Tracing tools for low latency microservices

Eya-Tom Augustin SANGAM

June 2023

Polytechnique Montréal

DORSAL Laboratory

Agenda

- Context and goal
- Related work
- Considerations
- Proposed solution
- Benchmarks
- Current and future work
- Conclusion

Context and goal

We have :

- C Microservices communicating with each other using ZeroMQ

We want to **collect telemetry data (TD)** :

- Application logs
- Application and host metrics
- Requests traces (aka spans)

Related Work : LTTng and LTTng-UST

- LTTng can help collect host metrics (CPU, RAM usage etc.)
- LTTng-UST can help collect applications metrics, applications logs and requests traces
  - Advantage: Record or not specific TD at runtime
  - Problem: We need to define a protocol over the standard LTTng-UST logging library for trace collection, metrics collection and context propagation
    - OpenTelemetry Specification already does that

Related Work : OpenTelemetry

- OpenTelemetry (OTel) is becoming the industry standard for creating and collecting TD
- OTel specification describes cross-language requirements and expectations for all OTel implementations.
- Many visualisation tools like Jaeger, Prometheus support OTel data schemas out of the box
- OTel created the OTel Collector which is a vendor-agnostic way to receive, process and export TD

Considerations (1/2)

- We want to do cross-hosts TD analysis
  - We need to bring all TD together at some point

- Some hosts have limited hard drive storage
  - A filtering mechanism is required to minimize the amount of data saved on the disk
  - e.g., we should be able to decide at runtime whether we want to save heartbeat traces or not

Considerations (2/2)

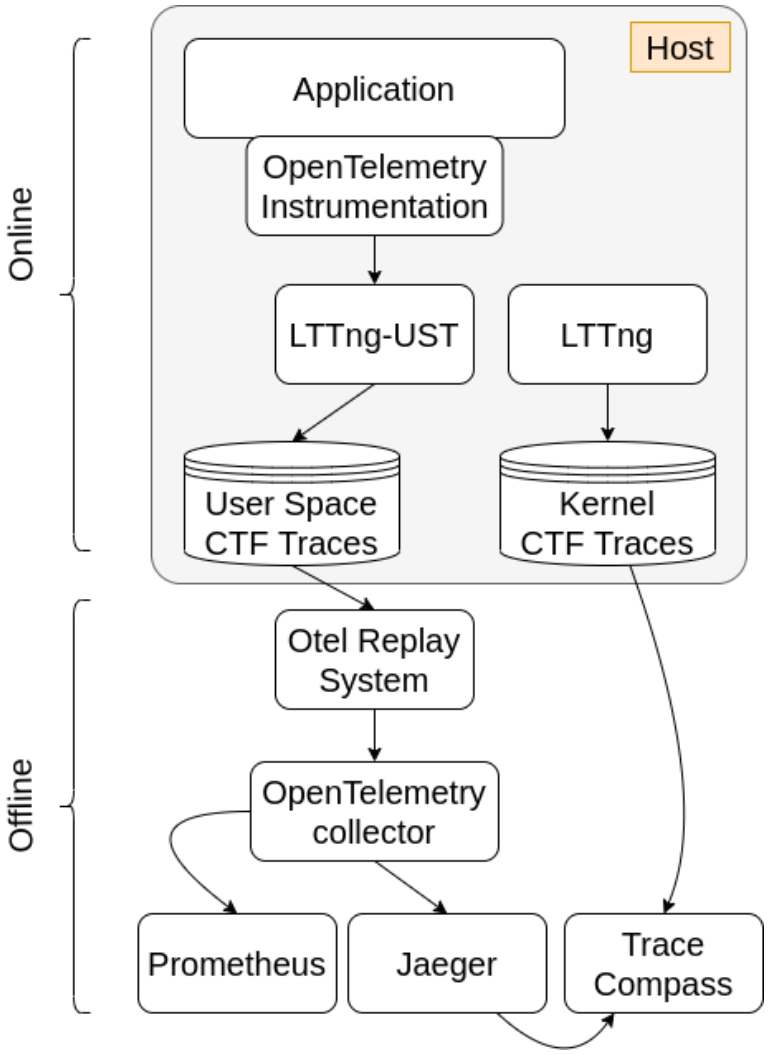
- Some applications run on hosts with limited resources
  - Installing OTel collector or an observability backend may highly affect the application behaviour

- Live monitoring is not required.

Proposed solution (1/2)

Online part
(When application runs)

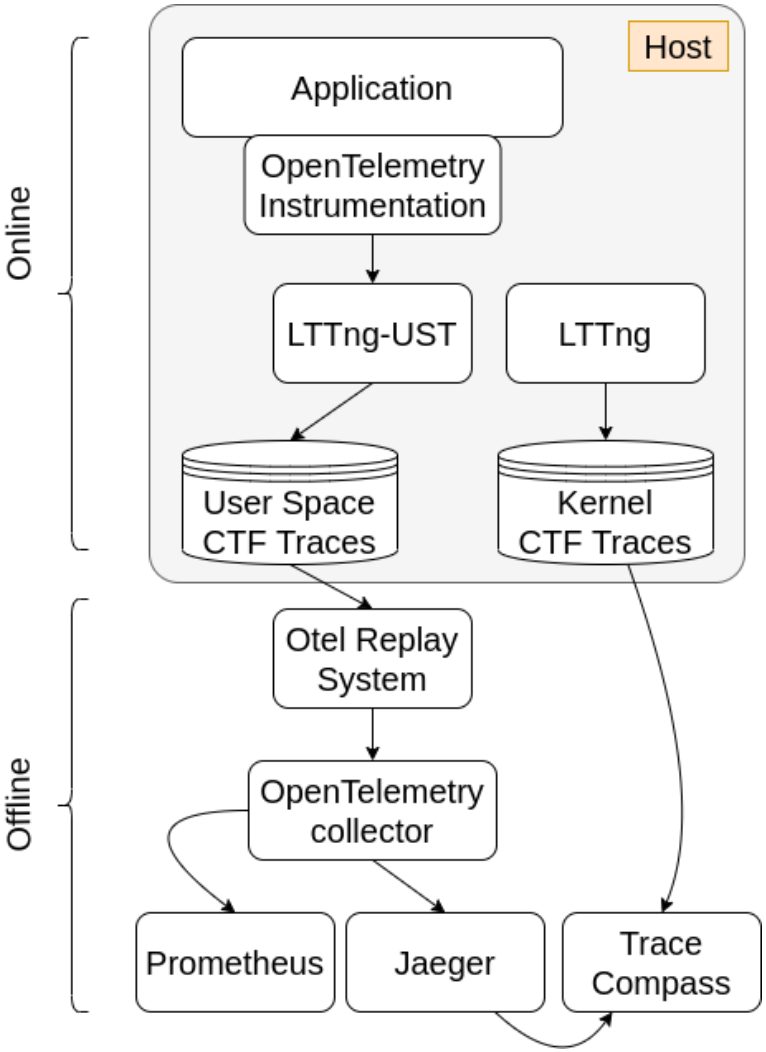
- LTTng is used to collect host metrics
- We use OTel instrumentation
- TD generated is logged to LTTng-UST and saved in CTF files
  - We can control what runtime data we save this way



Proposed solution (2/2)

Offline part
(Only when we want to do analysis)

- CTF files are copied from the host
- Host metrics could be viewed in Trace Compass directly
- The OTel Replay System reads TD and sends them to the OTel collector which will send them later to observability backends (Jaeger, Prometheus, etc.)



Solution

- Otel C wrapper
  - https://github.com/augustinsangam/opentelemetry-c
- Simple ZeroMQ client, proxy and server application traced using opentelemetry-c
  - https://github.com/augustinsangam/opentelemetry-c-demo
- OTel Replay System which reads the telemetry data and send them to the OTel collector which will send them later to observability backends (Jaeger, Prometheus, etc.)
  - https://github.com/augustinsangam/otel-replayer
- Benchmarks
  - https://github.com/augustinsangam/opentelemetry-c-performance
  - Deep dive doc

Trace Benchmarks (1/3)

- Scenario : Start a span and end it right away. Measure the time to do the operation.
- Multiple configurations tested :
  - LTTng configuration: No LTTng session running (NLS), LTTng session without recording (LSWR), LTTng session recording UST telemetry data (LSRU), LTTng remote session recording UST telemetry data (LRSRU)
  - Type of instrumentation: No instrumentation (NI), OpenTelemetry (OTel)
  - Type of exporter: LTTng Exporter (LE), Local OTel collector (LOC), Remote OTel collector (ROC)
  - OTel Traces Processor (applies only for traces benchmarks): Simple (SP), Batching processor (BP)

Trace Benchmarks (2/3)

- Exporting spans one by one as they are created using remote OpenTelemetry collector vs using Local Lttng exporter vs Exporting one by one to remote LTTng

Test cases	NLS-OTel-ROC-SP	LSRU-OTel-LE-SP	LRSRU-OTel-LE-SP
n spans	5,000	20,000	20,000
min (ns)	1,931,562	94,947	61,689
mean (ns)	2,945,936	288,689	287,596
max (ns)	15,251,23	957,472	1,512,586
median (ns)	2,796,951	305,975	283,274
std (ns)	478,621	22,681	23,003
real (ms)	65,391	208,483	208,473
user (ms)	8,079	6,029	5,969
sys (ms)	369	407	461

When using simple processor, spans are processed synchronously after they are created. In this situation, using LTTng to log spans should be preferred over sending traces overs the network

Trace Benchmarks (3/3)

- Same comparison but we export traces every 5s in batch of a maximum of 512 spans in a background thread

Test cases	NLS-OTel-ROC-BP	LSRU-OTel-LE-BP	LRSRU-OTel-LE-BP
n spans	20,000	20,000	20,000
min (ns)	21,101	23,063	43,641
mean (ns)	116,657	117,143	116,836
max (ns)	455,129	536,921	396,297
median (ns)	117,134	113,691	131,189
std (ns)	9,668	9,394	9,482
real (ms)	204,911	205,077	205,048
user (ms)	3,663	3,259	3,268
sys (ms)	330	405	379

- In production, the remote collector could be in a different network, which could make these results vary
- The preferred solution should be logging all traces locally to LTTng. This avoids running an OTel collector and dealing with all the network communications troubles it could add

Metrics Benchmarks

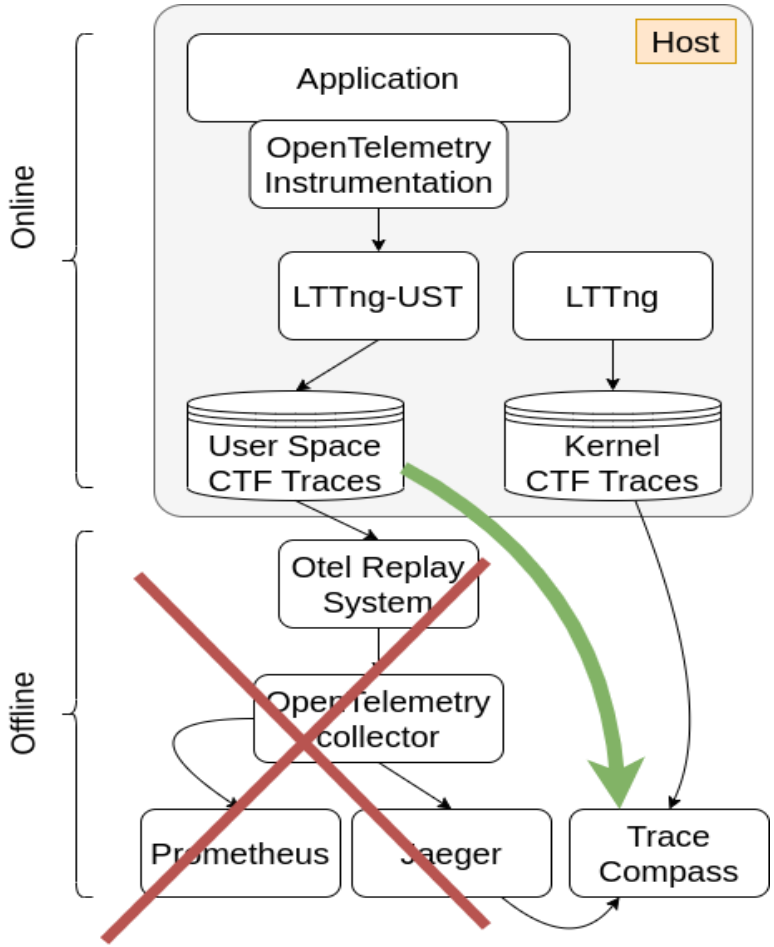
- Pattern: We measure the time to do an operation without collecting any kind of metrics. And we repeat the same operation while exporting metrics every 500/1000 ms.
- Comparison: No metrics vs exporting metrics to a remote Otel collector vs exporting metrics to a local LTTng session vs exporting metrics to a remote LTTng session

Scenarios	NI		NLS-OTel-ROC		LSRU-OTel-LE		LRSRU-OTel-LE	
Export delay (ms)	500	1000	500	1000	500	1000	500	1000
duration (ms)	114,541	114,539	115,290	115,030	114,712	114,681	114,649	114,572
overhead (%)			0.654	0.656	0.149	0.151	0.094	0.096
cpu time (ms)	114,537	114,535	115,816	115,348	114,836	114,749	114,776	114,650
cpu time overhead (%)			1.116	0.71	0.261	0.187	0.208	0.1

For all configurations, the execution time overhead is less than 1.2% and the larger the export interval, the lower the overhead.
LTTng Metrics exporter is approximatively 50% faster than the remote exporter but the CPU time spent in user space is similar for the two configurations.

Current and future work

- Analyse Otel userspace traces directly in Tracecompass without having to use any telemetry backend
- Add Spans view : Support Otel schemas, trace synchronisation and add filtering capabilities
- Metrics view : Add counters view and support basic query language (ex : metric1 + metric2)



Thanks !

Questions, ideas, remarks ?

Appendix : Different ways of collecting telemetry data