Using OpenTelemetry to keep track of data flow in distributed systems

Elias Leonhardsberger

BACHELOR THESIS (PROPOSAL)

submitted to the
Bachelor's Degree Program
Software Engineering

at the
University of Applied Sciences Upper Austria
in Hagenberg

2025

Supervisor: FH-Prof. DI Johann Heinzelreiter

Contents

Kurzfassung **ii**

Abstract **iii**

Exposé **1**
1 Motivation 1
2 Problem 1
3 Goals 2
4 Methodology 2

References **3**

Kurzfassung

Den Überblick über verteilte Softwaresysteme zu behalten, ist schon schwierig genug, aber wenn große Datenmengen von und zu mehreren Endpunkten transportiert werden, scheitert traditionelles Logging oft daran Fragen über die Datenqualität und Programmzuverlässigkeit zu beantworten.

Traces und Metriken können dabei helfen, diese Probleme zu bewältigen, aber die Programme die diese Daten transferieren, sind oft als unabhängige Hintergrund Prozesse mit unterschiedlichen Ausführungszeiten implementiert, oft sogar auf unabhängigen Maschinen.

Glücklicherweise kann man mit OpenTelemetry Traces über mehrere Maschinen hinweg durch einen Kollektor vereinheitlichen. Dadurch bleibt aber noch das Problem des Umfangs eines Traces.

Der Umfang eines Traces ist normalerweise eine Anfrage oder Jobausführung. Das macht es schwierig den Überblick über einzelne Datensätze zu behalten. Der Fokus dieser Arbeit liegt darauf dieses Problem zu lösen.

Das Ziel ist es, eine Möglichkeit zu schaffen, Traces zu sammeln, zu vereinheitlichen und dann in neue Traces aufzuteilen, die zeigen, wie sich Datenpunkte durch das System bewegen.

Die Arbeit erstreckt sich von der Erstellung von Traces mit semantischen Ereignissen auf der Job Ebene, über die Verbindung von Traces von verschieden Jobs, die an den gleichen Daten arbeiten, dann das Aufteilen und Abtasten der Traces in je einen Trace für jeden Datenpunkt in einem OpenTelemetry Kollektor und schließlich die Generierung von Metriken aus diesen Traces, um ein Gesamtbild des Datentransfers zu erhalten.

Abstract

Keeping track of distributed software systems is hard as it is, but when moving a large amount of data from and to multiple endpoints, traditional logging often fails to answer questions about data quality and program reliability.

Traces and metrics can help to mitigate these problems, but the programs transferring the data are often implemented as independent background jobs using different schedules, often even on independent machines.

Luckily using OpenTelemetry you can unify traces across multiple machines easily with collectors, leaving only the problem of trace scope.

The scope of a trace is normally one request or job execution, which makes keeping track of specific data points hard.

The focus of this thesis is fixing this problem.

The goal is to provide a way to collect, unify and then split traces into new traces showing data points travelling through a system.

This thesis stretches from creating traces with semantic events on the job level, to connecting traces from different jobs working on the same data, then splitting and sampling these traces into multiple traces for each data point in an OpenTelemetry collector and finally generating metrics out of these traces to provide a big picture view of the data transfer.

Exposé

1 Motivation

OpenTelemetry has established itself as a groundbreaking new standard in the world of observability. It is replacing many vendor specific solutions with a single, standardized, extensible, open source and language integrated solution.

OpenTelemetry has been designed with microservices in mind, but it is not exclusive to microservices. Distributed systems of all kinds, even legacy systems, can profit of OpenTelemetry.

2 Problem

A big problem in legacy systems, especially naturally grown ones, is that often no single source of truth exists. So data entries, like employee data, are often stored in multiple loosely connected sub systems. For example the information which employee has which manager could be stored in another subsystem as the information in which team an employee is working. Issues like these are often structural and can not directly and immediately be influenced by the developers tasked with dealing with the system. Unifying the data sources and saving them in a centralized system is a tedious and long undertaking, requiring the whole companies effort to clean up past technical dept and incorrect data.

To keep the company operational while this is and to migrate the system step by step, the data from the legacy sources needs to be synced with the new system, but there are bound to be mismatches, invalid states and other errors, due to no single source of truth existing. These issues can often take a long time to fix, but they should not stop the entire syncing process. The developers and decision makers do however need to be informed when, for how long and why errors are happening, without having to click through every applications logs.

To help getting through this chaos, all the observability data, meaning logs, traces and metrics, should be collected to a single dashboard showing all the issues and the performance of the whole system.

This alone is easily achieved using simple zero-code instrumentation and a telemetry backend of your choice like Jaeger, Zipkin, Prometheus or in this case Microsoft Application Insights. The problem with this solution is that the logs that get collected are very difficult to search through, especially if the amount of data and potential logs

exceed a thousand entries per execution, while traces are almost unused.

3 Goals

The goal of this thesis can be split into two parts, the creation of traces for transferred data points and the collection and regrouping of these traces to represent the flow of data through the system. Creation is handled in the application code, while collection and regrouping is handled by a custom OpenTelemetry collector.

Creation

A trace should be created for each execution of a data transfer job, which contains child spans for every data point in this job. Those child spans contain the status of the transfer, the time, a type and an identification for the data point. The parent trace contains information about source, destination and a flag, indicating that the custom collector should work on this trace.

Collection

OpenTelemetry Traces, Metrics and Logs can be exported either directly to a telemetry backend, or to a proxy, called a collector. This collector gathers telemetry data from different sources, performs operations like batching or filtering and then exports the telemetry data to different backends.

The created traces are taken out of the normally exported data and are then re-grouped based on the data point identification. They are first split into the spans for each data point, the spans are then grouped by their identification and then they get sampled and exported.

Due to the jobs being independent of each other, the telemetry data has to be saved to be able to group jobs happening hours from each other.

4 Methodology

In the scope of this thesis a prototype will be created using C# with .NET, for the base system, and GO, for the collector.

References