Accelerating Distributed Tracing in Serverless Computing Using Differential Tracing

Yuhan Yang
Shanghai Jiao Tong University

Xingda Wei
Shanghai Jiao Tong University

***Abstract*—Distributed tracing is a fundamental tool for analyzing the execution of applications that involve multiple devices, such as AIoT systems. Meanwhile, serverless computing is a promising edge-cloud cooperation scheme that offers improved resource and cost efficiency. However, the efficiency of distributed tracing in serverless computing is compromised due to the excessive traces generated by various system components. In this paper, we introduce Differential Tracing, an efficient and accurate distributed tracing system specifically designed for serverless computing, which minimizes overhead of tracing while maintaining high accuracy.**

***Index Terms*—Distributed Tracing, Serverless, Performance Monitoring**

I. Introduction

Performance debugging in distributed systems can be challenging, especially nowadays as these systems are becoming increasingly complex. To address this challenge, *distributed tracing* has been widely adopted to help developers understand the behavior of the systems. Distributed tracing obtains the causal relationships between different components of the system and the execution time of each component. Given its debugging and performance analysis capabilities, distributed tracing is widely used in AIoT systems as well.

While traditional distributed tracing systems(e.g., Zipkin and Jaeger [1], [2]) work well in microservices architectures, they encounter new challenges in serverless scenarios.

First, compared to microservices, serverless introduces platform components, such as scheduler and load balancer, during the request handling process, thereby introducing more potential anomalies and performance bottlenecks.

Second, to monitor the status of serverless platform components, it is necessary to trace the components as well, which generates more trace data. Reporting this extensive volume of trace data to the backend could result in data loss, thereby leading to an inaccurate analysis of the system's behavior.

This work proposes a system that offers efficient and accurate distributed tracing for serverless. Our system's design is grounded in the observation that, in a serverless environment, not all spans need to be reported, and a significant number can be omitted. This enables us to decrease the volume of meaningless spans reported to the backend, thereby enhancing the accuracy of the trace.

II. Background and Motivation

Distributed tracing and full tracing. Tracing is a key pillar for analyzing the behavior of how a request is handled by a distributed system comprised of multiple components. It records the lifecycle of how a component executes a sub-part of the request, termed *span*, which includes information such as the start and end times. It also tracks the causal relationship between these components, i.e., how one component triggers others. The spans, along with their relationships, comprises the *trace* of the request.

While enhancing observability, recording traces inevitably adds overhead to the requests. A key challenge in distributed tracing is how to achieve *full tracing*—record all traces of requests—with minimal overhead.

Serverless computing. Serverless is an emerging cloud computing paradigm that allows developers to deploy components elastically to handle requests, which is widely adopted by major cloud vendors [3]–[6]. In serverless, developers only need to specify a *workflow* to handle a request. This workflow, typically described as a Directed Acyclic Graph (DAG), describes how components coordinate to handle the request. In the DAG, nodes represent components, and edges signify dependencies among them. For instance, in the case of aßb, b is triggered after a.

A unique feature of serverless over traditional request handling paradigms like microservice is the on-demand allocation of components based on workloads. Specifically, if a component is needed, the cloud platform will allocate a server, start a container (e.g., Docker), and execute the component's logic within the container.

Challenge of full tracing in Serverless. Full tracing is challenging in the new setup because the number of traces for each request is significantly larger than in microservices. To show this, we conduct an initial experiment using workloads from the Alibaba trace [7] and deploy them on a popular open-source serverless platform—Knative [8]. Our results illustrate that full tracing on Knative requires recording traces 29$\times$ larger than when deployed as microservices. The key reason is that each component is executed elastically, involving interaction with many serverless platform components, such as activator. These components are not needed in microservices.

III. Overview

Design goals. Our objective is to achieve full tracing with minimal to no observable overhead for serverless applications. From a systematic perspective, there are two optimization directions: (1) reducing the number of spans needed without

979-8-3503-6338-8/24/$31.00 ©2024 IEEE
DOI 10.1109/CSCAIoT62585.2024.00007

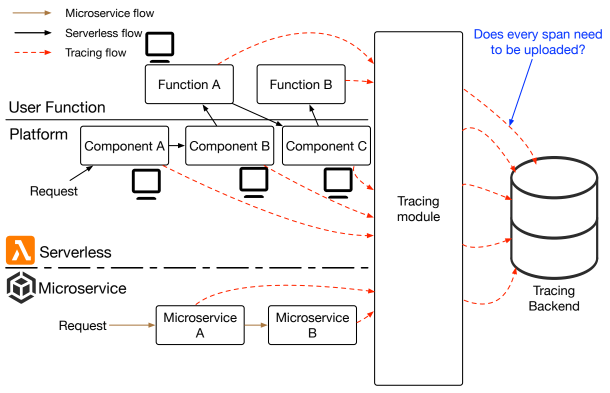

Fig. 1. Distributed Tracing under Serverless and Microservice

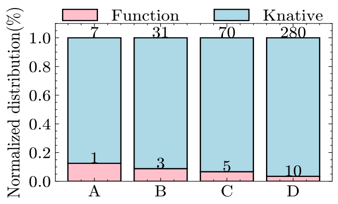

Fig. 2. Span Distribution under Knative

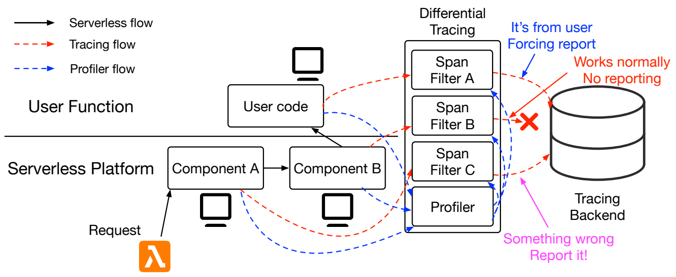

Fig. 3. Differential Tracing Architecture

compromising accuracy and (2) minimizing the overhead of recording and reporting each trace.

Opportunity in serverless. Our key observation is that in a serverless environment, not all spans need to be recorded and reported. Specifically, if we know a span is correct, meaning it is not involved in any exceptions or performance anomalies, we can abandon it [9].

Determining which span should be abandoned is challenging in traditional microservices, because the logic of the component being executed (user-code) is unknown. But it is possible in serverless because most spans are generated by system components—those managed by the platform, as shown in Figure 1. To confirm this, we further selected four workflows **{A, B, C, D}** from production trace [7] and examine where the spans are generated on knative. Figure 2 indicates that the spans generated by the platform components constitute a significant proportion of the total spans (96.6% at workflow **D**).

Challenges and solutions. To achieve the above optimization, two challenges should be addressed:

- How to keep the accuracy of tracing while reducing the number of spans.
- How to accelerate the speed of reporting spans to the backend.

Differential tracing addresses these challenges as follows. First, to maintain the accuracy of tracing while abandoning spans, we will leverage an off-line profiler to profile the workload so as to reconstruct the traces if necessary. We encode essential information about the span (e.g., source/destination) into the request header. This approach allows the span to propagate with the request, enabling us to aggregate them when the request reaches the user function. Then we employ a profiler to reconstruct the spans and report them to the backend when needed.

Second, to expedite the process of reporting spans, we leverage the capabilities of new data center hardware such as SmartNIC. With tne DMA communication between DPU and Host to transfer span data to the DPU's memory without engaging the host's CPU (e.g. using NVIDIA DOCA DMA SDK [10]), and by utilizing RDMA for rapid reporting of spans to the backend, we significantly speed up the reporting process.

IV. System Design

Figure 3 shows the core components of realizing Differential Tracing. First, like conventional distributed tracing systems, it collects spans from both user functions and platform components. Subsequently, the Span Filter classifies these spans. If a span is generated by user functions, it is automatically added to the report queue. For spans from platform components, the Span Filter applies various rules (e.g., exceptions, queuing, resource contention) to assess the components' normalcy. Anomalies prompt the inclusion of the component's span in the report queue for backend transmission; otherwise, the span is omitted.

Moreover, **Profiler** periodically collects metrics data from both the user functions and platform components. Upon data reception, **Profiler** updates the filter rules to adapt to the current system status. Additionally, **Profiler** provides an approximate duration for the span of the components even if the span is not reported. By doing so, Differential Tracing can provide a full trace for the request when the anomaly occurs. This also enables Differential Tracing to diagnose anomalies in the user function based on the function's average duration.

V. Future work

Another way to enhance tracing accuracy is by accelerating span reporting. We plan to utilize SmartNIC and RDMA for this acceleration. With the memory resources provided by the SmartNIC, we can leverage DMA to transfer the span from the host's memory to the SmartNIC. Subsequently, the spans can be reported to the backend over the RDMA network directly from the smartNIC.

References

[1] Zipkin. Distributed tracing system. [Online]. Available: https://zipkin.io/

[2] Jaeger. open source, distributed tracing platform. [Online]. Available: https://www.jaegertracing.io/

[3] AWS, "Aws lambda," https://aws.amazon.com/lambda, 2022.

[4] Microsoft, "Azure functions," https://azure.microsoft.com/en-us/services/functions/, 2022.

[5] Huawei, "Huawei clound functions," https://developer.huawei.com/consumer/en/agconnect/cloud-function/, 2022.

[6] A. cloud, "Alibaba serverless application engine," https://www.aliyun.com/product/aliware/sae, 2022.

[7] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, C. Curino, G. Koutrika, and R. Netravali, Eds. ACM, 2021, pp. 412–426. [Online]. Available: https://doi.org/10.1145/3472883.3487003

[8] Knative, https://knative.dev, 2022.

[9] L. Zhang, Z. Xie, V. Anand, Y. Vigfusson, and J. Mace, "The benefit of hindsight: Tracing edge-cases in distributed systems," in *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, M. Balakrishnan and M. Ghobadi, Eds. USENIX Association, 2023, pp. 321–339. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/zhang-lei

[10] Nvidia doca software framework. [Online]. Available: https://developer.nvidia.com/networking/doca