Contents

An Overview of Distributed Consensus

Drew Ripberger

December 8, 2019

1 Verification and Checking

1.1 Analysis and Checking Frameworks

- *Verdi: A Framework for Implementing and Formally Verifying Distributed Systems* [18]

  Verdi is a framework for practically verifying distributed systems. Often implementations of distributed systems are too complex to be exhaustively tested, so, Verdi attempts to choose an appropriate fault model to more effectively enumerate bugs and faults. A toolchain is provided to assist in transforming the formal model of the system into implementation.

- *Teaching Rigorous Distributed Systems with Efficient Model Checking* [13]

  While exhaustively determining bugs in a distributed system can be incredibly effective, it can at the same time, be incredibly costly for developers. This paper purposes a model that allows students, or developers with fewer resources at their disposal to efficiently verify their systems and visually debug them. Also included are methods to reduce the search space for potential faults in the system and to detect errors in realtime.

- *An Analysis of Network-Partitioning Failures in Cloud Systems* [1]

  In an analysis of many popular distributed systems, they found that many, while extensively tested, still have numerous faults present. Many of these faults require little to no user input, and are present once a network partition occurs. With this analysis, NEAT is created. NEAT is a testing framework that intelligently injects network-partition faults into a distributed system in order to more efficiently and accurately identify potential errors in the implementation.

1.2 Reducing Fault Search Space

- *SAMC: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems* [9]

Model checking is pertinent for ensuring the safety of large cloud systems, however doing so in implementation is incredibly expensive. SAMC proposes policies to simplify the state of model checking. It uses systematic methods to reduce the verification necessary, with no randomness. SAMC introduces four novel policies to reduce state-space: local message independence(LMI), crash-message independence(CMI), crash recovery symmetry(CRS), and reboot synchronization symmetry(RSS).

- *Lineage-driven Fault Injection* [2]

  A common approach to detecting faults in a distributed system is to randomly inject failures or malformed messages into the network. However, with this approach it is very unlikely that we can detect rarely occurring errors, and moreover it is important to note that this random injection can never guarantee to enumerate all errors in the system. The proposed solution is Molly: a fault injection technique that utilizes a lineage based approach to discover bugs based on what possible errors could have affected the correct outcomes of the system.

2 Generalizations

- *A Generalised Solution to Distributed Consensus* [4]

  This paper attempts to simplify the general consensus problem. It looks at the general consensus problem, and considers how it may be simplified in universal terms with respect to immutable state. They look specifically at the Paxos algorithm as an example. It is synonymous with consensus, though, can be incredibly difficult to understand. This generalized solution to consensus hopes to quell some of this confusion. In analysis, they find that quorum requirements of many algorithms could in fact be weakened.

- *Generalized Consensus and Paxos* [6]

  This is a generalized way of representing the consensus problem. Lamport boils down the main goals of the Paxos algorithm as a set of mathematical generalizations, that can be proven. He also illustrates that main goals for consensus in terms of command-structure sets.

3 Consensus

3.1 Improvements on Paxos and State Machine Replication

- *Fast Paxos* [7]

  Fast Paxos is a new variant of Paxos from Leslie Lamport that emphasizes speed of consensus. By reducing the quorum size, and implementing a new

fast round of Paxos, that tests for liveness. In Fast Paxos rounds are split into Fast, and Classic rounds. During fast rounds, clients may submit proposals directly to the acceptors, though requiring a larger quorum of acceptors.

- *The FuzzyLog: A Partially Ordered Shared Log* [11]

  Given the cost of maintaining a total order with a shared log, FuzzLog proposes using a partial order in order to cut down on the associated expense. In this partially ordered log there exist DAGs of updates, that are uniquely *colored* based on geographic region. Resulting replication is much simpler to achieve, as differently *colored* chains are stored and have to be explicitly updated at each replica.

3.2 Geo-replication and WANs

- *SDPaxos: Building Efficient Semi-Decentralized Geo-replicated State Machines* [19]

  Systems attempting geo-replication have run into multiple notorious problems: mainly load imbalance. SDPaxos proposes an alternative algorithm that is based on Paxos, that separates consensus into two distinct phases, replicating the commands to the nodes, and enforcing a consistent order on the nodes. This is done in an attempt to curb workload imbalance by maintaining optimal one-trip latency in two steps.

- *Mencius: building efficient replicated state machines for WANs* [12]

  Traditional consensus algorithms, like Paxos, are effective in local contexts, but in WANs they often suffer the consequence of geographic separation. Often when Paxos, or a version of it, is implemented in a WAN there is a definite increase in network latency, decrease in network throughput and much more prevalent problems with load distribution. Mencius however, is the proposed algorithm that attempts to lessen problems traditionally associated with WANs and consensus. It does this by partitioning sequences (of commits) across multiple nodes in the network, slowly reducing the load on any one specific participant. Mencius adaptively allows nodes with less load to skip their turns and propose changes.

- *MDCC: Multi-Data Center Consistency* [5]

  MDCC is a commit protocol for geographically separated data centers. Given the increased round-trip time for distant datacenters, it becomes imperative to reduce any possible unnecessary messages. MDCC maintains strong consistency while most other similar protocols rely on eventual consistency. MDCC's one round-trip commit time is achieved by piggybacking commit state on transaction messages and by executing Generalized Paxos in parallel on individual records.

- *On the correctness of Egalitarian Paxos* [16]

  Egalitarian Paxos utilizes an execution graph to order commands in the state machines of individual processes. Generally this speeds up latency, as in the most favorable and most common case, only one round trip time is taken to commit the next command. Though while the algorithm is fundamentally correct, there is an error that can potentially lead to inconsistency between replicas present in both the Go implementation and the TLA^+ specification.

- *There Is More Consensus in Egalitarian Paxos* [14]

  EPaxos is a variant of the Paxos algorithm that builds off previous improvements brought on by projects like Mencius and Generalized Paxos. It looks to improve load balancing across a wide area network and to improve the network throughput. With EPaxos, a simply majority of replicas need to be non-faulty. This is achieved by removing any leader process, which would serve as a bottleneck. Instead participating nodes have choice as to where they submit. As a result the network load can be much more evenly distributed. Now no network recovery is needed when a leader process is downed, creating greater availability.

- *Don't Settle for Eventual: Scalable Causal Consistency for Wide-area Storage with COPS* [10]

  COPS is a distributed key-value store that implements the newly defined *causal+*, a strong consistency level for WANs. In the past many protocols have elected to use a weaker consistency level for the benefit to availability and throughput, but a stronger consistency makes reasoning about the behavior of a system much easier for developers. To make wide-area distributed systems more practical and scalable, COPS continually checks causal dependencies in a local cluster before exposing writes.

4 Databases and Implementations

- *Spanner: Google's Globally-Distributed Database* [3]

  Spanner is a distributed database developed at Google, with the goal of highly available data in different geographic regions. The database is sharded into multiple Paxos replicas in order to better horizontally scale. Data in Spanner is automatically resharded to balance load. Using their novel *TrueTime* API, Spanner shows how it can practically guarantee consistency on top of availability with its hyper realistic clock.

- *Calvin: fast distributed transactions for partitioned database systems* [17]

  Calvin serves as a replication layer that sits on top of a distributed database, attempting to efficiently and cost effectively allow for distributed transactions and easy scaling. The main goal of Calvin is to make it possible

to turn a generic unreplicated database into a fully ACID complaint distributed database. To do so, it implements multiple layers for sequencing and ordering transactions into a serial order in a global log. This global log is then used to ensure a proper ordering for each individual partition.

5 The Byzantine Generals Problem

The fundamental problem of consensus can be represented with the Byzantine Generals Problem [8].

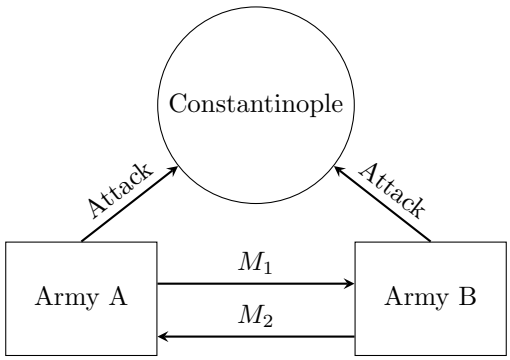

Figure 1: Illustration of the Byzantine Generals problem for two armies A & B

To illustrate the fundemental problem that consensus algorithms are trying to solve, we can imagine a hypothetical situation where two armies A and B are attempting to attack the city of Constantinople. This is classically called the Byzantine Generals problem.

Imagine that two armies at once are required to attack at once if they would like to successfully take Constantinople, so both A and B need to coordinate if they want wage a successful attack on the city. Both armies also have the ability to communicate, but only with messengers. In an attempt to start planning for their siege, army A sends M_1 to army B with the following message:

Attack Constantinople at 02:00.
- Army A

After receiving M_1, army B agrees to the plan, so to confirm with army A they reply in agreement with M_2:

Sounds like a plan. Attacking at 02:00.
- Army B

At the surface level everything looks simple here. Army A and B have a simple process on planning out their attack on the city. However, this is assuming they have uninterrupted communication.

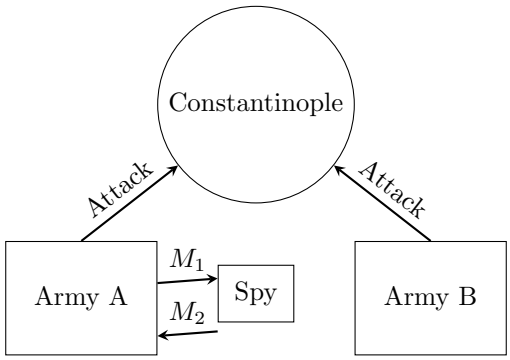

Figure 2: The Risks in The Byzantine Generals problem

6 Breakdown of Consensus Algorithms

Algorithm	Date	Local/WAN	Citation
Raft	2014	Local	[15]
Generalized Paxos	2005	Local	[6]
EPaxos	2013	WAN	[14]
SDPaxos	2018	WAN	[19]
Mencius	2008	WAN	[12]

7 Consensus over WANs

8 Modern Distributed Databases

References

[1] Alquraan, A., Takruri, H., Alfatafta, M., and Al-Kiswany, S. An analysis of network-partitioning failures in cloud systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, Oct. 2018), USENIX Association, pp. 51–68.

[2] Alvaro, P., Rosen, J., and Hellerstein, J. M. Lineage-driven fault injection. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 331–346.

[3] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., and Woodford, D.

Spanner: Google's globally-distributed database. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)* (Hollywood, CA, 2012), USENIX Association, pp. 261–264.

[4] Howard, H., and Mortier, R. A generalised solution to distributed consensus. *CoRR abs/1902.06776* (2019).

[5] Kraska, T., Pang, G., Franklin, M. J., Madden, S., and Fekete, A. Mdcc: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, 2013), EuroSys '13, ACM, pp. 113–126.

[6] Lamport, L. Generalized consensus and paxos. Tech. Rep. MSR-TR-2005-33, March 2005.

[7] Lamport, L. Fast paxos. *Distributed Computing 19* (October 2006), 79–103.

[8] Lamport, L., Shostak, R., and Pease, M. The byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (July 1982), 382–401.

[9] Leesatapornwongsa, T., Hao, M., Joshi, P., Lukman, J. F., and Gunawi, H. S. SAMC: Semantic-aware model checking for fast discovery of deep bugs in cloud systems. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association, pp. 399–414.

[10] Lloyd, W., Freedman, M. J., Kaminsky, M., and Andersen, D. G. Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2011), SOSP '11, ACM, pp. 401–416.

[11] Lockerman, J., Faleiro, J. M., Kim, J., Sankaran, S., Abadi, D. J., Aspnes, J., Sen, S., and Balakrishnan, M. The fuzzylog: A partially ordered shared log. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, Oct. 2018), USENIX Association, pp. 357–372.

[12] Mao, Y., Junqueira, F. P., and Marzullo, K. Mencius: Building efficient replicated state machines for wans. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 369–384.

[13] Michael, E., Woos, D., Anderson, T., Ernst, M. D., , and Tatlock, Z. Teaching rigorous distributed systems with efficient model checking. In *EuroSys* (Dresden, Germany, Mar. 2019).

[14] Moraru, I., Andersen, D. G., and Kaminsky, M. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 358–372.

[15] Ongaro, D., and Ousterhout, J. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX ATC'14, USENIX Association, pp. 305–320.

[16] Sutra, P. On the correctness of egalitarian paxos. *CoRR abs/1906.10917* (2019).

[17] Thomson, A., Diamond, T., Weng, S.-C., Ren, K., Shao, P., and Abadi, D. J. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2012), SIGMOD '12, ACM, pp. 1–12.

[18] Wilcox, J. R., Woos, D., Panchekha, P., Tatlock, Z., Wang, X., Ernst, M. D., and Anderson, T. Verdi: A framework for implementing and formally verifying distributed systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2015), PLDI '15, ACM, pp. 357–368.

[19] Zhao, H., Zhang, Q., Yang, Z., Wu, M., and Dai, Y. Sdpaxos: Building efficient semi-decentralized geo-replicated state machines. In *ACM Symposium on Cloud Computing 2018 (SoCC)* (October 2018), ACM.