnoise workload also reports the $ of interference that happens between two time() reads

osnoise tracer example

```
[root@f32 ~]# cd /sys/kernel/tracing/
[root@f32 tracing]# echo osnoise > current_tracer
[root@f32 tracing]# cat trace
# tracer: osnoise
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth                                   MAX
#                              || /                                                     SINGLE      Interference counters:
#                              ||||                RUNTIME      NOISE   % OF CPU  NOISE    +-----------------------------+
#           TASK-PID      CPU# ||||   TIMESTAMP    IN US       IN US  AVAILABLE  IN US     HW    NMI    IRQ   SIRQ THREAD
#              | |         |   ||||      |           |           |    |            |      |      |      |      |      |
            <...>-859     [000] ....    81.637220: 1000000        190  99.98100       9     18      0   1007     18      1
            <...>-860     [001] ....    81.638154: 1000000        656  99.93440      74     23      0   1006     16      3
            <...>-861     [002] ....    81.638193: 1000000       5675  99.43250     202      6      0   1013     25     21
            <...>-862     [003] ....    81.638242: 1000000        125  99.98750      45      1      0   1011     23      0
            <...>-863     [004] ....    81.638260: 1000000       1721  99.82790     168      7      0   1002     49     41
            <...>-864     [005] ....    81.638286: 1000000        263  99.97370      57      6      0   1006     26      2
            <...>-865     [006] ....    81.638302: 1000000        109  99.98910      21      3      0   1006     18      1
            <...>-866     [007] ....    81.638326: 1000000       7816  99.21840     107      8      0   1016     39     19
```

```
[root@f32 ~]# cd /sys/kernel/tracing/
[root@f32 tracing]# echo osnoise > current_tracer
[root@f32 tracing]# echo osnoise > set_event
[root@f32 tracing]# echo 8 > osnoise/stop_tracing_us
[root@f32 tracing]# cat trace
[...]
  osnoise/8-960     [007] d.h.  5789.857530: irq_noise: local_timer:236 start 5789.857527123 duration 1867 ns
  osnoise/8-961     [008] d.h.  5789.857532: irq_noise: local_timer:236 start 5789.857529929 duration 1845 ns
  osnoise/8-961     [008] dNh.  5789.858408: irq_noise: local_timer:236 start 5789.858404871 duration 2848 ns
migration/8-54      [008] d...  5789.858413: thread_noise: migration/8:54 start 5789.858409300 duration 3068 ns
  osnoise/8-961     [008] ....  5789.858413: sample_threshold: start 5789.858404555 duration 8812 ns interferences 2
```

rtla: an interface for the osnoise/timerlat tracers

- timerlat and osnoise work fine as **tracers**

- But they could also work as **white box** tests:
  - Report the values as a benchmark tool
  - Report the tracer if unexpected values are hit
    - All in the same session!

- With a more **intuitive interface**
  - The tracers have a nice list of config options:
    - It is possible to config runtime/period, CPU mask, ...
  - Some other things can be done automatically in user-space:
    - Setting priority to the tracer workload
    - Saving trace to a file

- Demo:
  - https://www.youtube.com/watch?v=fR4tjeI4rbs

rtla: the code

- This code is fresh!
  - osnoise/timerlat available since 5.14
  - rtla **RFC sent last Friday** (conference driven development)
    - [RFC 00/19] RTLA: An interface for osnoise/timerlat tracers
      - https://lore.kernel.org/lkml/cover.1631889858.git.bristot@kernel.org/
- It is in C
  - Uses **libtracefs**
  - Although it is not yet using eBPF, it will likely be used soon.
    - I just did not have a reason for using it in this tool yet.
    - I like eBPF!
- The tracers use rtsl code (presented last year)
  - rtla will also be the interface for rtsl
  - But rtsl was postponed because it will require some more kernel code
  - And these two tools are needed now
  - There are more tools in the pipeline
  - Bonus: the RV interface in user-space will re-use lots of this code

Discussion time

- Request
  - libtracefs:
    - A method to collect trace without consuming it from the circular buffer
    - A method to save trace to a file
      - I did one myself, but it would be better to have it in the library
      - I see that trace-cmd library will allow that to .dat file, right?
        - Is it possible to collect data
    - A method to parse histogram data in C
      - I saw a thread from Steven, but it was missing a .l file
    - Min, Max, Avg, and Over values for kernel histogram
  - How do I connect these tools with rteval?
  - How do I connect it with testing tools like LAVA?

- What do you all think?

- What feature would you like to see?