A Short Primer on Causal Consistency

WYATT LLOYD, MICHAEL J. FREEDMAN, MICHAEL KAMINSKY, AND DAVID G. ANDERSEN

Wyatt Lloyd is a Postdoctoral Researcher at Facebook and will begin a position as an Assistant Professor at the University of Southern California in 2014. His research interests include the distributed systems and networking problems that underlie the architecture of large-scale Web sites, cloud computing, and big data. He received his Ph.D. from Princeton University in 2013 and his BS from Penn State University in 2007, both in Computer Science. Wyatt.Lloyd@gmail.com

Michael J. Freedman is an Associate Professor of Computer Science at Princeton University, with a research focus on distributed systems, networking, and security. Recent honors include a Presidential Early Career Award (PECASE), as well as early investigator awards through the NSF and ONR, a Sloan Fellowship, and DARPA CSSG membership. mfreed@cs.princeton.edu

Michael Kaminsky is a Senior Research Scientist at Intel Labs and is an adjunct faculty member of the Computer Science Department at Carnegie Mellon University. He is part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), based in Pittsburgh, PA. His research interests include distributed systems, operating systems, and networking. michael.e.kaminsky@intel.com

David G. Andersen is an Associate Professor of Computer Science at Carnegie Mellon University. He completed his S.M. and Ph.D. degrees at MIT, and holds BS degrees in Biology and Computer Science from the University of Utah. In 1995, he co-founded an Internet Service Provider in Salt Lake City, Utah. dga@cs.cmu.edu

The growing prevalence of geo-distributed services that span multiple geographically separate locations has triggered a resurgence of research on consistency for distributed storage. The CAP theorem and other earlier results prove that no distributed storage system can simultaneously provide all desirable properties—e.g., CAP shows this for strong Consistency, Availability, and Partition tolerance—and some must be sacrificed to enable others. In this article, we suggest causal consistency represents an excellent point in this tradeoff space; it is compatible with strong performance and liveness properties while being far easier to reason about than the previously-settled-for choice: "eventual" consistency.

Geo-distributed services are growing in popularity because they can survive datacenter failures and because they move services closer to end users, which lowers page load time and in turn drives up user engagement. For example, companies such as Facebook distribute their service across datacenters on the West Coast, East Coast, and Europe. The recent work in this space includes systems such as PNUTS [2], Walter [11], Gemini [6], Spanner [3], MDCC [5], and Bolt-on [1], as well as our own work on COPS [7] and Eiger [8].

So why does the increasing number of geo-distributed services make consistency a hot topic? Because there is a fundamental, unavoidable tradeoff between having guaranteed low-latency access (which we define as not having to send packets back-and-forth across the country) and making sure that every client sees a single ordering of all operations in the system (strong consistency) [7]. Guaranteed low latency is important because it keeps page load times low. Consistency is important because it makes systems easier to program. In our first work on this subject, COPS, we coined a term for low-latency-favoring systems: ALPS ("Availability, Low-latency, Partition tolerance, and Scalability"). This tradeoff is unavoidable as readers familiar with the famous CAP theorem might remember. Here's an example:

Consider concurrent writes and reads at two different datacenters. If you want both the write to have low latency and the read to have low latency, then you must satisfy them faster than the information can propagate to the other datacenter. In some circumstances, for example, a client might write data to the West Coast datacenter just before another client reads that object from the East Coast datacenter. The East Coast read will return stale information (i.e., it won't reflect that write that actually happened first) because, although the write completed on the West Coast, it hasn't propagated to the other datacenter. You could avoid this behavior and make the write take longer (wait for it to propagate to the East Coast) or the read take longer (fetch the data from the West Coast), but you cannot have both.

This tradeoff is pretty well understood, and is one of the several reasons behind the increasing prevalence of "eventual consistency," popularized by Amazon's Dynamo [4]. The other, of course, is availability: in this example, if the two datacenters cannot communicate, at least one of them must stop processing requests. Eventual consistency allows the datacenters to each return local results, rapidly, even if the other one is down. What it sacrifices, of course, is consistency: queries at different datacenters may see different results, in different order.

This is where causality comes in: you can provide something better than "eventual" consistency without sacrificing availability or low latency. That something is causal consistency, and it has been proved that no stronger form of consistency exists that can also guarantee low latency [9].

What Is Causal Consistency?

Causal consistency means that two events that are "causally" related (or potentially so) must appear in the same order. In other words, if action B occurred after action A (either because a user did A and then B, or because a different user saw A and then did B), then B must appear after A. As a concrete example, consider replying to a snarky comment on someone's Facebook post: your reply should be causally ordered after the snark. And, indeed, this is exactly what causally consistent replication can provide: your reply will never appear to have happened before the snark that triggered it.

Causal Consistency Is Good for Users

Causal consistency improves user experience because with it actions appear to everyone in the correct order. A common scenario where causality is important, but often isn't provided, is comments on social network posts, which sometimes appear out of order.

Consider this stream of posts:

> Oh no! My cat just jumped out the window.
> [a few minutes later] Whew, the catnip plant broke her fall.
> [reply from a friend] I love when that happens to cats!

It looks a little weird if what shows up on someone else's screen is:

> Oh no! My cat just jumped out the window.
> [reply from a friend] I love when that happens to cats!

There are even better examples, widely used, when talking about access control:

> [Removes boss from friends list]
> [Posts]: "My boss is the worst, I need a new job!"

If these two actions occur in the wrong order, then my post will not have been hidden from my boss as intended. Bad news bears.

Causal Consistency Is Good for Programmers

A stronger consistency model restricts the potential orderings of events that can show up at a remote datacenter. This simplifies the reasoning required of a programmer. Imagine two causally related events: Creating a new photo album and then uploading an image to it. If those events are replicated out-of-order, your code might have to try to cope with the idea of an image being uploaded to a nonexistent photo album. Or crash, because you never expected it to happen. In contrast, in a causally consistent system, you might never see the photo upload (or it could be delayed), but it will always occur after the creation of the album. This is the big win from causal consistency for programmers: They do not need to reason about out-of-order actions. Easier code, happier programmers.

What Are the Limitations of Causal Consistency?

Causal consistency is achievable with low latency, and it benefits users and programmers. But it has three drawbacks that practitioners should be aware of.

Drawback #1: Can only capture causality it sees. Actions that take place outside of the system are not seen and, unfortunately, not ordered by the system. A common example of this is a phone call: if I do action A, call my friend on another continent to tell her about A, and then she does action B, we will not capture the causal link between A and B.

Drawback #2: Cannot always enforce global invariants. Each datacenter in a causally consistent system is optimistic in that writes return once they are accepted in the local datacenter. This optimism makes it impossible to allow writes at every datacenter and guarantees global invariants, such as enforcing the rule that bank accounts never drop below 0 dollars.

True global invariants, however, may be rarer than you think. E-commerce is an often cited example, but online stores often handle stock that falls below 0 by issuing back orders for any sales that cannot be filled immediately. And readers familiar with the recent string of concurrent withdrawal attacks where bandits withdrew $40 million from 12 accounts [10] will recognize that even banks rarely enforce global invariants.

Drawback #3: Programmers must reason about concurrent writes. The optimism inherent in causality (when accepting writes at all datacenters) that prevents causal systems from enforcing global invariants also allows there to be concurrent writes to the same data. For instance, a person on the West Coast could update a data item while a person on the East Coast is simultaneously updating that same data item. What should a datacenter do when it has both updates? One common strategy—called the last-writer-wins rule or Thomas's write rule—is to pick one update arbitrarily and have it overwrite the other. This simple procedure is often sufficient: e.g., a social network user can only have one hometown.

There are situations, however, where a more complicated procedure is necessary. For instance, consider a friend request on the East Coast being accepted concurrently with a friend request on the West Coast. Each accepted friend request should increase the count of a user's friends by one (for a total of +2), but if we use the last-writer-wins rule, one update will overwrite the other (for only +1). Instead, we need programmers to write special functions to merge the concurrent updates together (that add the +1s together).

Reasoning about concurrent writes is the main difficulty with using causal consistency for programmers. Specifically, they must ask "are overwrite semantics sufficient?" and if they are not, they must write special functions that preserve the semantics they need.

Conclusion

Causal consistency is a better-than-eventual consistency model that still allows guaranteed low latency operations. It captures the causal relationships between operations and ensures that everyone sees operations in that order. This makes Web sites more intuitive for users, because their actions appear, and are applied, in the order they intended. Causal consistency also makes programming simpler by eliminating the need for programmers to reason about out-of-order operations.

References

[1] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, "Bolt-on Causal Consistency," SIGMOD, June 2013.

[2] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni, "PNUTS: Yahoo!'s Hosted Data Serving Platform," VLDB, August 2008.

[3] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J.J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford, "Spanner: Google's Globally Distributed Database," OSDI, October 2012.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," SOSP, October 2007.

[5] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete, "MDCC: Multi-Datacenter Consistency," EuroSys, April 2013.

[6] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues, "Making Geo-Replicated Systems Fast as Possible, Consistent When Necessary," OSDI, October 2012.

[7] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen, "Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS," SOSP, October 2011.

[8] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen, "Stronger Semantics for Low-Latency Geo-Replicated Storage," NSDI, April 2013.

[9] Prince Mahajan, Lorenzo Alvisi, and Mike Dahlin, "Consistency, Availability, and Convergence," Technical Report TR-11-22, University of Texas at Austin, Department of Computer Science, 2011.

[10] Marc Santora, "In Hours, Thieves Took $45 Million in A.T.M. Scheme," *New York Times,* May 5, 2013.

[11] Yair Sovran, Russell Power, Marcos K. Aguilera, and Jinyang Li, "Transactional Storage for Geo-Replicated Systems," SOSP, October 2011.