Distributed Traces of Microservices Architecture: Publicly Available Datasets

Abhishek Tiwari

Citation: *A. Tiwari*, "Distributed Traces of Microservices Architecture: Publicly Available Datasets", Abhishek Tiwari, 2024.
doi:10.59350/6w839-77r53

Published on: December 27, 2024

Distributed traces are essential for understanding the behaviour, performance, and reliability of microservices architecture. They can be used to surface meaningful observations about service dependencies, call graphs, and runtime dynamics, enabling software engineers and scientists to develop new tools for optimisation and fault diagnosis. However, there is a clear gap in terms of publicly available real-world distributed tracing datasets of microservices architecture to validate theoretical research with actual observations.

In recent years several comprehensive distributed tracing datasets containing detailed runtime metrics of microservices architecture have been published. These datasets fall into two categories: (1) traces from real-world large-scale microservices ecosystems, such as those at Meta, Alibaba, and Uber, and (2) traces generated using testbeds. While real-world datasets are rich in scale, coverage, and fidelity, testbed-generated datasets often lack these characteristics. This article curates publicly available distributed trace datasets useful for research related to microservices architecture .

Traces from Real-World Microservices Ecosystems

Meta's Distributed Canopy Traces

Meta's distributed tracing dataset provides an in-depth view of the company's large-scale microservices architecture underpinning social media services like Facebook and Instagram. The dataset includes 6.5 million traces representing 0.5% of traces collected from a single day (2022/12/21). These traces capture Meta's heterogeneous and highly dynamic topology, marked by frequent changes and high concurrency in service interactions. For more insight into Meta's microservices architecture see [1], [2], and [3].

Canopy

At the core of Meta's tracing infrastructure is Canopy, their distributed tracing framework that operates similarly to other event-based tracing systems but with some unique characteristics. Canopy allows software developers at Meta to define request workflows and capture critical information through trace points, which are similar to log messages embedded in the service code. At runtime, trace points are annotated with request context and timestamps. The framework then aggregates records with identical trace IDs and orders them based on happens-before relationships to create comprehensive traces.

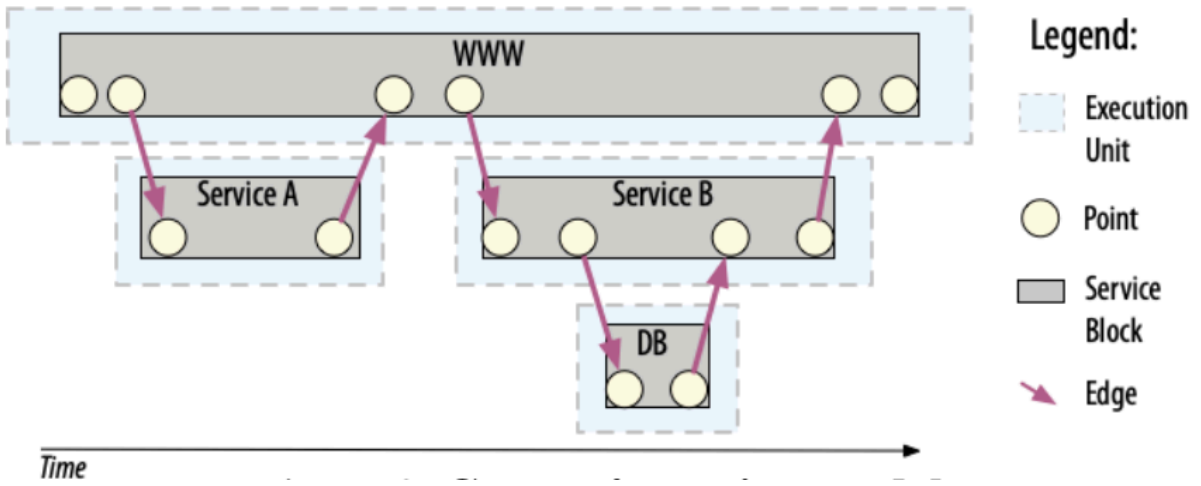

Figure 1: Canopy Traces. Image credits Huye et. al

One of Canopy's key features is its sampling approach. Rather than implementing uniform sampling across all services, Canopy supports per-service sampling profiles with customisable policies. These profiles can be attached to any service and specify sampling rules based on request attributes. This flexible sampling strategy means that traces may start at services deep within the topology or end prematurely at certain services based on their sampling profiles.

Data

The released dataset contains three main components: Service History logs documenting service deployments and deprecations over 22 months, Service Endpoints data listing exposed endpoints for each service over 30 days, and Distributed Traces collected by Canopy over 30 days. The traces component alone comprises 13.1 petabytes of data, with only a subset containing 6.5 million traces made available.

Potential use cases

Researchers can study trace properties like depth, width, and concurrency patterns to understand how requests propagate through call chain. The data reveals that most traces are relatively shallow but wide, suggesting extensive use of parallel processing and data sharding.

Another promising research direction is studying the predictability of request call graph. The traces show that while some properties like the set of services called by a parent service are somewhat predictable, other aspects like concurrency patterns and total number of calls are more variable. This understanding could help to develop more effective monitoring, debugging, and capacity planning tools.

The dataset also highlights important challenges in microservice observability. Many trace branches terminate prematurely due to rate limiting and/or non-instrumented services, with deeper call paths being overly affected with these issues. This insight is valuable for researchers working on improving observability solutions and developing techniques to work with incomplete tracing data.

For those studying testing and deployment of microservices, the data reveals that Meta's architecture includes various types of services beyond the typical microservices model. Some services are more monolithic in nature, exposing thousands of endpoints, while others are "ill-fitting" entities that require custom handling for scheduling and scaling. This heterogeneity presents interesting challenges for developing representative test environments and deployment models.

The scale and fidelity of this dataset make it valuable for the microservices research community. It provides ground truth data about how microservices architecture operate at scale, which can help validate existing research and open new directions. Researchers can use this data to develop more realistic microservices benchmarks, improve observability tools, and develop new best practices for managing large-scale distributed systems.

Limitations

Due to sampling traces may not cover all interactions, and the specific sampling policies used could affect analysis outcomes. Additionally, the data represents just one organisation's approach to microservices, albeit at an impressive scale. Future research would benefit from similar datasets from other organisations to understand different architectural patterns and trade-offs.

Alibaba's Cluster Trace Data

The Alibaba microservices trace dataset contain detailed runtime metrics from nearly twenty thousand microservices collected from Alibaba production clusters running on over ten thousand bare-metal nodes during a twelve-hour period. Two different data sets were published - one in 2021 another in 2022. Although original study covered 10B traces among nearly 20k microservices in 7 days from Alibaba clusters, both published trace data are sampled at 0.5% rate and includes 20M+ traces. For more insight into Alibaba's microservices architecture see [4], [5], and [3].

Application Real-Time Monitoring Service

Alibaba uses the Application Real-Time Monitoring Service (ARMS) system for trace collection, which is functionally similar to Google's Dapper distributed tracing framework. The system collects comprehensive metrics at multiple levels - from hardware-level indicators like cache misses to operating system metrics like CPU and memory utilisation, as well as application-level data including Java virtual machine heap utilisation and garbage collection statistics.

Data

The trace data is structured into four main components. The first component captures bare-metal node runtime information, recording CPU and memory utilisation across over 1,300 nodes. The second component contains runtime information for over 90,000 containers running across these nodes, tracking resource utilisation at the container level. The third component provides detailed information about microservices call rates and response times. Final component contains sampled call graph data showing the relationships and dependencies between microservices.

The traces capture three distinct types of communication paradigms between microservices: inter-process communication, remote invocation, and indirect communication. Each call record includes detailed metadata such as timestamp, trace ID, RPC ID, and response time metrics. The dataset also distinguishes between stateless services and stateful services like databases and caches.

Alibaba dataset schemas and formats are well-documented and the traces are stored in standard formats, making it easier for researchers to process and analyse the data.

Shuffled version

In 2024, Huye et. al released shuffled copies of the original 2021 and 2022 Alibaba's trace datasets. The tables have been shuffled such that all rows for a traceid are contained within the same file. Shuffled trace files can be found on Harvard Dataverse: 2021, 2022.

Potential use cases

For researchers, this dataset enables several important areas of study. Performance analysis researchers can investigate how resource utilisation, communication patterns, and deployment configurations drives latency and overall performance of microservices. The detailed call graph data allows researchers to study microservices call patterns and their influence on system design and optimisation. Resource management researchers can use the traces to develop and evaluate scheduling algorithms and methods of resource allocation for microservice deployments.

The dataset is particularly valuable for studying how microservices behave under real production workloads. The traces capture both the regular periodic patterns in microservice behavior as well as variations and exceptions that occur in production environments. This allows researchers to create more realistic models and benchmarks compared to synthetic workload generators. The inclusion of multiple metric types and granular timing information allows for correlation of events and behaviors at different scales (e.g. nodes vs. containers vs. service).

For academic settings, these traces provide an invaluable resource for teaching distributed systems concepts and giving researchers exposure to real-world microservices deployment patterns. The scale and complexity of the traced system helps bridge the gap between simplified academic examples and production reality.

Limitations

While the dataset is comprehensive, researchers should note some limitations. The traces are sampled rather than complete, with call graphs sampled at a 0.5% rate to manage data volume. Additionally, some fields have been anonymised or removed for privacy and security reasons. However, the sampling is consistent and the key relationships and patterns are preserved, maintaining the dataset's utility for most research purposes.

Uber's CRISP Dataset

Uber's CRISP dataset supports critical path analysis within its microservices architecture (see [6]). It covers approximately 40,000 endpoints, recording millions of remote procedure call (RPC) interactions per day.

Jaeger

Uber uses Jaeger as distributed tracing frameworks. Jaeger agents run on every host, collecting trace spans over UDP from applications running on that host. These agents forward the spans to Jaeger collectors, which then buffer the data into Apache Kafka. The traces are then processed by multiple consumers: a Jaeger ingester that stores complete traces in Docstore (a distributed SQL database), a Jaeger indexer that enables searching through spans via Sawmill (a schema-agnostic logging platform), and Apache Flink jobs that generate multi-hop dependency graphs.

Dataset

The dataset mentioned in original CRISP paper includes more than 1 million traces, 4k services, and 40k endpoints but released data only contains a subset of 100k traces.

The traces have been sanitised to protect privacy while retaining their analytical value. Service names and endpoint information have been consistently mapped to anonymous identifiers, and absolute timestamps have been randomly shifted while keeping relative timing relationships within traces.

The dataset's documentation includes tools and scripts for analysis, particularly the CRISP framework for critical path analysis. This enables researchers to process the traces efficiently and draw meaningful observations about system behavior, performance bottlenecks, and architectural patterns.

Potential use cases

For researchers, this dataset presents some key opportunities. First, it enables the study of large-scale distributed systems behavior and performance characteristics. The traces contain detailed timing

information, showing how requests flow through multiple services and identifying critical paths. This can feed into optimisation research related to distributed systems and architectural patterns. More importantly Jaeger is widely used tracing framework making any research and tools developed using this dataset extensible.

Second, the dataset is particularly valuable for research into anomaly detection and system health monitoring. The traces include both normal operation patterns and examples of system issues, useful for developing and evaluating automated algorithms. The data's scale and complexity also makes it an excellent test bed for machine learning approaches to system monitoring.

Third, researchers can use these traces to study microservice architecture patterns and factors influencing their performance. With data from thousands of services interacting in production, the dataset can be used as good real-word reference on how different architectural decisions affect latency, reliability, system properties and behaviors at scale.

Limitation

One limitation researchers should note is that the published traces represent a subset of Uber's total trace data. While still substantial, with some traces containing up to 275,000 spans and showing concurrent operations across hundreds of services, they may not cover all call paths or rare failure modes present in the full production system.

Uber's Error Study Dataset

In addition to CRISP, Uber has released a dataset focusing on RPC error analysis (see [7]). This dataset includes over 11 billion RPCs and 1,900 user-facing endpoints, documenting non-fatal errors and their impacts on latency. Dataset contains comprehensive microservices traces (more than 1.5 million) collected from Uber microservices architecture.

Jaeger

The trace data was collected using Jaeger. Jaeger's instrumentation captures crucial metadata for each RPC call, including duration, timestamp, and various tags and logs. This information is combined into "spans" that represent individual RPCs performed on behalf of user requests. The system employs trace context metadata with unique trace IDs that are transmitted in-band across process boundaries, enabling the correlation of related spans into complete traces.

Dataset

Uber's implementation of distributed tracing operates at an impressive scale, processing approximately 840 million traces formed from 210 billion spans daily, averaging around 3 million spans per second. To manage this volume while maintaining system performance, Uber employs adaptive sampling rates below 1% for user-exposed endpoints. Despite this sampling, the collected data provides a statistically significant representation of system behavior.

The released dataset has been carefully sanitised to protect proprietary information while preserving critical performance characteristics necessary for academic research. It includes two main trace files containing over 1.5 million traces, along with a driver-sanitised file focusing on app-launch use cases. To ensure privacy, the data has undergone several modifications: unrelated fields and tags have been removed while retaining error-related information, service mappings have been consistently anonymised across traces, and trace timestamps have been randomly shifted to prevent temporal analysis.

Potential use cases

This dataset is particularly useful for studying error patterns and their impact on system performance. The traces retain error-related tags, allowing researchers to analyse how errors propagate through microservices call graph and affect overall system reliability. The data can also be used to investigate latency reduction techniques, as it provides detailed timing information about service interactions and their dependencies.

The scale and complexity of the dataset make it particularly valuable for developing and validating new approaches to performance optimisation in distributed systems. Researchers can use it to study critical path analysis, identify bottlenecks, and propose improvements to service architecture designs. The consistency of service mappings across traces enables longitudinal studies of service behavior and interaction patterns.

The dataset is especially relevant for research into microservice observability and monitoring. It can be used to develop new techniques for anomaly detection, root cause analysis, and performance prediction. The inclusion of both successful and error cases provides a realistic foundation for studying fault tolerance and resilience mechanisms in microservice architectures.

For machine learning researchers, the dataset offers opportunities to develop and evaluate models for performance prediction, anomaly detection, and system behavior analysis. The large number of traces provides sufficient data for training sophisticated models, while the preserved relationships between services enable the study of complex interaction patterns.

Limitation

The dataset's availability through Zenodo ensures long-term accessibility and reproducibility of research findings. However, researchers should note the significant storage and processing requirements, as each compressed file requires 300-500GB of disk space when decompressed. This consideration may influence research methodology and tool development approaches. Also, mapping is inconsistent between Uber's error and CRISP datasets so they can not be used together.

Testbed-Generated Traces

Pre-processed Tracing Data for Popular Microservice Benchmarks

This dataset contains tracing information from four enterprise-level microservice benchmarks deployed on a dedicated Kubernetes cluster comprising 15 heterogeneous nodes (see [8]). Rather than using sampling, the dataset focuses on specific request types from for four benchmark application: Social Network, Media Service, Hotel Reservation and Train Ticket Booking.

The source applications are derived from two established benchmark suites: DeathStarBench and Train-Ticket. To simulate performance issues, it used performance anomaly injector from the FIRM framework.

The dataset underwent preprocessing from raw data generated by FIRM's tracing system. The data is categorised based on the location of performance anomalies within the microservices architecture.

Anomalies in Microservice Architecture (train-ticket) based on version configurations

This work provides ten datasets containing monitoring data (logs, Jaeger Traces and Prometheus KPI data) for microservices (see [9]). The datasets was generated using Train-Ticket.

Comparison of Datasets

The following table summarises the key features of the discussed datasets:

Dataset	Source	Scale	Characteristics	Applications	URL
Huye et. al (2023)	Meta	18.5k services, 12M service instances, 6.5M traces	Highly dynamic topology, heterogeneous services	Topology modeling, performance prediction	URL
Luo et. al (2022)	Alibaba	20K services, 10K+ nodes, 20M traces, 1M+ containers	Detailed resource metrics, dynamic call graphs	Resource scheduling, dependency analysis	URL 2021 URL 2022
Zhang et. al (2022)	Uber	40K endpoints, 4K services, 100k traces	Critical path analysis, aggregated insights	Bottleneck analysis, anomaly detection	URL
Zhang et. al (2025)	Uber	11B RPCs, 1.9 endpoints, 1.5M+ traces	Error resilience, latency impact analysis	Error handling strategies, fault detection	URL Part 1 URL Part 2
Haoran et. al (2020)	Synthetic Testbed	Simulated workloads	Simplified e-commerce and social networking traces	Scheduling algorithms, fault injection	URL
Steidl, M. (2022)	Synthetic Testbed	Simulated workloads	Focused on user sessions and service interactions	Service orchestration, fault tolerance	URL

Conclusion

Distributed tracing datasets are critical for microservices research. The real-world distributed trace datasets from companies like Meta, Alibaba, and Netflix offer a glims into the challenges and opportunities within large-scale microservices architecture. Microservices testbeds offer alternative to collect distributed traces with limited coverage yet more flexibility. Leveraging both types of datasets ensures that researchers can address diverse challenges, from surfacing performance bottlenecks to modelling traces as service dependent and call graphs to designing caching at call chain level (see [10], [11]). Expanding the coverage and access to these datasets will be important to solve various challenges related to microservices architecture.

References

[1] D. Huye, Y. Shkuro, and R. R. Sambasivan, "Lifting the veil on Meta's microservice architecture," 2023, *USENIX Association*. Available: https://www.usenix.org/conference/atc23/presentation/huye

[2] A. Tiwari, "Microservice Architecture of Meta," 2024, *Abhishek Tiwari*. doi: 10.59350/7x9hc-t2q45.

[3] A. Tiwari, "Microservice Architecture of Alibaba," 2024, *Abhishek Tiwari*. doi: 10.59350/4fttv-pt832.

[4] S. Luo *et al.*, "An In-Depth Study of Microservice Call Graph and Runtime Performance," 2022, *IEEE*. doi: 10.1109/TPDS.2022.3174631.

[5] A. Tiwari, "Power Laws and Heavy-Tail Distributions in Hyperscale Microservices Architectures," 2024, *Abhishek Tiwari*. doi: 10.59350/fzpx8-8ps18.

[6] Z. Zhang, M. K. Ramanathan, P. Raj, A. Parwal, T. Sherwood, and M. Chabbi, "CRISP: Critical Path Analysis of Large-Scale Microservice Architectures," 2023, *USENIX Association*. Available: https://www.usenix.org/conference/atc22/presentation/zhang-zhizhou

[7] I.-T. A. Lee, Z. Zhang, A. Parwal, and M. Chabbi, "The Tale of Errors in Microservices," 2024, *Association for Computing Machinery*. doi: 10.1145/3700436.

[8] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "Pre-processed Tracing Data for Popular Microservice Benchmarks," 2020, *University of Illinois at Urbana-Champaign*. doi: 10.13012/B2IDB-6738796_V1.

[9] M. Steidl, "Anomalies in Microservice Architecture (train-ticket) based on version configurations," 2022, *Zenodo*. doi: 10.5281/zenodo.6979726.

[10] A. Tiwari, "Unveiling Graph Structures in Microservices: Service Dependency Graph, Call Graph, and Causal Graph," 2024, *Abhishek Tiwari*. doi: 10.59350/hkjz0-7fb09.

[11] A. Tiwari, “Cache Me If You Can: Taming the Caching Complexity of Microservice Call Graphs,” 2024, *Abhishek Tiwari*. doi: 10.59350/cq9aw-a9821.