Transparent Trace Annotation for Performance Debugging in Microservice-oriented Systems
(Work In Progress Paper)

Adel Belkhiri, Ahmad S. Bushehri, Felipe G. de Magalhaes, and Gabriela Nicolescu

HESL Lab. (Polytechnique Montréal) & HUMANITAS Inc.

14th ACM/SPEC International Conference on Performance Engineering (ICPE)
Coimbra, Portugal, April 15-19, 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)

- Recording low-level information about a program execution, as a series of events
- Each event is characterized by a name, timestamp, and payload (e.g., values of a program variables)
- Tracers are powerful tools that are widely used for diagnosing applications performance bugs

**Figure:** Analogy between a program and a bike, where instrumentation is like painting the bike's tires!
Software Tracing (2)

- 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
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 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] attach sidecars (e.g., Istio/Envoy) to containers to extract metadata from microservices requests and generate tracing data.

- Only eliminate the need to instrument the application’s source code to generate traces


- 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
**Proposed Solution**

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

Our framework annotate the trace with the system calls executed within spans

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)


**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

This project is a work in progress, so if you have ideas on how to improve it, please let us know!
Questions?
adel.belkhiri@polymtl.ca
Bibliographie


