Project Proposal for Distributed Systems
Muhammad Haseeb
mh6218@nyu.edu

TraceZip: A Compression Technique For Distributed Tracing

Motivation:

Due to the growing complexity of web services, application developers are switching towards an architecture that decomposes large services into smaller components that are developed and deployed separately. These smaller components, referred to as microservices, communicate with each other for serving the user requests. Microservices use Remote Procedure Calls (RPC)[1] for communication and the history of these RPCs constitutes microservices' traces. These traces are very helpful in debugging such applications (referred to in the literature as distributed tracing)[2].

Due to the huge volume of the traces, storing them is impractical in terms of storage space and network cost. So the volume of traces that are stored is reduced. One widely used method for achieving this is called sampling. However, reducing the data stored leads to information loss which in turn affects the debugging performance. My goal is to keep all the traces without sampling, yet still manage to maintain reasonable storage costs. This way there will be no information loss and debugging accuracy will not be compromised.

Proposed Work:

This project's aim is to eliminate sampling by efficiently compressing the traces before storing them in any database. The main contribution of my work would be to present a compression technique developed specifically for taking advantage of the structural properties of traces. And the technique will also be analyzed using real traces generated through a microservices' demo application, Online Boutique[3].

The traces can be represented in the form of a graph where each node represents a service and the edges symbolize the communication across the nodes. And every node stores some data e.g., processing time, timestamps, latency, and other metadata. Usually, these graphs differ from each other for some portions while some other portions are very much alike. So the similar subgraphs do not need to be stored separately for each graph. By storing them only once (*the compression*), we can potentially save a lot of storage space.

Evaluation Plan:

The compression technique will be applied to real traces generated through Online Boutique[3] before storing them in a SQL database. Another instance of a SQL database will also be used as a control group which will have the original non-compressed traces. The two databases will then be compared to illuminate any potential storage gains of the compression. If time permits, other typical compression methods (*lz4, std*) will also be used as a comparative analysis.

Division of Labor:

I am doing this project solo so I will be doing all of the proposed work.

Related Work:

Almost all of the related literature has focused on developing a smart sampling technique so that only a subset of the traces is needed to be stored. These sampling techniques can be categorized into two general classes: (i) head-based sampling and, (ii) tail-based sampling. In head-based sampling, the trace is stored or ignored before it has been completed. In tail-based sampling, a trace is first completed, and then it is sampled. Canopy[5], Dapper[4], and Jaeger[6] are examples of the former. Lightstep[7] and some parts of Canopy employ different flavors of the latter.

Another approach is query-based distributed tracing. This includes Pivot Tracing[8] and Snicket[9]. They do not store the traces as they run the queries in real-time in the microservices and only store the results of the pre-defined queries. The main disadvantage is that they limit the historical visibility into the microservices state when it comes to debugging. TraceZip can also be used as an extension of these works for improving the state visibility while maintaining the storage costs at a reasonable level.

References:

1. Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In ACM ASPLOS
2. Rodrigo Fonseca, George Porter, Randy H. Katz, and Scott Shenker. 2007. X-Trace: A Pervasive Network Tracing Framework. In USENIX NSDI
3. https://github.com/GoogleCloudPlatform/microservices-demo
4. Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical Report. Google, Inc
5. Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, Vinod Vekataraman, Kaushik Veeraraghavan, and Yee Jiun Song. 2017. Canopy: An End-to-End Performance Tracing And Analysis System. In ACM SOSP
6. Pavol Loffay. 2020. Data analytics with Jaeger aka Traces Tell Us More! https://medium.com/jaegertracing/ data-analytics-with-jaeger-aka-traces-tell-us-more-973669e6f848. Accessed: 2021-06-25
7. Robin Whitmore. 2021. How Lightstep Works. https://docs.lightstep. com/docs/how-lightstep-works. Accessed: 2021-06-25.
8. Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. 2015. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. In ACM SOSP
9. https://conferences.sigcomm.org/hotnets/2021/accepted.html#:~:text=Snicket%3A%20Query-Driven%20Distributed%20Tracing