Summary report of *In Search of an Understandable Consensus Algorithm*

Review

This paper introduce fundamentals of a easily understandable consensus algorithm named Raft. It can produce equivalent results at the equivalent efficiency to Paxos, another consensus algorithm but quite difficult to understand and master. The authors believe that Raft is superior to Paxos and other consensus algorithms, both for educational purposes and as a foundation for implementation. Because even though similar in many ways to other consensus algorithms, Raft has several novel features: strong leader, leader election and membership changes.

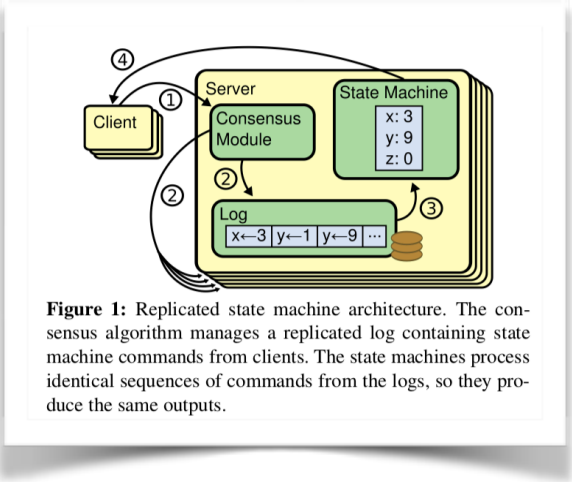


Figure 1: Replicated state machine architecture. The consensus algorithm manages a replicated log containing state machine commands from clients. The state machines process identical sequences of commands from the logs, so they produce the same outputs.

Distributed consensus is getting multiple servers to agree on shared state, even in the case of failures.Typically we use consensus to build what's called replicated state machine. Figure[1] is the architecture of replicated state machine, it's some data structure you want your clients to interact with. The figure is showing a cluster of four servers, where each server has its own copy of the state machine. At the same series of commands, the logs on different servers look the same, then all the state machines will end up the same state. The system as a whole is going to give an illusion that there is only one coherent state machine. Raft implements consensus by first electing a distinguished leader, then giving the leader complete responsibility for managing the replicated log.

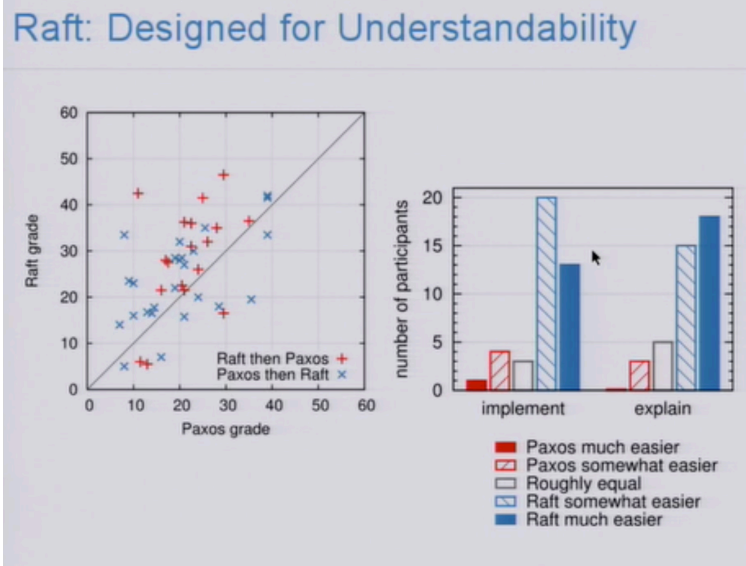
- Leader election: The idea is to select one of the servers as leader, and when that server goes down, we want to detect crashes and choose new leader to take its place.

- Log replication: That's what happens during normal operations. It's leader's job to take commands from clients and append them to its log. And then make the other servers' log match. To do that, the leader will replicates its log to other servers and overwrite in consistencies.
- Safety: They add a constriction on leaders that only a server with an up-to-date log can become leader. Given that, when the leader replicate its log to other servers, it is actually safe since the leader's log is up-to-date.

Three strong points of this paper

(1)Strong point: Raft is superior to Paxos and other consensus algorithms, both for educational purposes and as a foundation for implementation.

Explanation:



Authors have done a survey about Raft vs Paxos. Results from this user study demonstrate that Raft is easier for students to learn than Paxos. They want the algorithm to facilitate the development of intuitions that are essential for system builders. Because they believe that it is important not just for the algorithm to work, but for it to be obvious why it works. So Raft is superior to Paxos and other consensus algorithms, both for educational purposes and as a foundation for implementation.

(2)Strong point: Authors designed the Raft log mechanism to maintain a high level of coherency between the logs on different servers.
Explanation:
Not only does this simplify the system's behavior and make it more predictable, but it is an important component of ensuring safety. Raft maintains the following properties, which together constitute the Log Matching Property in Figure[3]:

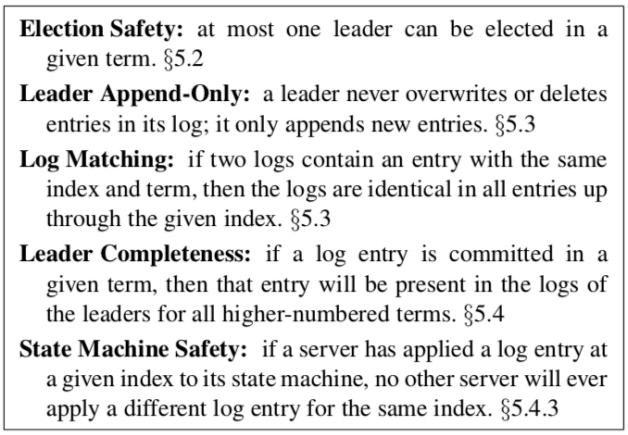


Figure 3: Raft guarantees that each of these properties is true at all times. The section numbers indicate where each property is discussed.

- If two entries in different logs have the same index and term, then they store the same command.
- If two entries in different logs have the same index and term, then the logs are identical in all preceding entries.

(3)Strong point: In order to ensure safety, configuration changes must use a two-phase approach.
Explanation:

The joint consensus combines both the old and new configurations:

- Log entries are replicated to all servers in both configurations.
- Any server from either configuration may serve as leader.
- Agreement (for elections and entry commitment) requires separate majorities from both the old and new configurations.

Three weak points of this paper

(1) Weak point: They use randomization to simplify the Raft leader election algorithm.

Explanation:

Authors are faced with two problems when designing Raft:

- how hard is it to explain each alternative?
- how easy will it be for a reader to completely understand the approach and its implications?

They recognize that there is a high degree of subjectivity in such analysis, so they choose two approaches, the second one is to simplify the state space by reducing the number of states to consider, making the system more coherent and eliminating nondeterminism where possible. In particular, randomized approaches introduce nondeterminism, but they tend to reduce the state space by handling all possible

choices in a similar fashion.I think this is a weak point because the benefit of choosing randomization is not well explained or proved.

(2) Weak point: With this mechanism, a leader does not need to take any special actions to restore log consistency when it comes to power.
Explanation: The paper claims that the leader just begins normal operation, and the logs automatically converge in response to failures of the Append-Entries consistency check. But not explained in details or profs, so I'm not sure about the correctness.

(3) Weak point: Raft RPCs are idempotent, so this causes no harm.
Explanation: If a server crashes after completing an RPC but before responding, then it will receive the same RPC again after it restarts. So the reason why this will cause no harm is not explained in details or profs.

My thought on this paper

I'm very impressed that authors of this paper put so much importance and effort on making a consensus algorithm easily understandable. They have several goals in designing Raft, and I think these goals are worth of reference if we want to build an algorithm of our own:

- it must provide a complete and practical foundation for system building, so that it significantly reduces the amount of design work required of developers.
- it must be safe under all conditions and available under typical operating conditions.
- it must be efficient for common operations.
- (most important goal and most difficult challenge) it must be possible for a large audience to understand the algorithm comfortably.
- it must be possible to develop intuitions about the algorithm, so that system builders can make the extensions that are inevitable in real-world implementations.