Final Project Exploratory Visualization

MIDS W209 Spring 2021
Justin Trobec

Introduction

Our final project is to build visualizations to generate insights from distributed trace data. Distributed tracing is a technique used to instrument the execution of an operation that is distributed over a group of software systems, generally referred to as services. For example, when an internet application makes a request to an API endpoint, that request often is serviced by fanning out various subtasks to other services in a data center. Distributed tracing systems instrument the calls from service to service (also known as Remote Procedure Calls or RPCs), and gather the instrumentation in a central place so that the execution of the operation across the fleet can be analyzed. There are many distributed tracing systems in use today. We chose to use the Zipkin system as it is open-source, actively developed, and relatively widely used. However, most distributed tracing systems use a similar schema to represent data, so whatever approaches to visualization we develop here should be applicable to other systems.

Begin by inspecting the available data without visualizing the data, and write down three hypotheses.

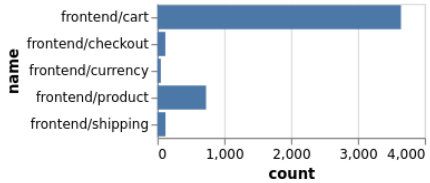
Zipkin and many other distributed trace systems represent traces as trees, whose nodes are referred to as spans. A single span represents and RPC call. Children of the span are other RPC calls generated by the parent in handling the incoming RPC request. Child spans link to their parents via the `parentSpanId` field. See the Zipkin Data Model page for more explanation of the data format. Our trace data was generated using a synthetic load generation tool running against a local Zipkin server. We used a script to download each generated trace into its own `json` file. In total, we produced 4666 traces. The load generation tool randomizes durations for the traces, as well as metadata for specific spans, including error conditions. In theory, since I generated the data, I should be able to predict it's shape, but in this EDA I will proceed as if this is a system I am unfamiliar with. In fact, I only briefly inspected the test topology, and made a single modification, so I am not really 100% sure what to expect. Here are some hypotheses I will investigate:

1. Trace tree structure is not 100% consistent for all traces with the same root.
2. Total duration of a trace is only loosely correlated with the total number of children.
3. For traces with the same root service-endpoint, differences in duration will be related to differences in child duration.

H1: Trace tree structure is not 100% consistent for all traces with the same root.

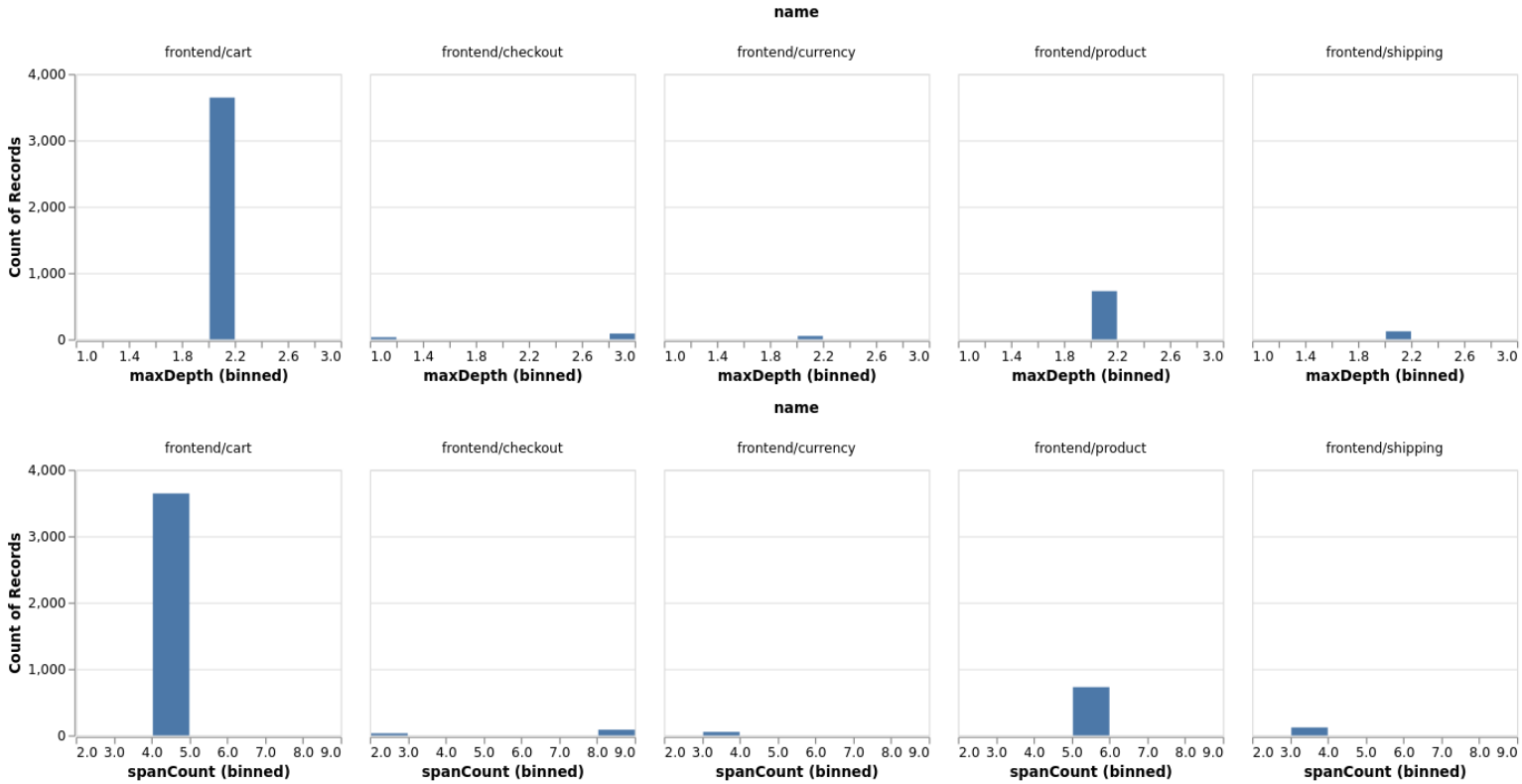
Next, investigate each of your hypotheses by visualizing relevant variables...

To start off, I'd like to get a rough ID of how many requests come into each endpoint. To do that, I will plot the counts of the service/name pair of the root span of each trace:

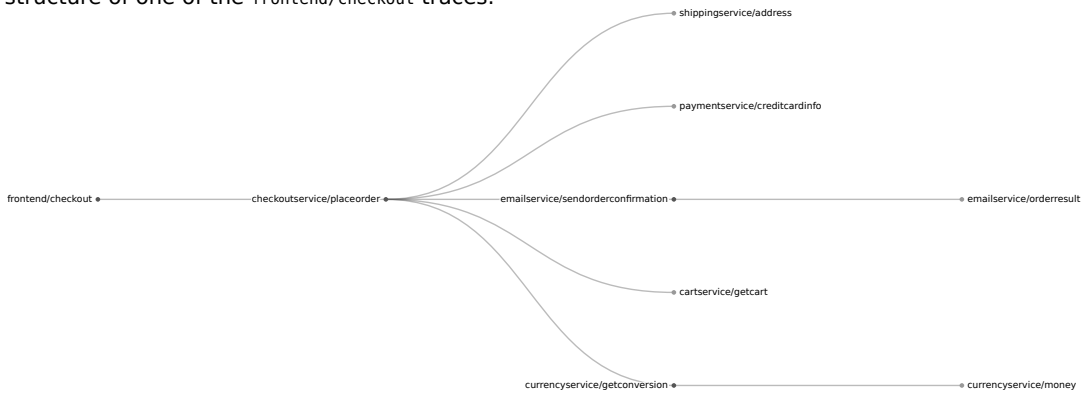


So it looks like the vast majority of traces come into the `frontend/cart` API endpoint. I would guess that

because this is the most common endpoint, we are likely to see the most variation in trace structure here, but that's not a given. Next, I want to look at each endpoint and figure out if the depth of each trace is consistent. We'll also check the distribution of span counts for traces.

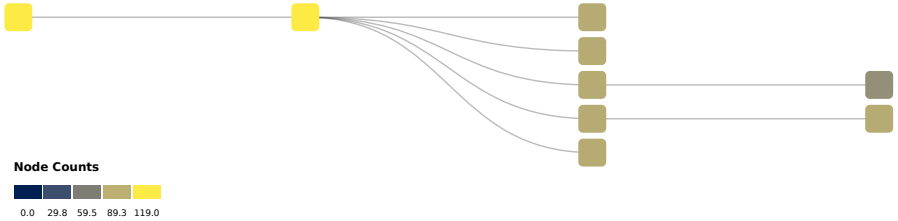


It seems that most of the traces are consistent in terms of their depth and span count, with the exception of `frontend/checkout`. That answers the question of trace structure consistency, in that we see clearly there is some inconsistency, but I'd like to know more about the inconsistency. I'm going to try and show the tree structure of one of the `frontend/checkout` traces.



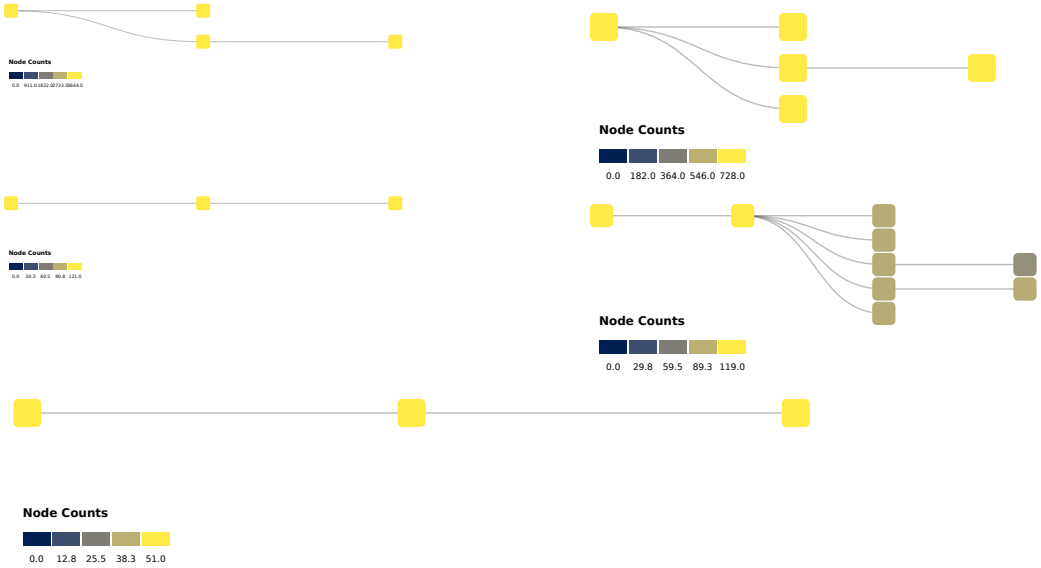
There seem to be two distinct structures for this trace, based on the histograms, so I'll also find an example of one of the 2-span versions.

This is ok, but my concern now is that a user would have to identify each possible trace form and plot them. If there are many traces, this could become really onerous. I think that we can do a more condensed tree format that shows the structure of all traces with a certain entry point (e.g. `frontend/checkout`), and maybe encode relative counts using either node size or color.

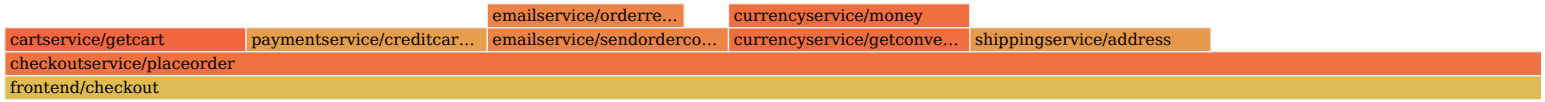


When I was iterating on this visualization, I would slice the number of traces that I used down to 500 instead of the full 4.5k. This allowed me to iterate more quickly since it took much less time to load. After I finished this visualization, I tested with the full set, and found this surprising insight: there are actually three distinct trace structures for this particular API endpoint. Based on the histograms I used earlier to see depth and count of spans, I thought there were only two. But it turns out that one of the deepest nodes is sometimes missing. I didn't see this in the depth count because similar traces are just as deep, and I couldn't see it in the span count histogram because the binning happened to put traces with 8 and 9 spans together. I think this is a pretty good demonstration of why this visualization is a more useful way to compare the structure of the trees than plotting summary statistics.

I can actually shrink this window significantly, and then show all the trace structures present in the data. Many of these are not particularly interesting, since they only really have one structure, but if they had more divergent structures, that would show up in the coloring of the nodes. The nice thing is, because each endpoint has all their traces represented in a single tree, I can get a sense of all the structures and relative counts at once.

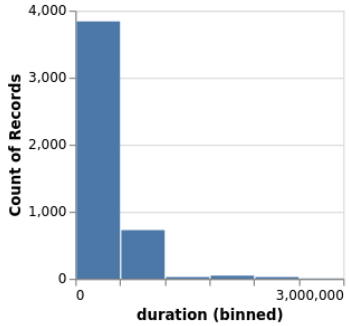


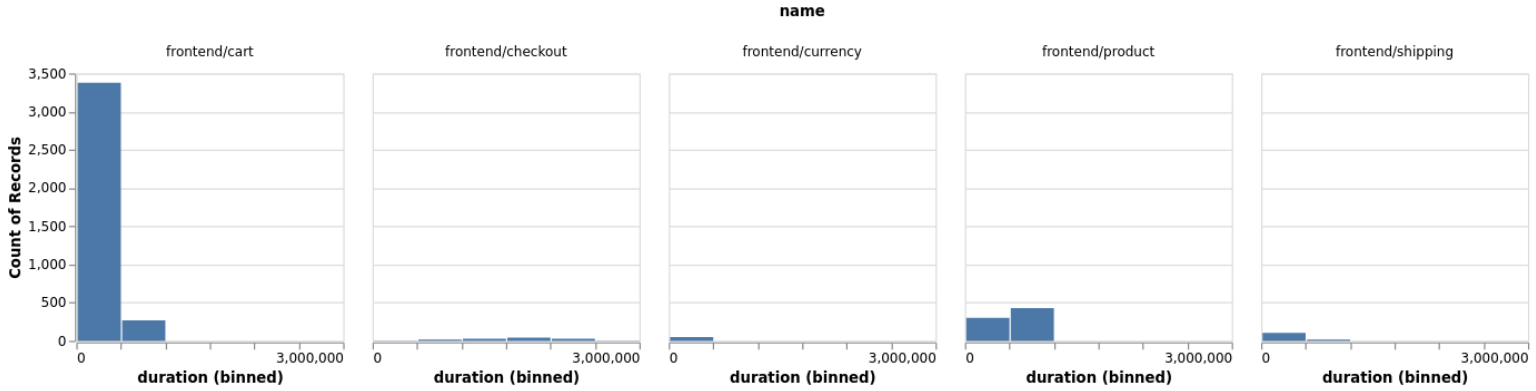
As an alternative, I wanted to try applying a flamegraph, which are somewhat popular in the performance engineering industry. The information is essentially the same; the width of a bar represents the relative percentage of parent calls in which the child appears, divided by the number of children the parent has. This is a complicated thing to articulate with words, but I believe the visual representation ends up being fairly intuitive, especially if you've seen the trace structure graph first. The flame graph allows drilling in on a specific child, so if you click on a node, it expands to fill-up 100% of the width, and its children expand so you can get a closer look at their relative occurrences.

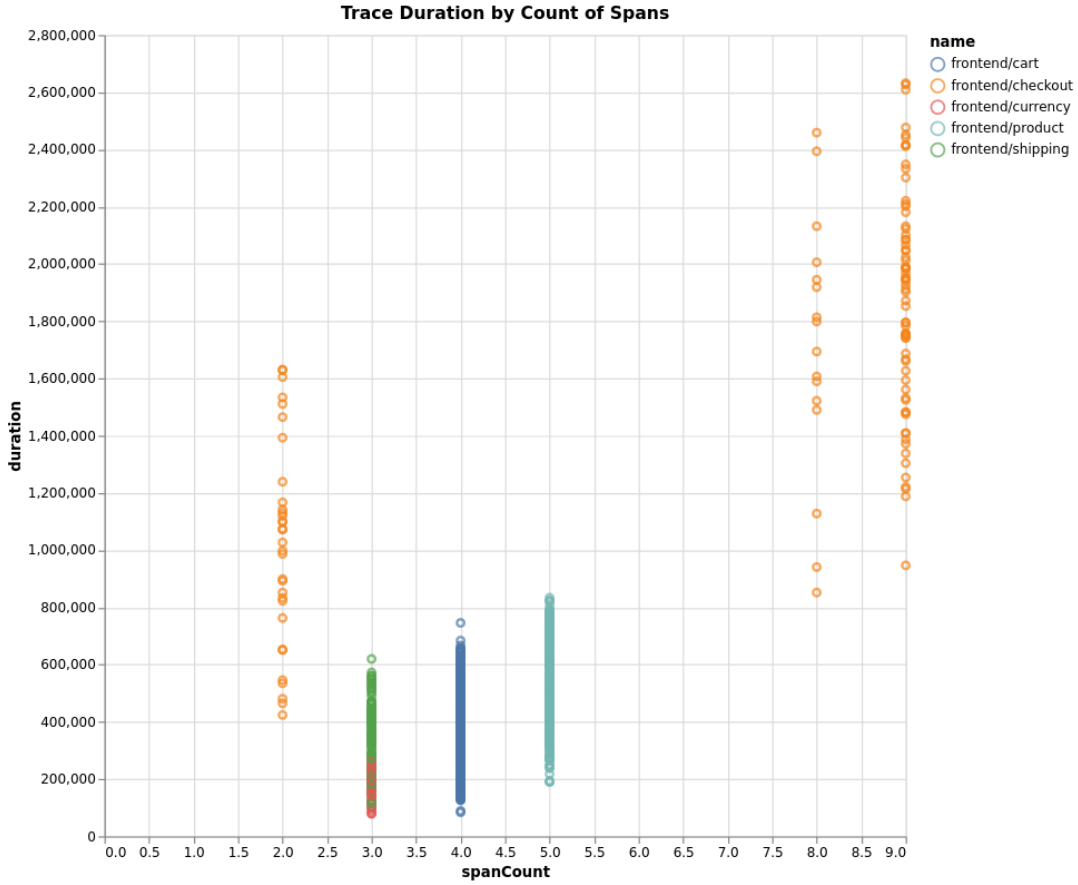


Ultimately, the hypothesis that trace structures are not 100% consistent within the same endpoint is easily validated by either view. It's good to note as well that the histograms I tried actually ended up misleading me about the shape of the traces, which underscores the importance of including the hierarchy as part of the visualization.

H2: Total duration of a trace is only loosely correlated with the total number of children.



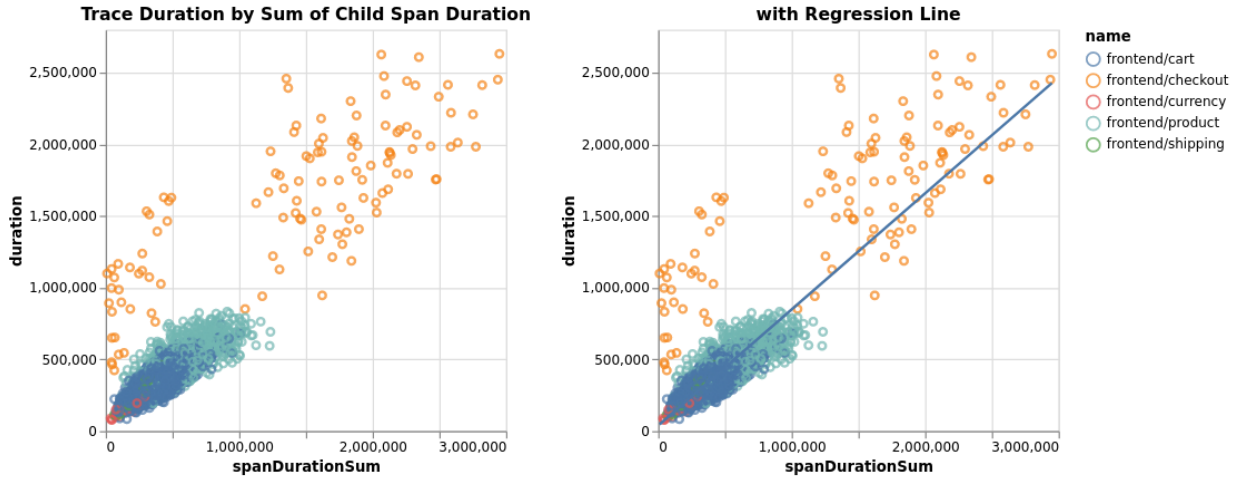




It's a little tough to draw a conclusion here. It seems pretty clear that the traces with more spans tend to be longer, however trace span counts are very segmented by trace type. Within trace types, there is a spread of durations, even though in most cases we have the same number of spans. In this case, I feel like the visualization mostly contradicts the hypothesis...it seems like there is a reasonably strong correlation between span count and trace duration.

H3: For traces with the same root service-endpoint, differences in duration will be related to differences in child duration.

For this particular hypothesis I should explain why this might not be a given. It's possible in these traces that a lot of work is executed in parallel. If that's the case, then the durations of the child spans may not directly impact the duration of the trace, so it's not a guarantee that we would have a tight relationship between parent duration and child span duration.



In this case the relationship is pretty clear. Adding the regression line shows that the slope is just shy of 1:1, so we have a very strong correlation between the duration of the trace and the total duration of child spans. This supports our hypothesis that the duration of child spans is a good predictor of the total trace duration.