LTTng: Beyond Ring-Buffer Based Tracing

Jérémie Galarneau (EfficiOS)
github.com/jgalar

Tracing Summit 2022
London, UK
11 October 2022

Photo by Ray Bilcliff

Outline

1. What is LTTng?
2. Ring–buffer tracing and its limitations
3. Triggers
4. Aggregation maps
5. Future work
6. Questions

What is LTTng?

LTTng in a nutshell

Open–source tracing framework

- First released in 2005
- Focused on system–wide introspection, **not just the kernel**

Collection of projects

- LTTng–modules: kernel tracing
- LTTng–UST: user space tracing
- LTTng–tools: tracing control

Design of LTTng

Focused on low–intrusiveness

Both kernel and user space tracers use per–CPU ring buffers

- Highly configurable
  - Memory footprint
  - Access permissions (per user/process)
  - Accommodate real–time constraints

Ring-buffer tracing and its limitations

Tracing is cheap: it can be a problem

Instrumentation is almost free when not in use

- Can be added almost everywhere
- Low cost per-event when active: ~150 ns*
- **Very easy to enable more events than really needed**

Most of LTTng features exist to mitigate this

* Xeon E5-2630; see benchmark references at the end for details

Event rules

Advanced filtering

- Wildcards, filter expressions, exclusions, log level filtering, and more
- Filter expressions converted to bytecode, interpreted at run time

Entirely dynamic

- No need to restart or reboot the kernel to change the configuration

Active debugging vs. monitoring

Debugging

- Trace to file
- Network streaming
- Live sessions

Monitoring

- Flight recorder tracing (snapshot mode)

Best of both worlds

- Keep high-level trace over a long period
- Have a low-level trace of the last few seconds available

Limitations of ring-buffer tracing

Setup can be complex

- Managing huge traces in production environments is quite a challenge
  - Storing vs. processing in place
- How do we detect the problems?

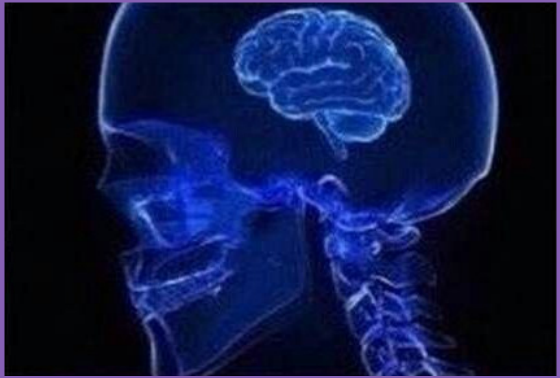
User feedback

- Consuming traces implies a significant delay
- Instrumentation already provides the information to detect issues

Triggers

Triggers

LTTng 2.10
2017



Small beginnings

A trigger associates a **condition** to an **action**

Narrow initial scope

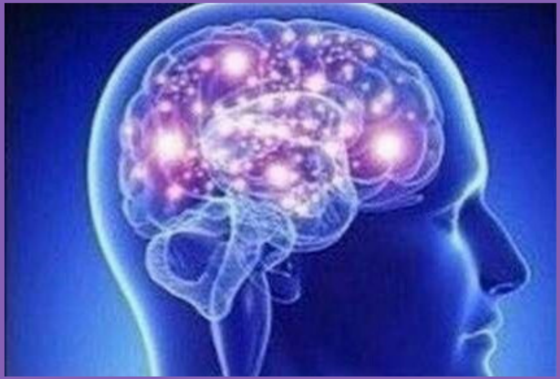
- Monitor ring–buffer usage (low/high thresholds)
- Send a notification to an external application

Used to implement tracing traffic shaping

- Disable less important event rules when I/O can't keep up

Triggers

LTTng 2.11
2019



Extended over time

New conditions

- Consumed size is greater than *X* bytes
- Ongoing session rotation
- Completed session rotation

Used to implement trace analysis pipelines

- Rotations are scheduled (on a time or size basis)
- External application notified of their availability
  - Processed in-place, sent through a message queue, or simply archived

Triggers

LTTng 2.13

2021

Smart tracepoints

New “event rule matches” condition

- Triggers can now “fire” when an event rule matches an event
- You can use existing instrumentation to react quickly

New actions

- Start, stop, rotate, and record a snapshot
- Any combination thereof

Demo 🤞

Triggers

Not a replacement for ring-buffer tracing!

Current use cases are low throughput

- Assume aggressive filtering at the source
- Cost of event rule triggers should be nonsignificant to the application

For these use cases, latency is more important than total throughput or minimizing overhead

Other limitations of ring-buffer tracing

Memory overhead

- Bandwidth
- Space

Not free in terms of CPU time (even though it's very efficient)

- Reading time and CPU number is expensive on some architectures (no VDSO implementation: requires full system calls)

Requires a post–processing step to be useful

Recording vs. aggregation: defining priorities

Recording: exact recording, order of events, precise timing, ...

```
[18:11:50.275355561] (+0.000000873) carbonara syscall_entry_recvmsg:
                                    { cpu_id = 5 }, { fd = 20, msg = 140676324897776, flags = 0 }
[18:11:50.275356143] (+0.000000582) carbonara kmem_kfree:
                                    { cpu_id = 5 }, { call_site = 0xFFFFFFFF94F5179D, ptr = 0x0 }
[18:11:50.275356397] (+0.000000254) carbonara syscall_exit_recvmsg:
                                    { cpu_id = 5 }, { ret = -11, msg = 140676324897776 }
[18:11:50.275358773] (+0.000002376) carbonara syscall_entry_recvmsg:
                                    { cpu_id = 5 }, { fd = 20, msg = 140676324897792, flags = 0 }
[18:11:50.275359412] (+0.000000639) carbonara kmem_kfree:
                                    { cpu_id = 5 }, { call_site = 0xFFFFFFFF94F5179D, ptr = 0x0 }
[18:11:50.275359733] (+0.000000321) carbonara syscall_exit_recvmsg:
                                    { cpu_id = 5 }, { ret = -11, msg = 140676324897792 }
```

Recording vs. aggregation: defining priorities

Aggregation: simply count occurrences of event rule matches

```
+------------------------------------------+-------------+----+-----+
| key                                      |         val | uf | of  |
+------------------------------------------+-------------+----+-----+
| syscall_entry_recvmsg                    |   3,404,391 |  0 |   0 |
+------------------------------------------+-------------+----+-----+
| kmem_kfree                               |     611,014 |  0 |   0 |
+------------------------------------------+-------------+----+-----+
```

Aggregation maps

Introducing **aggregation maps**

LTTng 2.14

Est. 2022

Per-CPU arrays of counters

- Associate a key (string) to a value
- Configurable width (32/64 bits)
- Configurable size (number of counters)
- Indicates overflow

Not a new concept for kernel users

(`BPF_MAP_TYPE_PERCPU_ARRAY`)

- Available to the user space tracer too

Performance

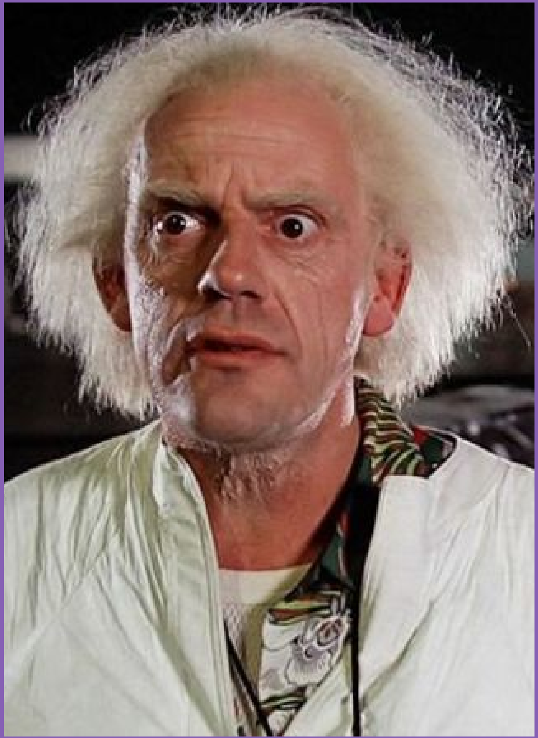
As expected, a lot cheaper than ring-buffer tracing

Method	Time per event (ns)	σ (stdev)
LTTng-UST ring-buffer (4 × 8 MiB)	158	0.222
LTTng-UST map	**43.3**	**0.656**
LTTng-modules ring-buffer (4 × 8 MiB)	151	0.824
LTTng-modules maps	**44.8**	**0.219**
eBPF per-CPU array	57.0	0.683

(Xeon E5-2630, see benchmark references at the end for details)

Demo 🤞

Future



New operations

- Native histogram support
- Decrement value
- Use event payload
- Use event record size

Performance improvements

- Make LTTng-UST `rseq()`-aware
- Reduce impact of kernel mitigations

Links

www.efficios.com

www.lttng.org

Photo by Lukas Kloeppel

Benchmark code:
www.github.com/jgalar/LinuxCon2022-Benchmarks