We are Losing Track: a Case for Causal Metadata in Distributed Systems

Rodrigo Fonseca Jonathan Mace
Brown University

As our systems move to more concurrent and distributed execution patterns, the tools and abstractions we have to understand, monitor, schedule, and enforce their behavior become progressively less effective or adequate.

In sequential and simple multi-threaded programs, the execution stack contains valuable information about the recent history of the execution, which we readily use to, for example, debug and profile programs. In addition, thread IDs are useful identifiers to add to log messages and use for resource accounting, and thread-local variables can store useful information about the execution that is more persistent than the stack. These tools, however, become a lot less useful when we introduce patterns such as event loops and continuations, or architectures based on queues and threadpools such as SEDA. Further, important context about the execution in a distributed system is lost when we cross boundaries of software components and machines, making it hard to reason about the execution in a coherent way throughout the system. Inevitably, though, one eventually needs the ability to, at one point of the system, correlate the current context with events that are meaningful at other parts of the system, even when crossing component or machine boundaries.

Of course, this is anything but a new observation, and it has been directly or indirectly addressed in a vast body of existing work, by maintaining a notion of context that follows the execution patterns of applications through events, queues, thread pools, files, caches, and messages between distributed system components.

For example, the node.js community has introduced continuation-local storage (CLS) to address the growing frustration with debugging event-based javascript server code. Taint tracking and DIFC maintain and propagate security labels as the system executes, warning of or prohibiting policy violations; work on data provenance propagates information about the lineage of data as different system components manipulate it; work on consistent updates and snapshots uses vector clocks which are maintained and updated throughout concurrent executions and message exchanges. Many systems propagate an activity ID following execution patterns for debugging and profiling, anomaly detection, resource accounting, or resource management, while others stitch existing identifiers for part of this task. Paraphrasing Greenspun's tenth rule of programming,[1] *any sufficiently complicated distributed system contains an ad hoc, informally specified, siloed implementation of causal metadata propagation.* We can, and should, do better.

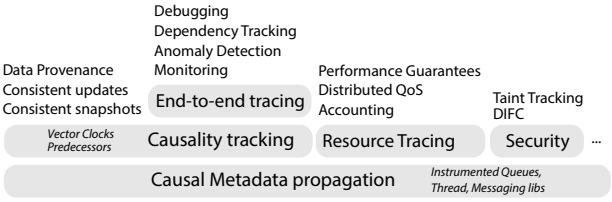

Figure 1: Causal metadata propagation base layer

In our experience examining many of these solutions and building some of them, the most intrusive portion of instrumenting a system to propagate and manipulate context is the propagation itself. Not because it is conceptually hard (although correctly dealing with concurrency is sometimes subtle), but because it requires intervention in key points of the application, libraries, or the OS. Due to this, it is also the hardest to change later, and is usually the focus of criticism against adoption of these approaches. More importantly, however, this effort currently has to be duplicated for each of the examples above.

We argue that systems should be built with causal propagation of generic metadata as a first class primitive, to serve as the *narrow waist* upon which all of the above use cases could be built, in an analogy to the role of the IP layer in networking. Figure 1 shows some of the tools that could use such a propagation abstraction. This layer should be generic, pervasive, incrementally deployable, and interoperable across system components. With a simple and common API it could use language, library, and OS support, and reuse mechanisms that do propagate context in limited scopes, such as thread-local variables or continuation-local storage. Many systems have part of this built-in from necessity, such as HTTP servers that propagate request IDs across modules, or applications servers that share session IDs among components. With Dapper, Google has shown that a similar service can be widely deployed, by instrumenting all of their internal RPCs with causal propagation of IDs for tracing.

We are not the first to propose similar ideas. In this talk we will describe one possible implementation of this metadata propagation layer, many of the existing challenges, and why, even if we haven't done this pervasively enough before, we should address this problem now.

[1] http://philip.greenspun.com/research/

About

Rodrigo Fonseca, the corresponding author, is an assistant professor at Brown University's Computer Science department, interested in networking, distributed systems and operating systems. Of particular relevance to this submission, he has been working on a number of systems and frameworks that use causal metadata propagation. X-Trace [2] instruments distributed systems and produces a distributed call graphs of concurrent system executions spanning multiple layers, components, and machines. Quanto [1] used similar techniques to propagate activity IDs on a sensornet application, for energy accounting across the network. More recently, with his PhD student Jonathan Mace, they have used the propagation of tenant IDs on the entire Hadoop stack to provide near-real-time resource monitoring and accounting in Retro [3], and are currently working on a system that combines dynamic instrumentation with causality tracking to allow for flexible, runtime-defined monitoring of distributed systems (in submission).

More details at `http://www.cs.brown.edu/people/rfonseca`.

References

[1] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI'08)*, pages 323–338, December 2008.

[2] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, Berkeley, CA, USA, 2007. USENIX Association.

[3] J. Mace, P. Bodik, M. Musuvathi, and R. Fonseca. Retro: Targeted resource management in multi-tenant distributed systems. In *NSDI '15: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, May 2015.