Alternative Visualisation of Distributed Tracing data in a complex, large-scale distributed system

Noah Santschi-Cooney - 116361061

March 3, 2020

1 Introduction

Modern Internet services are often implemented as complex, large-scale distributed systems. These applications are constructed from collections of software modules that could span many thousands of machines across multiple physical facilities. With the rise of modern Micro-service and Service-Oriented designs, traditional tooling used to monitor application behaviour is no longer viable, especially at scale. To understanding the flow and lifecycle of a unit of work performed in multiple pieces across various components in a distributed system, the concept of Distributed Tracing was born.

Distributed Tracing was first introduced to the mainstream world in 2010 after the publication of Google's Dapper paper. Since then, various standards have evolved and numerous vendors have come out with their own Dapper-inspired services, most of them utilising visualizations such as flame or timeline graphs. This final year project aims to explore ways of leveraging modern distributed tracing standards to create novel ways of consuming the outputs of instrumented applications.

2 Distributed Tracing

The concept of distributed tracing has existed for over a decade at the time of writing. This section will provide a brief history and overview of the main concepts and implementations of distributed tracing, from the first published paper of the implementation at Google to modern day standards.

2.1 Dapper

Released in April 2010, Google published a paper describing the design decisions behind an in-house implementation of distributed tracing, named Dapper. It is commonly believed that this paper describes the common ancestor to many tools that implement a form of distributed tracing.

The Dapper paper introduces some of the core primitives that underpin modern day standards. Most notable are the concepts of a directed acyclic graph (DAG) called a *trace tree* and its nodes, which are referred to as *spans*. The trace tree forms a relationship between spans, not unakin to a tree of stack frames that may be generated by gathering stack frames over time, albeit generally at a much higher level than at the level of individual subroutine calls.

Figure 1 illustrates a trace tree with five spans. Each span is shown to contain 3 specific pieces of metadata alongside the start and end timestamps necessarily to reconstruct the temporal relationships: a human-readable *span name*, an integer *span ID* and an integer *parent ID*. The latter two data points are used to reconstruct the relationship between individual spans. A span without a parent ID becomes the *root span* of a trace tree. Not shown is another important but, as of right now, not relevant piece of metadata, the *trace ID*, which is common amongst all spans within a single trace tree.
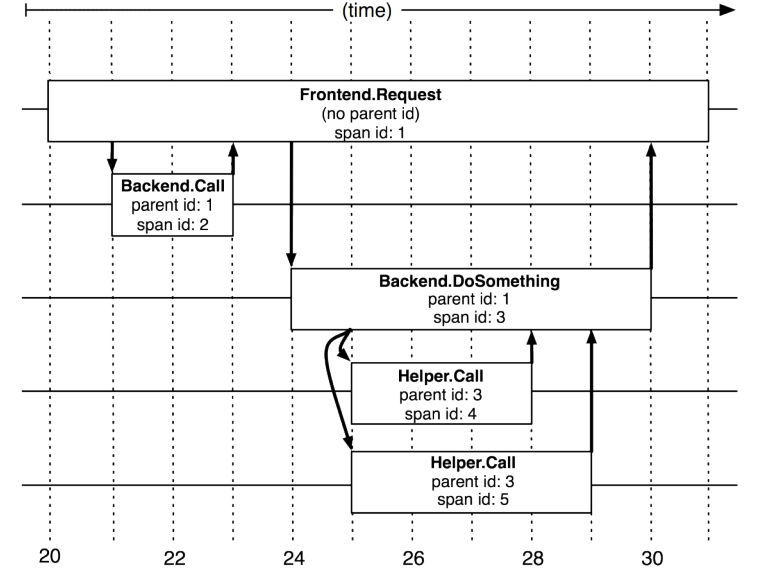
Figure 1: The relationships between traces in a trace tree

As described thus far, Dapper trace trees allow for a detailed view of the relationships of distributed systems within Google. When using this data for debugging or performance analysis, it can often be convenient or even necessary to have additional context surrounding a trace tree or its individual spans. As such, the paper describes a simple API through which application developers can provide a combination of two types of annotations: timestamped textual annotations and key-value, allowing for defining arbitrary equivalence classes between traces which can be operated upon in the analysis tools.

2.2 OpenTracing

OpenTracing project's inception came about in October 2015, since which time it has become a project under the Cloud Native Computing Foundation in 2016, created to standardize a set of vendor neutral and programming language agnostic application programming interfaces (APIs) for instrumenting code for distributed tracing. Heavily inspired by the Dapper paper, it borrows many of the nouns and verbs outlined in the Dapper paper, including *traces* and *spans*. Dapper's timestamped annotations are referred to as *logs* in the OpenTracing specification, while the key-value pairs are named *tags*.

The OpenTracing API also specifies how a trace cross process boundaries, so that spans created in different processes can be associated with a common trace tree. This was named the *span context* and at it's most basic level contains the overlying trace ID as well as the current span ID. With this, new spans

generated across process boundaries will be able to specify their parent span as well as their common trace, without propagating an entire span, which may prove costly as more tags and logs are attached to a span.

3 Visualizations of Distributed Tracing Data

All this telemetry data would be of little use if it wasn't consumed in some manner. As distributed tracing has only become a more prominent topic in the industry in the last half decade, the set of visualizations that exist and are commonly employed is still rather small in numer. In this section, we will discuss some of the most common forms of visualizations, leading up to the methods explored in this project.

3.1 Flame Graph

By far the most commonly adopted visualization is a style of graph closely modeled after the *flame graph* style, closely resembling Figure 1. An example of such can be seen in Figure 2. Each entry in the graph corresponds to a span in the overall trace tree. They can be expanded to provide the logs and tags associated with a given span. The value that can be derived from this visualization has resulted in large adoption of the flame graph style in distributed tracing providers and platforms, leading many to consider it the de-facto visualization option for tracing data.
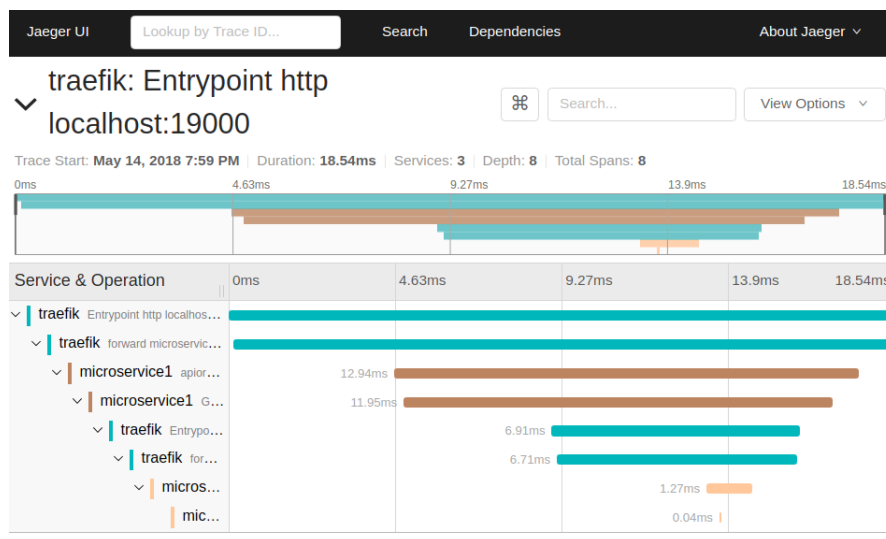

Figure 2: Example of the visualization of tracing data in Uber's Distributed Tracing platform, Jaeger

3.2 Plot Graph

Less commonly found is the traditional *plot graph*, with complete trace duration on the y-axis and time on the x-axis. This is often complementary to an overview of all collected traces, allowing to choose at a glance to inspect certain traces slow durations. This

4 Conclusion