How Does Consensus-Based Replication Work in Distributed Databases?

Sid Choudhury
Aug 3, 2018 · 6 min read

Whether it be a WordPress website's MySQL backend or Dropbox's multi-exabyte storage system, data replication is at the heart of making data durable and available in the presence of hardware failures such as machine crashes, disk failures, network partitions and clock skews. The basic idea behind replication is very simple: keep multiple copies of data on physically isolated hardware so that the failure in one does not impact the others and as a result, the system does not lose any data and remains highly available. An additional benefit of replication is that the replicas can be used to serve more client requests and that too faster, leading to higher throughput and lower latency.

Data Replication in Action (Source)

However, the implementation of replication protocols is far from simple. This post focuses on consensus-based replication and how it gets implemented in distributed databases.

Understanding Consensus

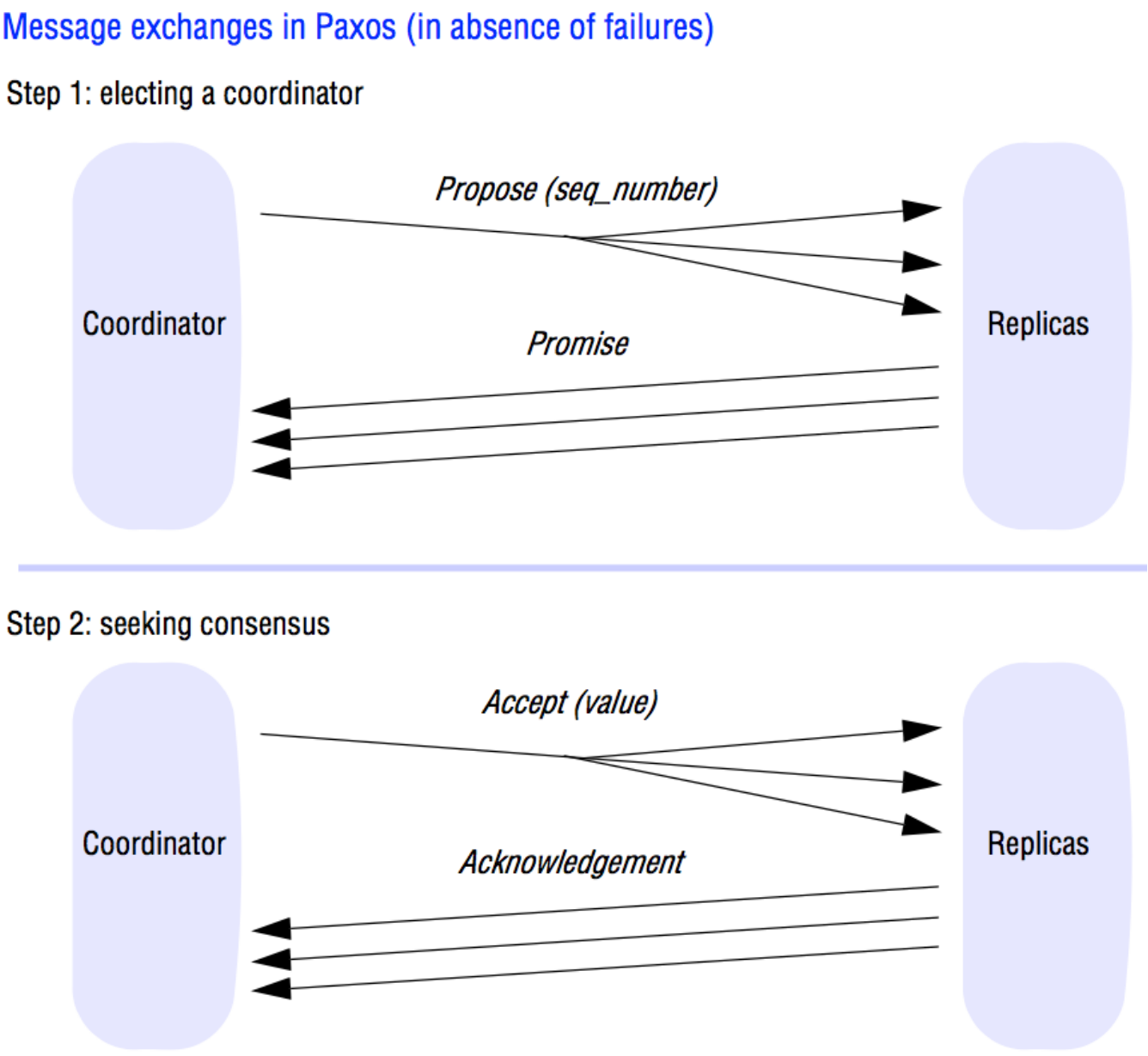
Consensus (aka Distributed consensus) involves multiple servers agreeing on values. It is a fundamental problem in fault-tolerant distributed systems. Once the servers agree on a value, that agreement is final. Typical consensus algorithms accept write requests when any majority (aka quorum) of their servers is available; for example, a cluster of 5 servers can continue to accept writes even if 2 servers fail. If more servers fail, they stop accepting any new write requests. This also means that the values in the remaining available servers do not change and read requests continue to be served with correct values. The end result is a strongly consistent system. Examples of applications of consensus include whether to commit a transaction to a database, agreeing on the identity of a leader, state machine replication, and atomic broadcasts.

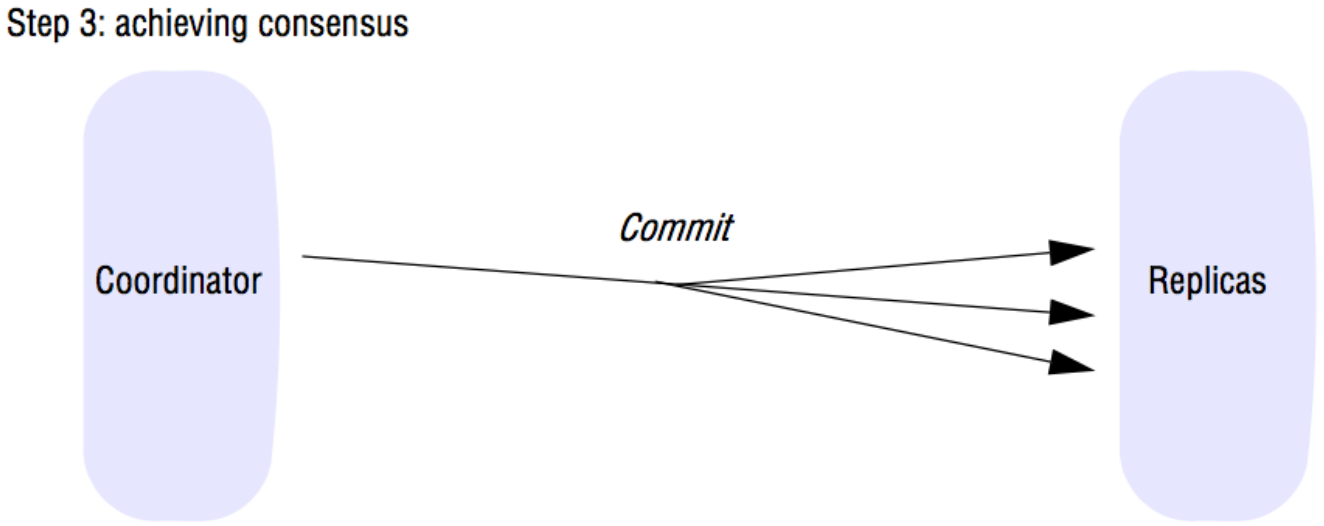
Consensus protocols can be broadly classified into two categories: leader-based and leaderless. Paxos and Raft are the two most commonly used leader-based consensus protocols where the task of data updates and replication is handled by a **leader**. Strongly consistent distributed databases over the years have standardized onto one of these protocols. Leaderless protocols are harder to implement but have higher availability than leader-based protocols. Application of leaderless protocols can be found in blockchain distributed ledgers. Given the focus of this article on distributed databases, we will review Paxos and Raft in depth here on.

Paxos

The Paxos algorithm was first described by Turing Award winner Leslie Lamport in 1990 using the example of a parliament in the ancient Greek island of Paxos. Given the Greek mythology context and a Indiana-Jones-style protagonist, it was not taken seriously at the beginning. However, it was finally published in 1998. People continued to find it hard to understand, prompting Lamport to publish Paxos Made Simple in 2001. Since then it has been recognized as one of the seminal papers in distributed systems.

At the heart of Paxos is a three-phase commit protocol that allows participants to give up on other stalled participants after some amount of time. It describes the actions of the participants by their roles in the protocol: client, acceptor, proposer, learner, and leader (aka a distinguished proposer). Many participants may believe they are leaders, but the protocol only guarantees progress if one of them is chosen. This essentially becomes the first phase of the protocol as shown in the figure below.





Paxos as a 3PC Protocol (Source)

Google's Chubby distributed lock service is one of the most cited Paxos implementations given its wide usage inside Google. Examples of distributed databases with Paxos replication are Google's Spanner and Apple's FoundationDB.

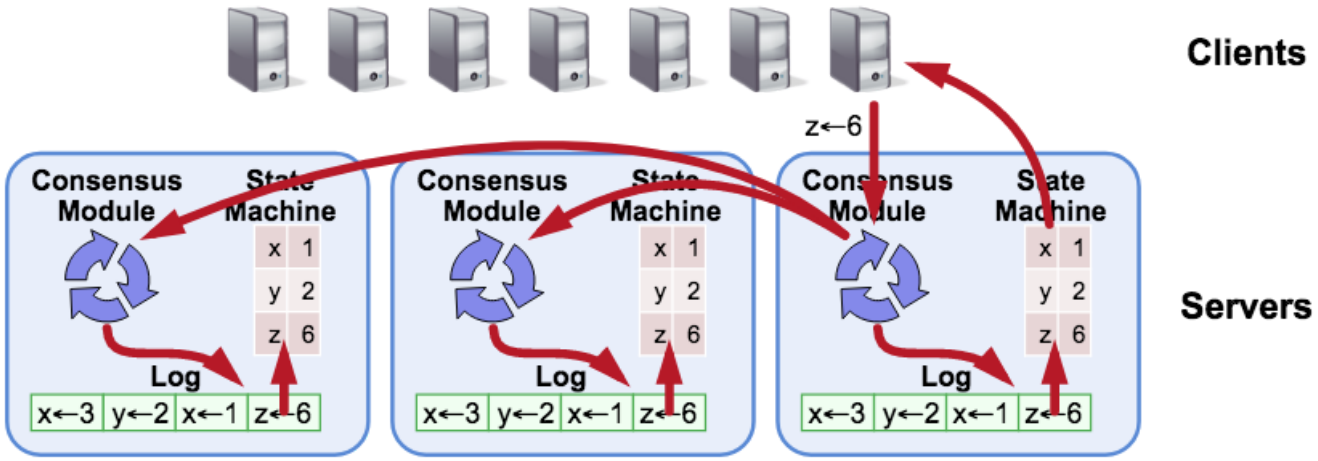
The single biggest problem with Paxos even after so many years in practice is that it remains hard to understand and thereafter correctly implement. In fact, Paxos has evolved into a family of protocols so new tradeoffs can be introduced and the implementation can be simplified. Google's 2006 paper Paxos Made Live — An Engineering Perspective highlights this problem.

> *Despite the existing literature in the field, building such a database (Paxos-based) proved to be non-trivial.*

Raft

Raft, first proposed by Stanford researchers in 2013, is a consensus algorithm that's specifically designed as an alternative to Paxos. It is more understandable than Paxos by means of separation of logic and is also formally proven safe. The separation of logic stems from the fact that Raft makes leader election a separate and mandatory step before the leader can get the data replicated where as a Paxos implementation would make the leader election process an implicit outcome of reaching agreement through data replication.

Separating leader election allows dynamic membership changes with ease — server additions and removals, now more important than ever before because of the public cloud, can be handled by simply re-running leader election. Additionally, in the absence of any unplanned failures or planned membership changes, the leader election step can be skipped altogether. Note that the leader election step is automatic whenever such changes happen even when no new writes are coming into the system. Finally, Raft imposes the restriction that only the servers with most up-to-date data can become leaders. These optimizations radically simplify edge cases in which a succession of leadership changes can result in data discrepancies, but the tradeoff is that leader election in Raft can be more complicated than its counterpart in Paxos.



Raft in Action (Source)

With 40+ open source implementations, Raft is the de-facto standard today for achieving consistency in modern distributed systems. Popular implementations include those from etcd and consul. Next-generation distributed databases such as YugaByte DB, CockroachDB and TiDB use Raft for both leader election and data replication. Other databases such as MongoDB and InfluxDB use Raft partially. MongoDB's leader election in a Replica Set became Raft-based since v3.4 but data replication remains asynchronous (secondary members periodically pull from the primary member of the

Replica Set). InfluxDB uses Raft for high availability of its metadata nodes while using simple non-consensus replication for the actual data nodes (see next section).

Paxos/Raft vs. Non-Consensus Replication Protocols

A design best practice in distributed databases is that Paxos and Raft are applied on an individual shard level as opposed to all the data in the database. This means the leaders (of the various shards) are not present on a single server but are distributed across all the servers. A common alternative to Paxos/Raft is a non-consensus (aka peer-to-peer) replication protocol such as the ones used by first-generation NoSQL databases such as Amazon DynamoDB, Apache Cassandra, Couchbase and InfluxDB. In this case, every replica of a shard is treated equal and hence can accept write requests. Depending on the consistency level configured, the replica taking the write will decide whether to update other replicas synchronously or asynchronously. The challenge with this approach is that concurrent writes on the same record at two different replicas are considered perfectly valid and the final value has to determined non-deterministically using heuristics such as Last-Writer-Wins (LWW) and Conflict Free Replicated Data Types (CRDT). Such systems are considered eventually consistent (since replicas may not agree on the final value) and are prone to data loss upon failures.

Summary

Consensus-based replication has become key to building resilient, strongly consistent distributed systems. While Paxos has evolved over the years into a family of protocols with various tradeoffs, it still remains complex to implement. Raft has emerged as the leading alternative to Paxos given its focus on understandability without sacrificing correctness and performance. Distributed databases are increasingly reliant on Raft consensus protocol for their basic replication needs. We will review how YugaByte DB uses Raft for its leader election and data replication in a future post.

What's Next?

- Read 6 Signs You Might be Misunderstanding ACID Transactions in Distributed Databases to learn how ACID compliance is built on top of consensus-based replication.
- Compare YugaByte DB to databases like DynamoDB, Cassandra, MongoDB and Cosmos DB.
- Get started with YugaByte DB.

- Contact us to learn more about licensing, pricing or to schedule a technical overview.

. . .

Originally published at blog.yugabyte.com.

Database NoSQL Sql Data Replication AWS