Conditional Distributed Tracing

Will Sargent (he/him)
Software Engineer, eero

o11ycon

o11ycon

The Question

What Problems Does Tracing Solve?

- System Health
- Latency Trends and Outliers
- Control Flow Graph
- Asynchronous Process Visualization
- "Debugging Microservices"

So Why Conditional Tracing?

- Flexibility on a Different Axis
- Application Based Span Control
- Exploration of OpenTelemetry SDK
- What Does Debugging Mean?
- And To Whom?

Matt Klein

"Sure, looking at a pretty trace page is fun, but how does a user find a trace that is relevant to their debugging needs? The honest answer is that they often can't or are confused by the UI and give up."

"The reality for most companies is that once deployed, tracing systems get very little real use. Why?"

"Primarily because developers are used to logging and stats, but also because it is simply too hard to find relevant traces, especially when sampling, with existing tooling."

https://twitter.com/mattklein123/status/1049813560686067712

Cindy Sridharan

We've Been Doing It Wrong

"Being able to quickly and cheaply test hypotheses and refine one's mental model accordingly is the cornerstone of debugging. Any tool that aims to assist in the process of debugging needs to be an interactive tool that helps to either whittle down the search space or in the case of a red herring, help the user backtrack and refocus on a different area of the system."

"Instead of seeing an entire trace, what I really want to be seeing is a portion of the trace where something interesting or unusual is happening."

Abby Bangser

Twitter

"A key to observability is BOTH being able to get a high level view AND the detailed debugging. Therefore, if you need to use two different tools for those two different views, each is not independently an observability tool."

"In addition, observability is about empowering an exploratory testing cycle. Continuously generating hypotheses, testing them, collecting outcomes, & using those to generate the next hypothesis."

SWE vs. SRE mental models and tools

[Debugging Incidents in Google's Distributed Systems]

SWEs are more likely to consult logs earlier in their debugging workflow, where they look for errors that could indicate where a failure occurred.

SREs rely on a more generic approach to debugging: Because SREs are often on call for multiple services, they apply a general approach to debugging based on known characteristics of their system(s). They look for common failure patterns across service health metrics (for example, errors and latency for requests) to isolate where the issue is happening, and often dig into logs only if they're still uncertain about the best mitigation strategy.

When In Doubt, Add More Spans

[When In Doubt, Add More Spans: A Tale of Tracing and Testing In Production]

But where Ben saw pain, Toshok saw opportunity. “Well, it’s undoubtedly renderJSON. I’ll optimize that and **add a span to the trace**.” But noooope.

“OK, FINE,” said Story Toshok, “I’ll **propagate the Go context down through to the methods that were calling the DB, and get spans back for /those/ calls** too….” A short time passes. “Hey wait–the code is calling two methods that hit the DB in a loop. Per user. That is…” [does math] “..way too many queries.”

Spans: the new printf debugging

o11ycon

[On the Dichotomy of Debugging Behavior Among Programmers]

[D]evelopers were well-informed about printf debugging and that it is a conscious choice if they employ it, often the beginning of a longer debugging process. Interviewees praised printf as a universal tool that one can always resort back to. [...] While frequently used, developers are also aware of its shortcomings, saying that “you are half-way toward either telemetry or toward tracing”

What goes into debugging statements?

[Industry Practices and Event Logging: Assessment of a Critical Software Development Process]

- **State dump**: this type of pattern reports the value of critical variables or entire data structures in the event log.
- **Execution tracing**: this pattern notifies that a given source code location has been reached during the program execution.
- **Event reporting**: the log entries belonging to this pattern provide textual information regarding the occurrence of an event of interest.

Developer Debugging Needs

Diagnostic Logging: Citations and Sources

Developers have a method to debug and resolve actual vs expected behavior:

- Create a pool of data from statements: state dumps, branches, and state transitions.
- Fish around in that pool with various hooks and queries.
- Keep the most useful statements around as diagnostic logging for later.

This works with logs because debug logging is not enabled by default in production.

But spans are printfs: there is no priority system. More spans = more data = more sampling.

Dynamic Fine Grained Resolution

[The Bones of the System: A Case Study of Logging and Telemetry at Microsoft]

In data-driven environments, such instrumentation is moving towards rule-based approaches, where instrumentation can be added once and then toggled without having to change the code itself. This functionality has enabled data-driven organizations to collect data not just during testing, but long after the product is deployed into retail. As Austin remarks, “What’s really for us is **to be able to realtime turn on and off of what log stuff you’re collecting at a pretty granular level**. And to be able to get the performance back when it is turned off is a big thing for us.”

o11ycon

Implementation

Goals For Conditional Tracing

- Allow behavior to depend on application-specific state
- Augment spans and traces with additional information
- Give Developers The Wheel!

Proof of Concept

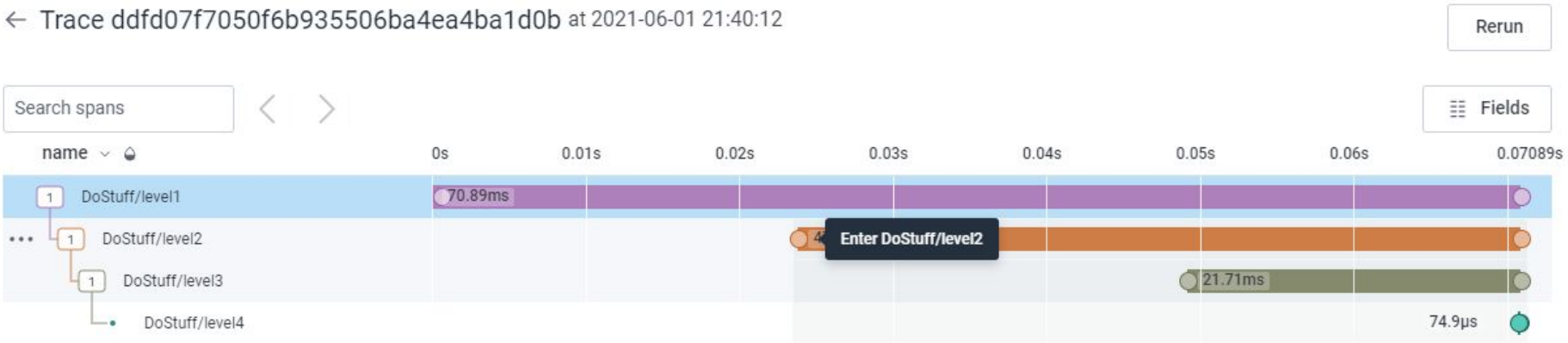
wsargent/conditional-tracing

Tracing driven by Groovy scripts hooked into OpenTelemetry SDK.

- Conditional Sampler allows script-driven sampling.
- Conditional Span Builder allows script-driven span creation.
- Not as fast as bytecode, but you can target by feature flag in production.

Conditional Tracing Demo

o11ycon



Conditional Sampling Demo

Sampling Decisions:

- DROP: not visible to exporter, i.e. Honeycomb.
- RECORD: recorded but not exported.
- RECORD_AND_SAMPLE: recorded and sent to exporter.

The exporter gets confused with exported child spans without an exported parent.

Conditional Sampling Demo

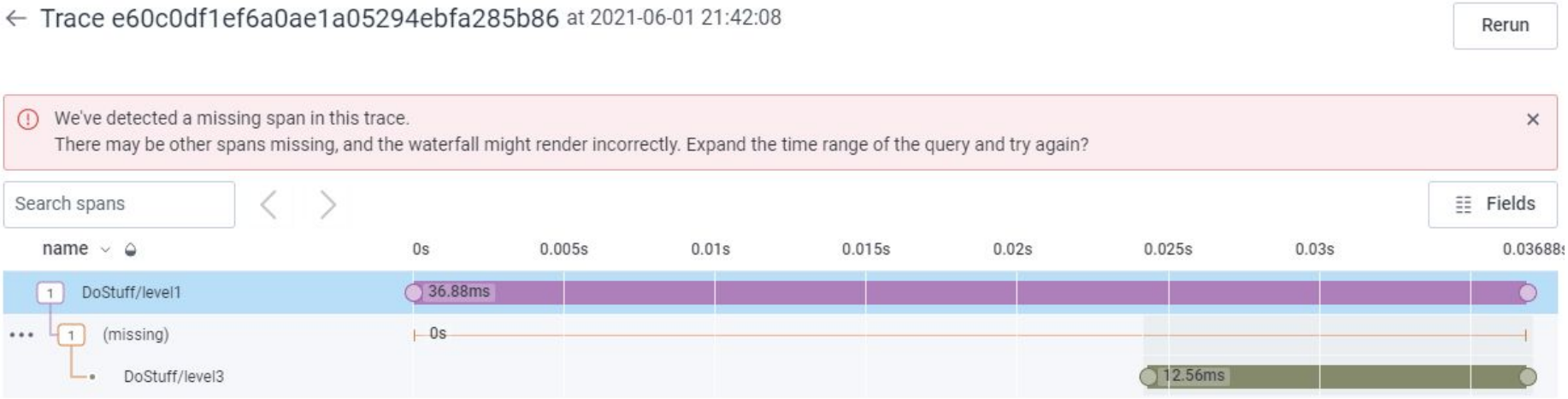
o11ycon

```
SamplingResult shouldSample(SamplingInstance instance) {
   byOddNanos(instance)
}

SamplingResult byOddNanos(SamplingInstance instance) {
   if (System.nanoTime() % 3 == 0) {
       println("byOddNanos: dropping ${instance.getName()} in trace id ${instance.traceId}")
       return SamplingResult.create(DROP);
   } else {
       Attributes attrs = Attributes.of(stringKey("sampler"), "byOddSecond")
       return SamplingResult.create(RECORD_AND_SAMPLE, attrs);
   }
}
```

Conditional Sampling Demo

o11ycon



Conditional Span Builder Demo

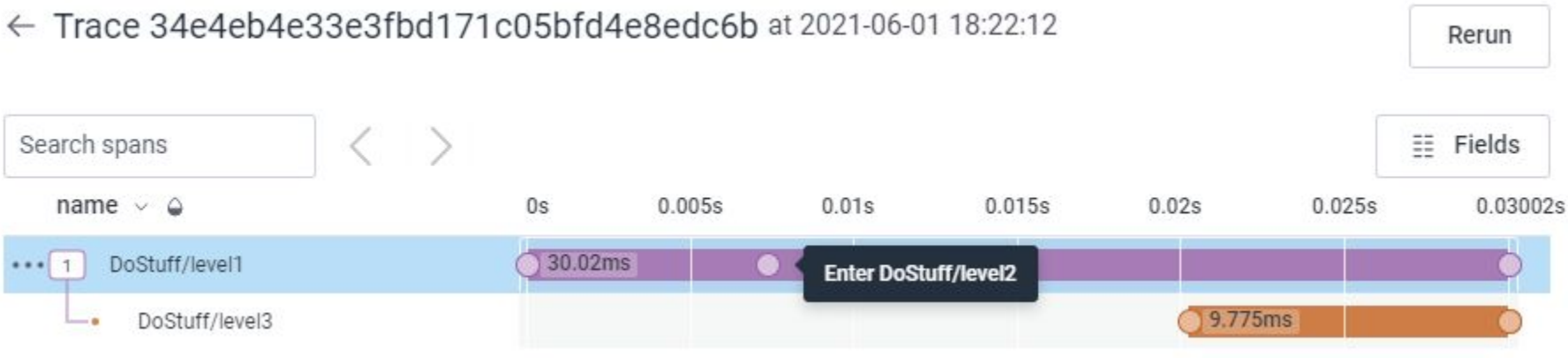
o11ycon

```
boolean shouldStartSpan(String spanName, Context parent, HashMap<AttributeKey<?>, Object>
attributes) {
   return byDebugAttribute(spanName, attributes);
}

boolean byDebugAttribute(String spanName, HashMap<AttributeKey<?>, Object> attributes) {
   if (attributes.containsKey(booleanKey("DEBUG"))) {
       println("byAttribute: ignoring debug span $spanName")
       return false;
   } else {
       return true;
   }
}
```

Conditional Span Builder Demo

o11ycon



Conditional Span Builder Demo

o11ycon

```
boolean shouldStartSpan(String spanName, Context parent, HashMap<AttributeKey<?>, Object>
attributes) {
   return byDeadline(spanName, startOfProjectTimeConst.plusMinutes(10));
}

boolean byDeadline(String spanName, LocalTime deadline) {
   LocalTime currentTime = LocalTime.now();
   if (currentTime.isAfter(deadline)) {
       println("byDeadline: ignoring ${spanName} because $currentTime is after $deadline")
       return false
   } else {
       return true
   }
}
```

Conditional Span Builder Demo

o11ycon

```
boolean byFeatureFlag(String spanName) {
    Context context = Context.current();
    LDClient ldClient = context.get(Main.LD_CLIENT_KEY);
    LDUser user = context.get(Main.LD_USER_KEY);

    if (ldClient.boolVariation("testflag", user, false)) {
        return true;
    } else {
        println("byFeatureFlag: ignoring $spanName for $user")
        return false;
    }
}
```

Conditional Span Builder Demo

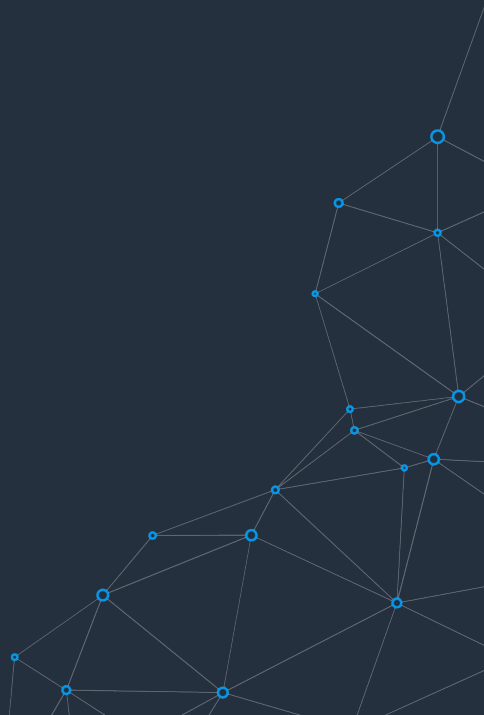
You can break out and fold spans at runtime!

- Only build spans if condition matches, otherwise Span.current().
- Elides spans, so span events all show on same span.
- Requires some SDK mashing.
- Sadly no way to reconstruct elided spans later.

Summary

- Developers and SREs conceptualize "debugging" differently.
- Exploratory debugging involves "drill-down" fine-grained detailed spans.
- Reconciling "debug traces in production" involves conditional logic.
- Application level logic in the SDK gives you **way** more control.
- Sampling is not a solution for rolling up spans...
- Spans are a "write-only" API so span building works fine.

Questions?



o11ycon

o11ycon-hnycon.io