Large-scale performance monitoring framework for cloud monitoring

Run-Time Latency Detection in Production

Julien Desfossez
Michel Dagenais

Décembre 2014
École Polytechnique de Montreal

Current Status

- lttng-live protocol is merged into LTTng 2.4
- Live viewers: Babeltrace and LTTngTop [1]
- Python analysis scripts [2] are starting to produce a wide range of quick reports

Live streaming session

On the server to trace :

$ lttng create **--live 2000000 -U net://10.0.0.1**

$ lttng enable-event -k sched_switch

$ lttng enable-event -k --syscall -a

$ lttng start

On the receiving server (10.0.0.1) :

$ lttng-relayd -d

On the viewer machine :

$ lttngtop -r **10.0.0.1**

Or

$ babeltrace -i lttng-live net://10.0.0.1

Demo

- Quick demo of LTTng live, LTTngTop live and the analyzes scripts

Next steps

- We now have to take care of the various overheads:
  - Tracing rate: disk and network overhead
  - Analysis time: the bigger the trace, the longer the analysis
  - Still a lot of manual investigation required even after the analysis completes
  - How to extract relevant information from a live trace

Live state system

- A state system with no end time
- Only keep 10 seconds of detailed history
- Keep original “entry” events until they are no longer useful (garbage collector)
- Allow to query the state of any process/FD
- Allow to dump the original events in the order they were produced

Example with 10sec moving window

```
10:00:01 open /tmp/test, fd = 4
10:00:02 write 8 kB to fd 4
10:00:03 open /tmp/test2, fd = 5
10:00:04 write 16 kB to fd 5
10:00:05 close fd 5
10:00:06 state:
           - fd 4 </tmp/test> opened at 10:00:01, 8 kB write
           - fd 5 </tmp/test2> opened at 10:00:03, 16 kB write
...
10:00:15 state:
           - fd 4 </tmp/test> opened at 10:00:01, 8 kB write
```

Live state system

- Working prototype with Babeltrace live [3]
- Integration with Redis (key/value in memory DB on the network)
- Lua scripts server-side (so we can use multiple clients/providers)
- Even with redis pipelining and events processing in C, the overhead of keeping track of the state takes around 20% CPU constantly for one idle desktop
- This approach gives us a great granularity to dig into the problems with a simplified state, but the overhead is far too high for 24/7 monitoring and most of the data is useless

Focusing on outliers

- Data centers already have tools to monitor average usage of all the resources, they scale and every sysadmin is used to them
- Averages are a convenient way to hide problems
- Really complex problems appear sporadically
- Pinpointing these problems can take days of tracing and maybe more in trace analysis

Introducing latency-tracker

- Prototype work in progress to help track down latency problems [4]
- Simple API that can be called from anywhere in the kernel (tracepoints, kprobes, netfilter hooks, hardcoded in other module or the kernel tree)
- Keep track of entry/exit events and calls a callback if the delay between the two events is higher than a threshold

Using it

```
tracker = latency_tracker_create();


latency_tracker_event_in(tracker, key,
      threshold, timeout, callback);

....

latency_tracker_event_out(tracker, key);
```

If the delay between the event_in and event_out for the same key is higher than “threshold”, the callback function is called.

The timeout parameter allows to launch the callback if the event_out takes too long to arrive (off-CPU profiling).

Implemented use-cases

- Block layer latency
  - Delay between block request issue and complete
- Scheduler latency
  - Delay between sched_wakeup and sched_switch
- Network latency
  - Delay between the arrival of a packet in the network stack to the delivery in user-space (or error/drop conditions)
- IRQ latency
  - Delay between the IRQ notification and the handler entry

Configuration

- All the examples have dynamically configurable parameter options: threshold, timeout and rate limiter
- A garbage collector is available for unbalanced events in/out
- No memory allocation performed in the critical path of the events
- IRQ-safe locking (currently studying scalable HT)

Callbacks

- Must be fast enough to avoid stalling the system, we are in the critical path
- Emitting tracepoints, doing some basic aggregation, waking-up a user-space process are good callbacks
- The tracepoint emitted from this module are “stateful tracepoints”
- Additionally, we can collect all the information we need during the callback (type of FD, etc)
- Easy integration with LTTng and Ftrace

Demo

Identifying and understanding a latency with a LTTng snapshot

- Load the latency_tracker and block_latency modules
- Wait on `/proc/block_tracker` with `cat`
- When it returns, call “`lttng snapshot record`”
- The trace generated contains around 10k events (700 kB) and covers around 8 seconds
- One of the events in the trace was generated by the latency tracker, so we automatically know where to focus the analysis
- Low overhead, nothing extracted until a problem occurs (measurements in progress)

Latency tracker current state

- Prototype working and stable
- Need more testing use-cases
- Performance measurements in progress
- Hashtable scaling optimization

Latency tracker future

- Adaptative threshold depending on the exploitation conditions (with a training phase)
- Detect “noisy neighbours” on cloud instances at run-time without benchmark
- Expose custom metrics through /proc to integrate with existing monitoring tools
- Port a similar framework to user-space

Other alternatives

- SystemTap and dtrace can perform this kind of aggregation
- Not designed to be called from the kernel or other module
- Embedded build system, hard integration with other projects
- The data structures are protected with a global mutex
- A simple SystemTap is ~1500 lines of generated C
- Designed as debug tools, not monitoring with production and scaling in mind

Install it

```
apt-get install git gcc make linux-headers-generic

git clone https://github.com/jdesfossez/latency_tracker.git

cd latency_tracker

make
```

Questions ?

References

[1] git://git.lttng.org/lttngtop.git

[2] https://github.com/jdesfossez/lttng-analyses.git

[3] https://github.com/jdesfossez/babeltrace-dev.git (livestatemachine)

[4] https://github.com/jdesfossez/latency_tracker.git