Eventually Returning to Strong Consistency

Marko Vukolić
IBM Research - Zurich
mvu@zurich.ibm.com

Abstract

Eventually and weakly consistent distributed systems have emerged in the past decade as an answer to scalability and availability issues associated with strong consistency semantics, such as linearizability.

However, systems offering strong consistency semantics have an advantage over systems based on weaker consistency models, as they are typically much simpler to reason about and are more intuitive to developers, exhibiting more predictable behavior. Therefore, a lot of research and development effort is being invested lately into the re-engineering of strongly consistent distributed systems, as well as into boosting their scalability and performance.

This paper overviews and discusses several novel directions in the design and implementation of strongly consistent systems in industries and research domains such as cloud computing, data center networking and blockchain. It also discusses a general trend of returning to strong consistency in distributed systems, when system requirements permit so.

1 Introduction

Strong consistency criteria, and, in particular, *linearizability* [16], have for years been the gold standard in distributed and concurrent data management. Linearizability has been favored by developers and users alike, as it brings a powerful abstraction that dramatically reduces the complexity of reasoning about data consistency in a distributed system. Specifically, linearizability requires every read/write[1] operation to appear to take place instantaneously at some point between operation's invocation and response. As a result, consistency-wise, linearizability reduces a distributed system to a centralized one — which developers and users have been traditionally accustomed to.

However, although very intuitive to understand, the strong semantics of linearizability make it challenging to implement. This is captured by the *CAP* theorem [6], an assertion that binds strong consistency (linearizability) to the ability of a system to maintain a non-trivial level of availability when confronted with network partitions. In a nutshell, the *CAP* theorem, formally proven in [15], states that in the presence of network partitions, a distributed storage system has to sacrifice either availability or (strong) consistency.

In response, many eventually and weakly consistent distributed systems have recently emerged as an answer to the scalability and availability issues associated with strong consistency semantics. In particular, eventually

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] In this paper, we denote operations that modify the state of a distributed data management system as *write* operations and those that do not as *read* operations.

consistent systems were pioneered already in the 1990s [34], but became popular in the past decade with the advent of cloud computing [36], where scalability and availability requirements are often put before consistency requirements. Roughly speaking, eventual consistency requires replicas in a distributed data management system to eventually converge to identical copies in the absence of further writes.

Over the years, many different consistency notions between strong and weak consistency have been proposed (eventual consistency being one of them). In non-transactional systems alone (i.e., systems in which operations are performed on one data object at a time), more than 50 different consistency flavors have been proposed [35]. Furthermore, dozens of additional consistency notions were proposed in the context of transactional database systems (see e.g., [1]). This multitude of consistency notions as well as the complexity and subtleties separating different nuances of consistency have contributed to the fact that strong consistency remains the preferred correctness condition of a distributed system for vast majority of practitioners and users [33, 9].

Therefore, it is not surprising that a lot of recent and ongoing research has focused on exploring how to make strongly consistent systems scale as much as possible and how to boost their performance. In this paper, we briefly overview a subset of recent research efforts, specifically those focused around the following axes:

- *Consistency hardening (in software).* First, in Section 2, we discuss recent efforts that turn eventually consistent cloud storage systems into strongly consistent ones in a scalable way, effectively *hardening* their consistency notion.
- *Strong consistency in hardware.* Then, in Section 3, we discuss recent efforts that aim at boosting the performance of strongly consistent system by implementing them using modern hardware technologies such as FPGAs, RDMA, Infiniband, etc.
- *Scaling strong consistency for blockchains.* Finally, in Section 4, we discuss ongoing trends in blockchains and cryptoledgers (such as Bitcoin [23]), related to moving from eventually consistent consensus protocols (e.g., a proof-of-work consensus protocol of Bitcoin) to strongly consistent ones. We also highlight the main scalability challenges in this context.

2 Consistency hardening (in software)

As already discussed, eventual consistency is used very often in cloud storage systems. In particular, eventual consistency established itself as a go-to consistency model for very-large-scale object storage systems that provide general-purpose storage to web applications at low cost, typically through a REST interface. For instance, eventual consistency is offered by commercial services based on popular open-source technologies such as Openstack Swift (e.g., IBM Softlayer Object Storage), as well as on proprietary solutions, such as Amazon S3.

Eventual consistency of such storage services increases the complexity of value-added services built on top of them. For instance, multi-cloud storage solutions built on top of cloud object stores only (e.g., [3, 4]) can at best offer *consistency proportionality*, in the sense that the consistency of value-added depends on the consistency of the underlying clouds [4].

To rectify this, recent systems, such as Hybris [13] and SCFS [5], propose to strengthen the consistency of eventually consistent cloud storage systems by relying on a small portion of metadata that is kept strongly consistent. In the following, we briefly outline this technique, called *consistency hardening* [13] (or, alternatively, *consistency anchoring* [5]).

To achieve consistency hardening, systems such as Hybris build on established architectural decision to separate data and metadata (control) planes in distributed storage systems (see HDFS [31] as an example). Then, in addition to typical storage system metadata, such as version control numbers, Hybris adds a cryptographic hash of an object stored in an eventually consistent cloud store. In a sense, Hybris maintains hashes of objects

in a strongly consistent way, while keeping the bulk data separately in eventually consistent clouds. Then on reading data objects from eventually consistent cloud stores, Hybris compares this data to the hash stored in a strongly consistent metadata store, detecting potential inconsistencies and allowing re-tries, effectively masking the temporary inconsistencies. Whereas Hybris keeps hashes in a strongly consistent, Zookeeper [17] cluster on a private cloud, hashes and metadata can be kept in any strongly consistent smaller-scale data store. A Hybris performance evaluation [13] attests that consistency hardening achieves very good performance and scales easily to tens of thousands of operations.

Note that Hybris and SCFS are data-agnostic, in the sense that they rely on system-architectural decisions rather than exploiting semantic aspects of stored data to harden consistency. This is different from other approaches to "putting more strength" into eventual consistency that actually exploit data semantics to turn weaker consistency notions into stronger ones. In particular, Shapiro et al. [30] propose *strong eventual* consistency exploiting conflict-free replicated data types (CRDTs) (e.g., those data types in which operations commute) to boost the consistency guarantees of eventual consistency. In short, an eventually consistent system satisfies strong eventual consistency if correct data replicas that have delivered the same updates also have equivalent state [30].

3 Strong consistency in hardware

As data centers grow in size and volume, with services often running on hundreds to thousands of machines, they increasingly depend on a strongly consistent coordination (or metadata) service. Earlier, we have already mentioned that modern, strongly consistent coordination services (e.g., Zookeeper) easily achieve a throughput of tens of thousands operations per second when run on commodity hardware, combined with reasonable latencies on the order of milliseconds. However, these performance numbers are not sufficient for the high demand of data-center applications which often leads to relaxing consistency, which in turn requires building more complex logic in data-center applications and services to deal with inconsistencies.

To rectify this, a lot of research effort has recently been devoted to speeding up strongly consistent services using modern hardware readily available in data centers. This hardware includes, but is not limited to, field-programmable gate arrays (FPGAs), Remote Direct Memory Access (RDMA), Infiniband and 40/100 Gbps Ethernet (40/100 GbE) networking, etc. In particular, the focus of this research has been on implementing consensus, total order (atomic) broadcast [7] and state-machine replication [29], i.e., conceptually equivalent abstractions on top of which any strongly consistent service can be built. In the following, we briefly describe some of the prominent systems that delegate strong consistency to hardware.

Recently, Istvan et al. [18] demonstrated a Zookeeper-like system in which Zookeeper atomic broadcast [19] is entirely implemented in FPGAs. This implementation of Zookeeper atomic broadcast comes in two network flavors: TCP and a custom-built messaging protocol for boosting performance even further. On 40GbE networking, atomic broadcast of [18] achieves peak throughputs of nearly 4 million operations per second for the custom-built protocol and around 2.5 million operations per second for the TCP variant. The system further exhibits very low latencies on the order of few microseconds, without significant tail latencies. These performance numbers are very promising for enabling strong consistency at data-center scale. For example, it is not difficult to see how systems such as Hybris (see Section 2) would profit from more than two orders of magnitude better performance of their strongly consistent component when implemented in modern hardware instead in software.

DARE [26] is another recent example of a strongly consistent system exploiting modern hardware. It implements a state-machine replication protocol similar to Raft [25]. DARE is optimized for one-sided RDMA, and achieves consensus latency of less than 15 μs with 0.5-0.75 million operations per second running over an Infiniband network. Similarly, FaRM [14] is designed for RDMA over 40GbE and Infiniband. FaRM is a distributed main-memory key value store with consensus-based replication that achieves very high throughput

(up to 10 million requests per second per node for a mixed read/write workload).

Finally, besides these implementations that exploit modern hardware, another interesting and emerging research direction is using software-defined networking (SDNs). Examples include NetPaxos [12], Speculative Paxos [27] and an implementation of Paxos [20] in switches [11]. Although these SDN-based systems typically achieve one to two orders of magnitude worse performance than the hardware implementations discussed above, they get some of the benefits of implementing strongly consistent services closer to hardware, while maintaining a higher level of programming abstraction.

4 Scaling strong consistency for blockchains

Many different *blockchains* or *distributed ledgers*, led by Bitcoin [23], have been emerging in recent years. Although initially reserved for cryptocurrencies (such as Bitcoin itself), blockchain technology is maturing very fast and is embracing all types of asset transactions, ranging from simple currency transactions á la Bitcoin, to complex transactions containing "smart contracts" i.e., custom code executed in a context of a modern blockchain such as Ethereum [38]. Regardless of the type of the transaction, a blockchain should enforce strong consistency, or total order among transactions (at least for transactions that depend on each other and may conflict) to prevent issues such as asset double-spending.

Even though Bitcoin and similar alternative cryptocurrencies (i.e., *altcoints*) boast distributed consensus [23], this is not the classical consensus that underlies total order broadcast and state-machine replication, but rather a sort of an *eventual consensus*. Indeed, when participants in the Bitcoin network try to solve the difficult cryptographic puzzle (i.e., *mine* a block using proof-of-work (PoW)) [23] in an attempt to agree on the next block of transactions, more than one participant may actually mine the next candidate block. Conflicts between candidate blocks are resolved later on by conflict-resolution rules, such as the longest (most difficult) branch rule of Bitcoin, or alternative rules such as the GHOST rule [32].

However, regardless of conflict resolution, classical PoW consensus remains only eventual. This might come as a surprise, having in mind the requirement that blockchain should enforce total order to prevent asset double-spending, as discussed above. The fact that PoW consensus is only eventual is often informally referred to as absence of *consensus finality* in PoW blockchains; in short, *consensus finality* requires that a valid block can never be removed from the blockchain once appended to it [37].

To cope with the absence of consensus finality of PoW eventual consensus and to eliminate the enormous computational overhead of PoW-based consensus [24], blockchain and distributed ledger communities are increasingly turning back to classical, strongly-consistent distributed consensus to power the blockchain [10, 37]. In the case of the trust model of blockchain, this implies the use of Byzantine fault-tolerant (BFT) consensus, in which consensus participants can exhibit arbitrary or Byzantine [21] behavior. As classical BFT consensus is strongly consistent, it also guarantees consensus finality to blockchains based on it. This trend in blockchain research and development of moving from eventually-consistent PoW consensus to classical, strongly-consistent BFT consensus is exemplified by practical systems such as the consensus protocol underlying the Ripple network[2], or Openblockchain[3], an open-source proposal from IBM for the Linux Foundation Hyperledger project[4].

This shift to strong consistency in blockchains comes with a set of challenges. In particular, one of the key challenges for BFT consensus protocols is scalability in terms of the number of nodes (N). Specifically, whereas PoW eventual consensus scales easily with additional nodes, this is less obvious for BFT consensus protocols which often involve $O(N^2)$ message complexity [8], although deterministic protocols with $O(N)$ amortized message complexity exist [28, 2] and randomized protocols with $O(N)$ worst-case message complexity have been

[2] `https://ripple.com.`

[3] `https://github.com/openblockchain.`

[4] `https://www.hyperledger.org`

proposed [22]. For more detailed information on the scalability challenges in BFT-based blockchains, and for a general comparison between PoW (eventual) and BFT (strong) consensus, we refer the reader to [37].

5 Conclusion

The trend of moving from weak, e.g., eventual, consistency to strong consistency is affecting many industries and research communities, such as cloud computing, data center networking and blockchain. In this paper we gave a brief overview of the reasons underlying these trends and of techniques and systems that aim at making strong consistency practical in these important domains.

References

[1] Atul Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. Ph.D., MIT, Cambridge, MA, USA, March 1999. Also available as Technical Report MIT/LCS/TR-786.

[2] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 BFT protocols. *ACM Transactions on Computer Systems (TOCS)*, 32(4):12:1–12:45, January 2015.

[3] Cristina Basescu, Christian Cachin, Ittay Eyal, Robert Haas, Alessandro Sorniotti, Marko Vukolić, and Ido Zachevsky. Robust data sharing with key-value stores. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2012*, pages 1–12, 2012.

[4] Alysson Neves Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12, 2013.

[5] Alysson Neves Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Ferreira Neves, Miguel Correia, Marcelo Pasin, and Paulo Veríssimo. SCFS: A shared cloud-backed file system. In *USENIX Annual Technical Conference (ATC), 2014*, pages 169–180, 2014.

[6] Eric A. Brewer. Towards robust distributed systems (abstract). In *ACM Symposium on Principles of Distributed Computing (PODC)*, page 7, 2000.

[7] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[9] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. 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. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[10] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gun Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains (a position paper). In *3rd Workshop on Bitcoin and Blockchain Research*, 2016.

[11] Huynh Tu Dang, Marco Canini, Fernando Pedone, and Robert Soulé. Paxos made switch-y. *ACM SIGCOMM Computer Communication Review*, 2016.

[12] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. NetPaxos: Consensus at network speed. In *ACM SIGCOMM Symposium on SDN Research (SOSR)*, 2015.

[13] Dan Dobre, Paolo Viotti, and Marko Vukolić. Hybris: Robust hybrid cloud storage. In *ACM Symposium on Cloud Computing (SOCC)*, pages 12:1–12:14, 2014.

[14] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No compromises: distributed transactions with consistency, availability, and performance. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2015.

[15] Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[16] Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990.

[17] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference (ATC)*, 2010.

[18] Zsolt Istvan, David Sidler, Gustavo Alonso, and Marko Vukolić. Consensus in a box: Inexpensive coordination in hardware. In *USENIX symposium on Networked systems design and implementation (NSDI)*, 2016.

[19] Flavio Paiva Junqueira, Benjamin C. Reed, and Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 245–256, 2011.

[20] Leslie Lamport. Paxos made simple. *SIGACT News*, 32(4):51–58, 2001.

[21] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[22] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *Cryptology ePrint Archive 2016/199*, 2016.

[23] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.

[24] Karl J. O'Dwyer and David Malone. Bitcoin mining and its energy footprint. In *IET Irish Signals & Systems Conference*, 2014.

[25] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference (ATC)*, 2014.

[26] Marius Poke and Torsten Hoefler. DARE: high-performance state machine replication on RDMA networks. In *ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2015.

[27] Dan R. K. Ports, Jialin Li, Vincent Liu, Naveen Kr. Sharma, and Arvind Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *USENIX symposium on Networked systems design and implementation (NSDI)*, 2015.

[28] HariGovind V. Ramasamy and Christian Cachin. Parsimonious asynchronous Byzantine-fault-tolerant atomic broadcast. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 88–102, 2005.

[29] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[30] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 386–400, 2011.

[31] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *IEEE Symposium on Mass Storage Systems and Technologies, (MSST)*, pages 1–10, 2010.

[32] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in Bitcoin. In *Financial Cryptography and Data Security (FC)*, pages 507–527, 2015.

[33] Michael Stonebraker. Stonebraker on NoSQL and enterprises. *Communications of the ACM*, 54(8):10–11, 2011.

[34] Douglas B. Terry, Marvin Theimer, Karin Petersen, Alan J. Demers, Mike Spreitzer, and Carl Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 172–183, 1995.

[35] Paolo Viotti and Marko Vukolić. Consistency in non-transactional distributed storage systems. *ACM Computing Surveys (to appear). Also available as arXiv pre-print http://arxiv.org/abs/1512.00168.*

[36] Werner Vogels. Eventually consistent. *Queue*, 6(6):14–19, October 2008.

[37] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *IFIP WG 11.4 Workshop on Open Research Problems in Network Security (iNetSec)*, 2015.

[38] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf, 2015.