Poster Paper: Efficient Navigation of Cloud Performance with ’nuffTrace

S. Qasim*, M. Toslali†, Q. Clark*, S. Parthasarathy†, F. Oliveira†, A. Liu*, G. Stringhini* and A. K. Coskun*
*Boston University, †IBM Research

Abstract—**Distributed tracing has become an essential tool to navigate performance of complex, distributed cloud-native applications, providing a comprehensive view of a request from end-to-end. However, the sheer amount of data generated by distributed tracing can be overwhelming, making it difficult to store, process, and extract meaningful insights. This paper presents the vision of ’nuffTrace that embodies a novel tree-based probabilistic data structure that summarizes trace data in a compact form without storing all of the data, enabling developers to analyze cloud application performance with high accuracy and efficiency.**

I. Introduction

Distributed cloud applications are susceptible to performance fluctuations caused by various factors, such as performance bugs or faulty code launches. Such fluctuations can lead to extended recovery times in the cloud as problems spread across interdependent services. Delays in debugging problems result in costly downtime and reduced user satisfaction [1]–[3].

With the large and complex nature of cloud application architectures, it can be challenging to understand how a single request is processed. This makes it difficult to locate the source of unexpected slowdowns in request latency, particularly in the event of a performance issue.

Distributed tracing has emerged to address this problem by offering a means to track a request’s journey as it travels from one service to another in the cloud. Distributed tracing uses unique trace IDs that are propagated through the system as a service calls further services to process a single request, allowing for a complete end-to-end view of a request.

A trace consists of detailed timing information about how a single user request traverses the system, recording *spans* that represent the starting/ending time and causal relationships of operations being performed. Extensive research has demonstrated the efficacy of distributed tracing as a powerful tool for debugging [4]–[7]. By offering end-to-end visibility into requests, distributed tracing assists in identifying performance bottlenecks, conducting performance analysis, and troubleshooting in distributed cloud applications.

While distributed tracing offers numerous advantages, the amount of data generated is often large and overwhelming, making it difficult to store, process, and extract meaningful insights. For example, Netflix [8] comprises hundreds of microservices that interact with each other to offer users a seamless and highly available streaming experience. To guarantee the platform’s reliability and performance, Netflix employs distributed tracing to monitor and analyze the flow of requests and responses between services. In just one hour of operation, the system generates tens of millions of trace events, including information such as timestamps, trace IDs, and more. Consequently, storing and processing GB/s trace data poses a significant challenge, necessitating substantial storage capacity and processing power. By the same token, analyzing this data in a meaningful way is difficult, necessitating advanced data analysis tools and techniques to extract insights.

There is an emerging need for a resource and computationally efficient system to handle the vast amounts of tracing data and help extract meaningful insights. We claim that such a system can be accomplished by approximating trace statistics from a compressed representation of data. To realize this vision, we are developing a prototype system called ’nuffTrace that stores trace summaries in a resource-efficient manner while providing developers with substantially accelerated and highly accurate performance analysis of cloud applications. This paper provides promising results on a prototype implementation of ’nuffTrace.

II. The ’nuffTrace Approach

The end-to-end view of ’nuffTrace is depicted in Figure 1. An essential feature of ’nuffTrace is a tree-based probabilistic data structure that organizes and summarizes large trace data in a compact form without having to store all the data. Probabilistic data structures summarize information efficiently, while the tree-based organization captures the hierarchical nature of tracing, including user requests, services, and operations within the services.

’nuffTrace processes incoming traces and populates summaries of end-to-end request, service, and operation durations in our tree-based data structure. Rather than storing all individual observations, ’nuffTrace retains only enough information to maintain approximate representative values with a small relative error tolerance. This lightweight data structure, which only occupies kilobytes of storage for thousands of traces (as shown in our evaluation), allows ’nuffTrace to efficiently surface aggregate summaries (e.g., various percentiles of a request, service, or operation latency) and provide valuable insights (e.g., operations that contribute to request slow-downs) that enable developers to efficiently and swiftly navigate performance, and even help locate the source of performance issues in cloud applications.

We use probabilistic data structures to approximate quantiles. In a traditional solution, computing the exact percentile value of a stream of values requires retaining and sorting them. Instead, we propose a novel data structure that stores representative values that are close enough to one another. This enables us to map any observed latency value to one of the representatives with only a small loss in accuracy. To achieve this, we bucketize the range of latency values and choose the middle values of the buckets as representatives.

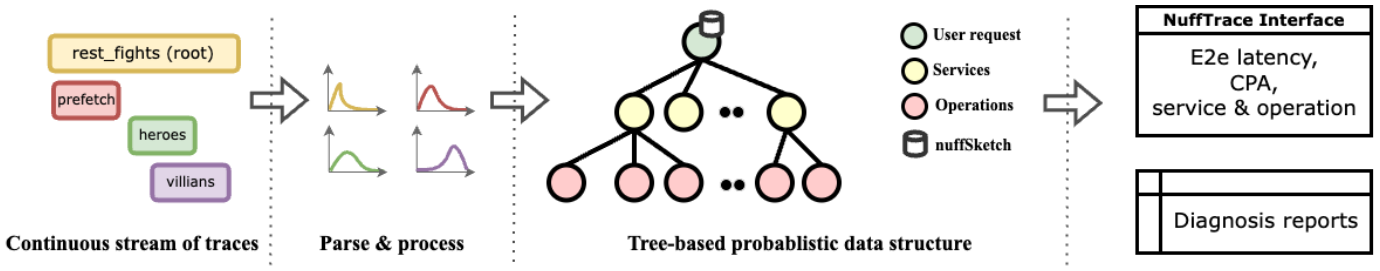
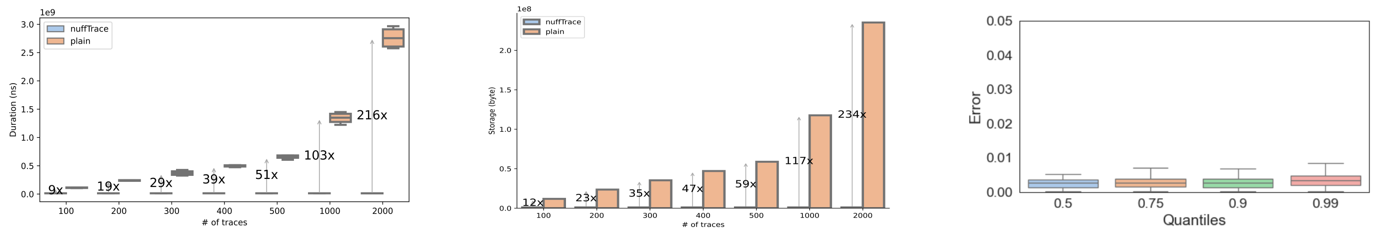


Fig. 1: End-to-end view of ’nuffTrace .



(a) Query durations for Train ticket. (b) Memory footprints for Train ticket. (c) Quantile error rates for Train ticket.

Fig. 2: ’nuffTrace vs. Plain approach comparison for Train Ticket application. The figure presents durations for computing various quantiles (of requests, services, operations) with ’nuffTrace vs. Plain, memory footprints of ’nuffTrace and Plain, and approximation error rate of ’nuffTrace.

Our data structure is a combination of α-accurate q-quantile sketch [9] and count-min sketch [10]. The α-accurate q-quantile sketch divides the range of latency values into buckets, with each bucket counting the number of values that fall between $(\gamma^{i-1}, \gamma^{i}]$, where $\gamma = (1+\alpha)/(1-\alpha)$. Given a latency value of x, it is assigned to the bucket indexed by $\lceil \log_{\gamma} x \rceil$ with the bucket width growing exponentially. we extract the span latency observations and self-execution times of spans (which captures latency propagation by subtracting child execution times from the parent span), to populate the ’nuffTrace data structure.

III. Experimental results

Figure 2 compares the query duration, accuracy (by measuring error rates of approximated quantiles), and memory usage of two methods: ’nuffTrace and a *Plain* approach that uses raw traces to calculate the precise quantiles on Train Ticket [11] application traces. The Plain approach is used by systems such as tprof [1], which hierarchically groups traces by request types and trace structures, and calculates increasingly detailed aggregated statistics. Our method demonstrates faster query durations compared to the Plain approach. This is because we iterate a fixed number of buckets of quantile sketch, retrieve corresponding counts from count-min sketch, and approximate quantiles, while Plain needs to iterate all traces, extract statistics, and sort to calculate the exact percentiles. Our computational efficiency gains become more significant as the number of traces increases (216x for 2000 traces). Compared to Plain aggregation of traces, NuffTrace requires much less memory, usually only a few kilobytes, while the Plain traces can consume several megabytes (or even gigabytes/terabytes in production systems). The magnitude of our savings becomes more significant as the number of traces increases (234x for 2000 traces). We measure the loss of information by calculating error rates for quantile estimation of span durations. Our approximations allow for a small error that is typically below 1%, which further justify the computational and resource savings we achieve.

References

[1] L. Huang and T. Zhu, “Tprof: Performance profiling via structural aggregation and automated analysis of distributed systems traces,” in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC ’21, p. 76–91, 2021.
[2] D. H. O’Dell, “The Debugging Mindset,” *ACM Queue*, vol. 15, p. 50, Feb. 2017.
[3] Y. Evinav, “Marissa mayer at web 2.0.,” Nov 2006.
[4] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, “Dapper, a large-scale distributed systems tracing infrastructure,” 2010.
[5] “Jaeger, open source, end-to-end distributed tracing.” https://www.jaegertracing.io/.
[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 *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.
[7] M. Toslali, E. Ates, A. Ellis, Z. Zhang, D. Huye, L. Liu, S. Puterman, A. K. Coskun, and R. R. Sambasivan, “Automating instrumentation choices for performance problems in distributed applications with VAIF,” in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC ’21, p. 61–75, 2021.
[8] P. Maulik, “Building netflix’s distributed tracing infrastructure.,” Oct 2020.
[9] C. Masson, J. E. Rim, and H. K. Lee, “Ddsketch: A fast and fully-mergeable quantile sketch with relative-error guarantees,” *Proc. VLDB Endow.*, vol. 12, p. 2195–2205, aug 2019.
[10] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
[11] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, “Latent error prediction and fault localization for microservice applications by learning from system trace logs,” in *Proc. of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE* (M. Dumas, D. Pfahl, S. Apel, and A. Russo, eds.), pp. 683–694, ACM, 2019.