POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

HUMANITAS

Transparent Trace Annotation for Performance Debugging in Microservice-oriented Systems

Adel Belkhiri and Gabriela Nicolescu

HESL Lab

DORSAL Progress Report Meeting

Montréal, June 02, 2023

Agenda

Introduction

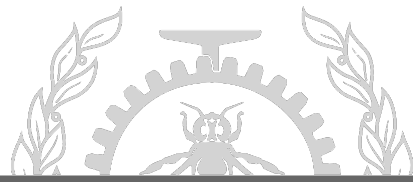
- The microservice architecture
- Software tracing and performance debugging

Motivation

Literature analysis

Proposed solution: framework for a transparent annotation of traces

Conclusion and future work



Microservice Architecture

- Microservices is a software architecture in which the application is implemented as a collection of small, independent, and loosely-coupled services that communicate through well-defined interfaces (e.g., RESTful APIs)

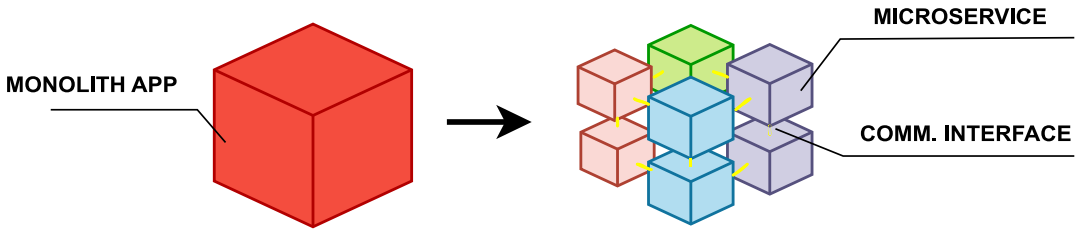


Figure: Monolithic architecture *vs.* microservices

- It presents indeed many advantages .. but complicates the debugging of latency-related problems :/

Software Tracing (1)

- There exist many tracers with different tracing capabilities and scopes:
  - **Standalone applications:** Ftrace, Systemtap, Uftrace, Dtrace, and LTTng
  - **Distributed applications:** Jaeger and Zipkin
    - Span: A tagged time interval denoting the execution latency of a particular operation (e.g., RPC or function calls)

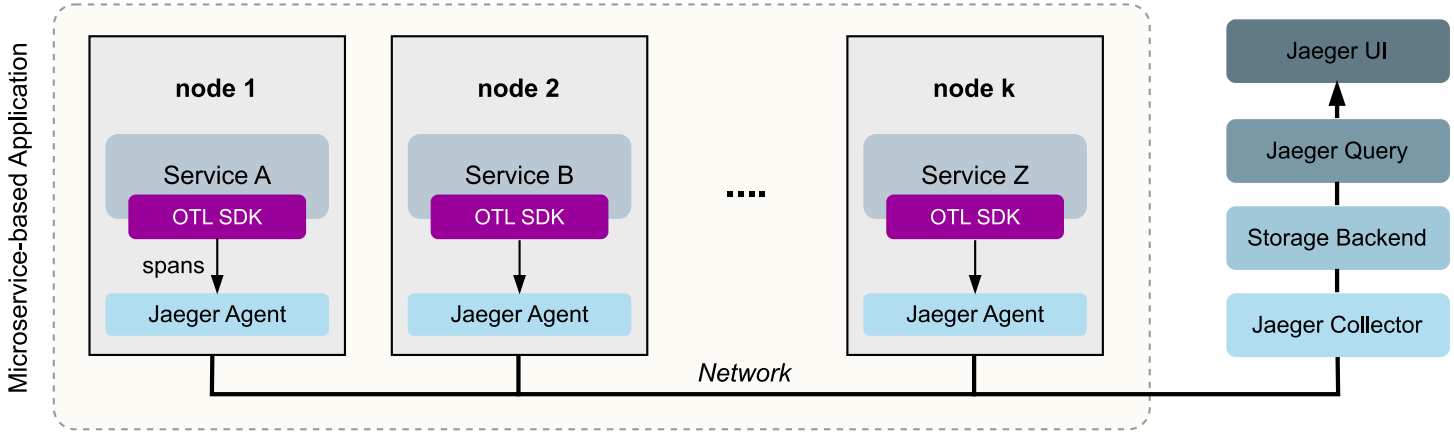


Figure: Reference architecture for a distributed tracer

Software Tracing (2)

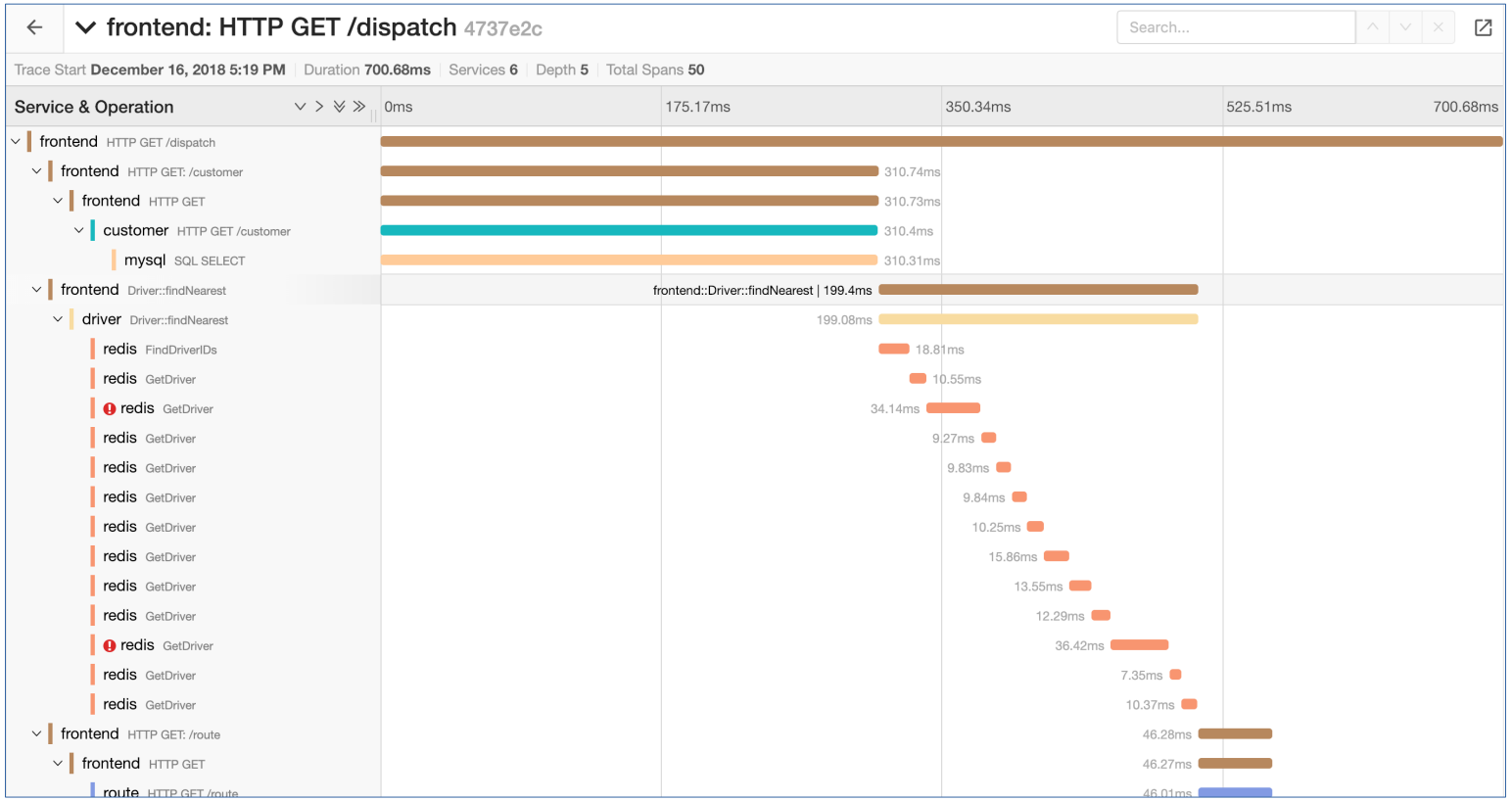


Figure: Jaeger UI showing microservices involved in processing a user request (a ride order) along with resulted spans

Motivation

- **Problem:** Distributed tracers can pinpoint slow services and detect latency-related problems, but cannot be used for identifying the causes of performance issues

- **Solution:** A framework for annotating traces generated by distributed tracers with useful information extracted from the Linux kernel

Literature Analysis

- Literature reports many open-source and proprietary **tracing tools**, such as Canopy [1], Dapper [2], Jaeger [3], and Zipkin [4]

  ☹ Cannot diagnose the causes of latency-related problems as they only leverage high-level data

- Frameworks in [5] and [6] use **service mesh** (e.g., Istio/Envoy) to extract metadata from microservices requests and generate tracing data.

  ☹ Only eliminate the need to instrument the application's source code to generate traces

- Frameworks in [5] and [6] propose **cross-layer tracing** for collecting and synchronizing kernel and distributed request events, using patched Jaeger clients and Linux Kernel

  ☹ Very intrusive as they require the modification of the tracer and the Kernel

The Span Latency Tracker Framework

- **Span latency tracker**

– Add annotation to long-lasting spans generated by monitored microservices to help understand the causes of unusual latencies

– Annotation is derived from kernel events: system calls, application/kernel call stack, and system wide metrics (example: average preemption time of threads)

– Architecture:

1) A set of monitoring libraries to preload, depending on the programming languages in which microservices were implemented (C++, GO, Python, etc.)

2) Three kernel modules: *span-latency-tracker.ko*, *latency-begin-end.ko*, and *latency-tracker.ko* [10]

Framework Architecture

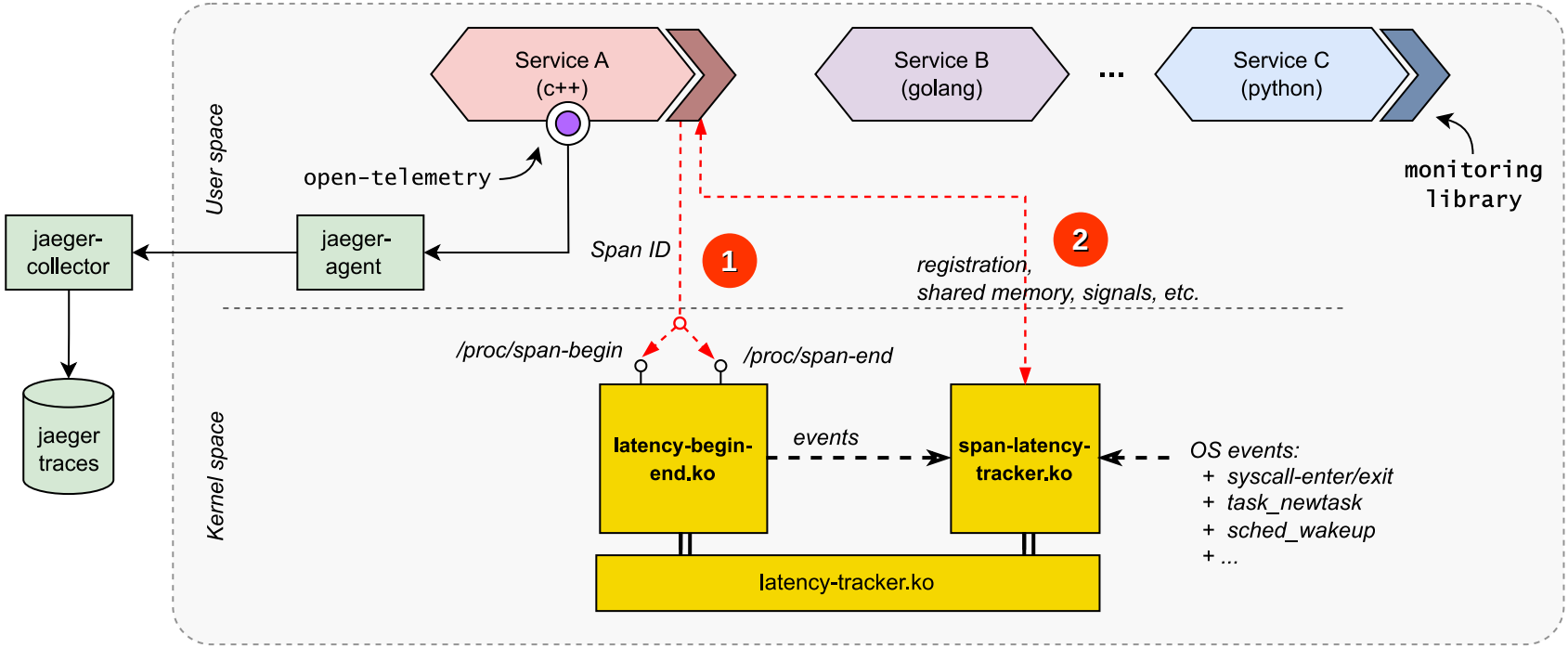


Figure: Proposed framework is composed of kernel modules and a set of monitoring libraries to pre-load when launching microservices

Proposed Framework

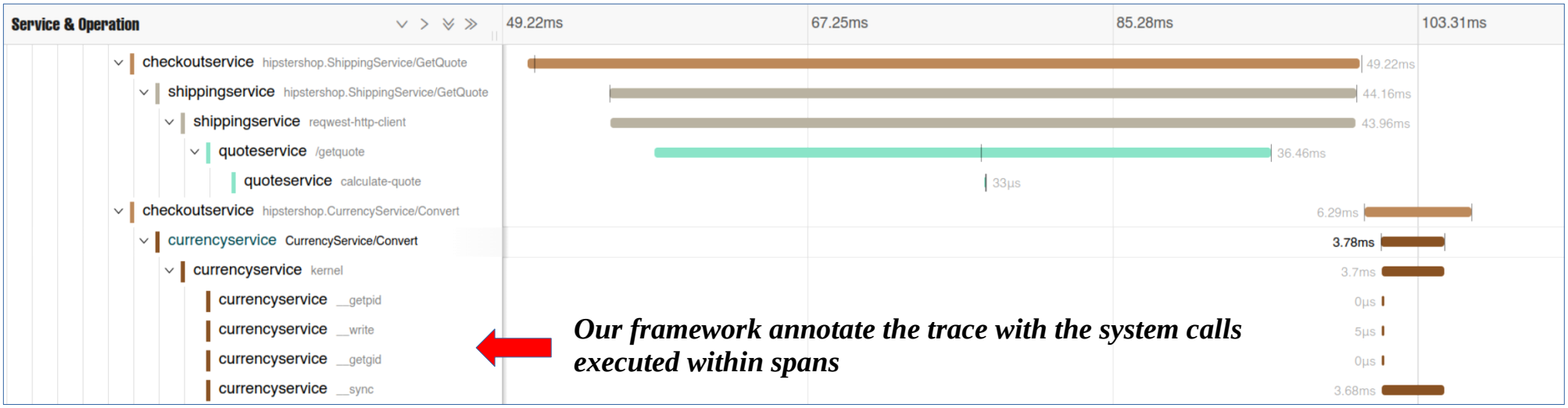


Figure: Annotating *CurrencyService/Convert* operation with the *system calls* executed within it

- System calls are added as sub-spans, and callstacks and metric values as span attributes and events

- The tool is very customizable: traces can be annotated with a subset of system calls of interest, user can choose which data to use for annotation and set a latency threshold for spans to be tracked, etc.

Results & Discussion (1)

- Overhead analysis based on the evaluation of the Astronomy Shop [9] application performance.

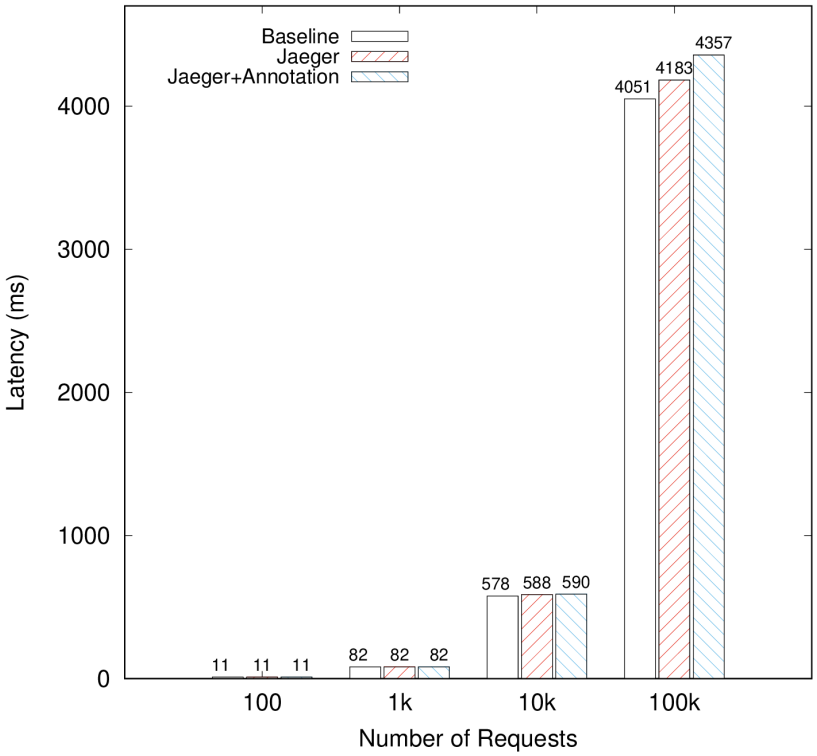


Fig. A: Execution time when tracing is not enabled, traced with Jaeger, and traced with our tool.

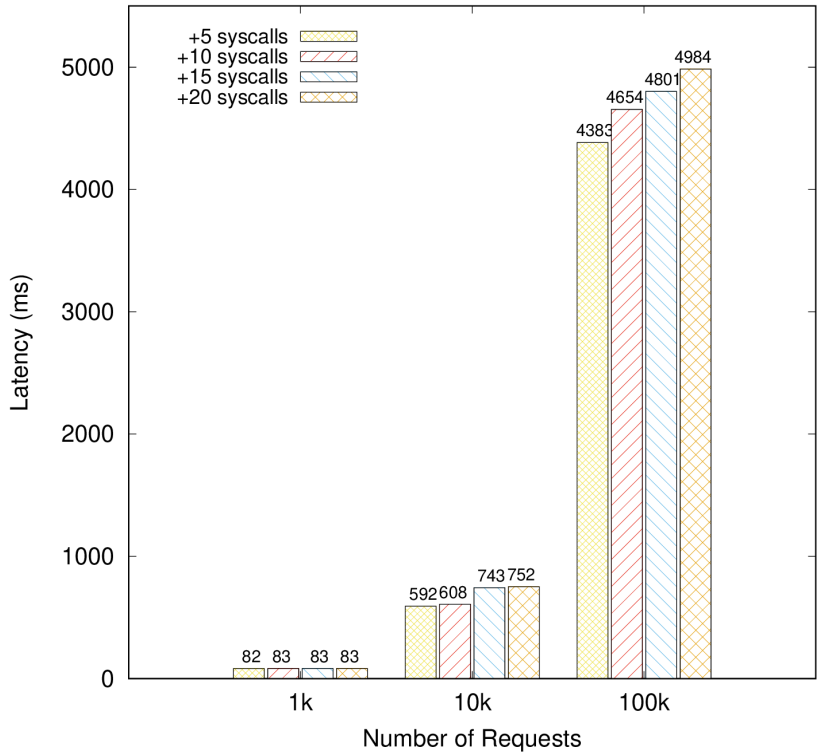


Fig. B: Execution time depending on the numbers of requests and injected system calls per span.

Results & Discussion (2)

- **Advantages:**
  - Proposed framework can be coupled with any distributed tracer that support OpenTelemetry
  - Non-intrusive approach for annotating traces
- **Limitations**
  - Incapacity to intercept system calls of the vDSO type.
  - Microservices written in bytecode-based languages (Java) are not supported yet.

Conclusion

- Framework for annotating distributed traces with information derived from kernel events
  - Particularly efficient in diagnosing the causes of long-tail latencies
  - Open-source*, non-intrusive, and induces low-overhead

Future Work

1) Extend the annotation mechanism to support bytecode-based microservices
2) Include more metrics and information into the trace annotation

*Authors' GitHub : https://github.com/adel-belkhiri

Questions?

adel.belkhiri@polymtl.ca

Bibliographie

[1] Jonathan Kaldor et al., 2017. Canopy: An End-to-End Performance Tracing And Analysis System. Proceedings of the 26th Symposium on Operating Systems Principles (2017), 34–50.

[2] Benjamin H. Sigelman et al., 2010. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical Report. Google Inc.

[3] Jaegertracing.io. 2022. Jaeger: Open Source, End-to-End Distributed Tracing. http://jaegertracing.io

[4] Zipkin.io. 2022. Zipkin. https://zipkin.io

[5] Donghun Cha et al., 2021. Service Mesh Based Distributed Tracing System. International Conference on Information and Communication Technology Convergence (2021).

[6] Chih-Cheng Hung et al., 2019. Transparent tracing of microservice-based applications. Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (2019).

[7] Loïc Gelle et al., 2021. Combining Distributed and Kernel Tracing for Performance Analysis of Cloud Applications. Electronics 10, 21 (2021), 2610.

[8] Harshal Sheth and Andrew Sun. 2018. Skua: Extending Distributed-Systems Tracing into the Linux Kernel. In Proceedings of the DevConf.US. 17–19.

[9] OpenTelemetry CNCF. 2022. Astronomy Shop, the OpenTelemetry Demo. https://github.com/open-telemetry/opentelemetry-demo

[10] Julien Desfossez, Mathieu Desnoyers, and Michel R. Dagenais. 2016. Runtime latency detection and analysis. Software: Practice and Experience 46, 10 (2016), 1397–1409. https://doi.org/10.1002/spe.2389