Towards using LLMs for Distributed Trace Comparison Abstract

Vaastav Anand
MPI-SWS
Germany

Pedro Las-Casas
Microsoft
USA

Rodrigo Fonseca
Microsoft
USA

Antoine Kaufmann
MPI-SWS
Germany

Motivation

Troubleshooting issues in modern cloud systems is a tedious task. To make this task easier for developers, researchers and practitioners have developed observability tools to capture relevant information about the system execution that can aid in triaging issues.

Distributed tracing [5] is an observability technique that tracks the execution of a single request across the various components of the system. The trace captures both the causal structure of the request execution across the various components as well as timing information about the execution. Thus, these traces are rich in both structural and temporal information.

To investigate incidents, operators often need to compare the structural and temporal properties of a trace representing an erroneous execution with a reference trace representing a successful execution. Pair-wise comparison of traces is challenging as the traces are large and multi-dimensional. Visual comparison tools such as Jaeger Compare [3] are often unintuitive as they fail in presenting an easily digestible visual representation of the difference between the two traces. This issue is further exacerbated if the differences between the traces are subtle and not easily visualizable.

Thus, there is a dire need for tooling support for operators to easily compare traces.

Trace Comparison with LLMs

In this abstract, we propose the use of LLMs for pairwise comparison of distributed traces with our novel tool, Para*ll*ax.

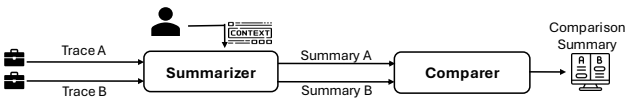

Figure 1: **Para*ll*ax Design**

System Design. Figure 1 shows the design of our LLM-based comparison tool called Para*ll*ax. Para*ll*ax is comprised of two different types of LLM agents - Summarizer (S) and Comparer (C). S takes as input the two traces that need to be compared, as well as optional context provided by the operator to focus the summarization on a specific issue. For each trace, S first produces a trace summary. C then takes the two generated summaries as input generates the final comparison output between the two traces.

Our current implementation of Para*ll*ax operates on raw traces in json format without performing any processing. The agents use `gpt-4o` model with a context window of `128k` tokens.

Preliminary Results. To show the efficacy of Para*ll*ax, we compare two traces from an open source trace dataset [2] of the social network application from the DeathStarBench suite [4]. Specifically, we select one error-free trace and one trace with timeout issues, generated from the `ComposePost` API. As shown in Figure 2, Para*ll*ax correctly identifies the differences between the two traces and provide additional context for the trace containing the error.

> Execution 1 provides a more comprehensive view of inter–service communication, including detailed steps for each service and mention of the database interaction in the `PostStorageService`.
> Execution 2 focuses more on the initialization and attempts of each service, and the recurring theme of client pool problems. Execution 2 highlights more critical client pool contention and timeout problems affecting the services, while Execution 1 reports smooth client pool operations.

Figure 2: **Excerpt from the generated comparison summary**

Challenges. While the early results with Para*ll*ax are promising, there are still pending challenges we need to overcome.

Challenge 1: Using Temporal Information. Current LLMs are notorious for poor numerical reasoning, which is crucial for comparing temporal information such as execution time of different tasks in a trace. This must be accurate to prevent operators from gaining an incorrect insight. Overcoming this challenge requires the Para*ll*ax agents to be robust, or the use of complementary tools. Spectroscope [6], for example, is good precisely at comparing timings of isomorphic traces, whereas LLMs can excel in topological and semantic comparisons.

Challenge 2: Large Traces. Traces can be arbitrarily large and as a consequence these traces may not completely fit in the context window of the LLMs. This is a challenge as complete traces are required for accurate comparison. Until LLMs with larger context windows such as that of Mnemosyne [1] become publicly available, we may need an additional processing step to extract information regarding important features.

Challenge 3: Comparison with Aggregates. Often, operators need to not only compare an error trace with a single reference trace but with a set of reference traces. Using LLMs to summarize a set of traces and then compare it with a bad trace represents an exciting opportunity that could fundamentally reduce the amount of effort required by operators to compare traces.

References

[1] A. Agrawal, J. Chen, Í. Goiri, R. Ramjee, C. Zhang, A. Tumanov, and E. Choukse. Mnemosyne: Parallelization strategies for efficiently serving multi-million context length llm inference requests without approximations. *arXiv preprint arXiv:2409.17264*, 2024.

[2] V. Anand and J. Mace. X-Trace trace dataset for DeathStarBench. Retrieved October 2019 from https://gitlab.mpi-sws.org/cld/trace-datasets/deathstarbench_traces/tree/master/socialNetwork, 2019.

[3] J. Farro. Trace comparisons arrive in jaeger 1.7. Retrieved November 2024 from https://medium.com/jaegertracing/trace-comparisons-arrive-in-jaeger-1-7-a97ad5e2d05d, 2018.

[4] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, et al. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems. In *24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*.

[5] A. Parker, D. Spoonhower, J. Mace, B. Sigelman, and R. Isaacs. *Distributed tracing in practice: Instrumenting, analyzing, and debugging microservices*. O'Reilly Media, 2020.

[6] R. R. Sambasivan, A. X. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing performance changes by comparing request flows. In *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 43–56. USENIX Association, Mar. 2011.