Distributed Consensus Protocols

ABSTRACT

In this paper, I compare Paxos, the most popular and influential of distributed consensus protocols, and Raft, a fairly new protocol that is considered to be a better alternative to Paxos.

1. INTRODUCTION

Distributed Consensus algorithms manage coherence across nodes that are connected over a network. It is mainly used in synchronous log replication in distributed systems. Every large, replicated software system makes use of a consensus protocol in some way.

Leslie Lamport introduced Paxos, the dominant consensus protocol for a long time, in the 1980s. Unfortunately, it made use of an analogy that few understood and so it was not published. He republished the paper, with a much more simpler explanation in 2001, and after that, the paper started gaining popularity.

While the proof of his protocol was clear, the implementation details of the protocol in real world systems was lacking in the paper. Lamport had some suggestions, but they did not work very well practically [2].

Google decided to use Paxos in one of their systems, Chubby. Chubby is a distributed locking system designed to store small files [3]. They found the implementation of Paxos to be non-trivial. The basic protocol (by Lamport) only explains how Paxos works for a single value, and a real world system would have to extend Paxos to work with a sequence of values (like a log). This came to be known as Multi-Paxos.

Diego Ongaro and John Ousterhout designed Raft in 2012 [1]. It had a result equivalent to Multi-Paxos and was also proven. The main difference though, was that it was structured to be more easily understandable. It was implemented by the authors, and hence was also more practical.

Raft has quickly gained popularity over the past few years, mainly because it's easy to understand and implement. A lot of open source implementations exist today in different languages.

This paper talks about the Motivation behind this comparison (Section 2), other popular consensus protocols as Related Work (Section 3), the Methodology I used for comparison (Section 4), my Evaluation of the protocols (Section 5), and finally the Conclusion and References (Sections 6 and 7).

2. MOTIVATION

Distributed Consensus protocols could be called the most important part of a Distributed system. The nodes of any distributed system should agree on something to be part of the same system.

I will explain, briefly, about the history of consensus protocols and the motivation behind protocols like Paxos and Raft.

2.1 2-Phase Commit

The most intuitive and basic of consensus protocols is the two-phase commit. Here is how the protocol works:

2.1.1 Proposal phase:

A node in the system (called the coordinator node) proposes a value to every other node in the system. The other nodes respond with a Yes or No depending on whether or not they accept that value.

2.1.2 Commit-or-abort phase:

If all the nodes agree to commit, the coordinator commits the value and informs all the other nodes to commit the value too. If any one of the nodes does not accept the value, the process is aborted.

The main problem with the 2PC is that if the coordinator crashes at any point, the whole system comes to a halt until the coordinator comes back up. Also, even if one of the other nodes failed, the system goes into deadlock, as every node's vote is needed to reach consensus.

2.2 3-Phase Commit

This protocol solves the problem of 2PC by adding an extra phase. After the proposal phase explained previously, the coordinator sends a prepare-to-commit message to the nodes with the decision of whether to commit or not. If the coordinator gets a response from the nodes, it goes ahead with the commit phase.

Now, as every node has the decision stored independently, the coordinator or any other node going down does not affect the entire transaction. A backup coordinator could be started up to continue the protocol.

The problem with the three-phase commit is that it does not withstand network partitions. A network partition could result in messages not being delivered, and a backup coordinator on either side of the partition could bring the system to two different states.

2.3 Paxos

The key difference between Paxos and the protocols before it was the concept of majority. A value was said to be committed if it existed in a majority of nodes in the system (above 50%) and not necessarily every node in the system.

Paxos can handle network partitions. It handle fail-stop and fail-recover failure scenarios but not Byzantine failures.

I was particularly intrigued by the Paxos protocol, as it was notorious for its difficulty. It actually looked deceptively simple at the surface, but as I went into the details, the complexity began to increase. I wondered if there were simpler alternatives used in practice today, and found Raft. I enjoyed the author's perspective of dumbing down Paxos, by changing the structure of the protocol, while still maintaining the same results. Hence I decided to do a comparison between these protocols.

3. RELATED WORK

Over the years, a few other consensus protocols have been discovered and are commonly used today. This section talks about the most popular of these.

3.1 Zab (Zookeeper)

Zookeeper is a service for coordinating processes in distributed applications [4]. It uses a consensus protocol called Zookeeper atomic broadcast (Zab). Zab looks like Multi-Paxos, with a leader and unique proposals and majority acknowledgements for commits.

The key difference between Zab and Multi-Paxos is the ordering of client requests. When Multi-Paxos gets concurrent requests from multiple clients, it is free to reorder them at will. The only guarantee it provides is that the ordering is the same in all nodes.

Zab follows strict ordering decided by the Leader. If the leader goes down, the next leader is not allowed to reorder requests, unlike Multi-Paxos. Zab is more useful for backup systems, whereas Multi-Paxos is more suited for state replication [5].

3.2 VSR (Harp file system)

Harp, a replicated UNIX file system, uses a consensus protocol called Viewstamped replication (VSR) [6]. VSR is similar to Zab in the sense that every node is not a state machine and just follows the order of the primary.

VSR differs from both Paxos and Raft in recovery, memory loss tolerance, time of execution and sequencer selection [7].

3.3 Proof of work (Bitcoin)

While Bitcoin differs from the other consensus protocols in terms of usage, I think it deserves a mention because of the sheer scale at which it operates on and its novel way of solving the consensus problem.

Bitcoin is a peer-to-peer online payment solution. As it does not have a central authority managing transactions, the entire system depends on a consensus protocol [8]. Unlike the other protocols mentioned here, Bitcoin's consensus protocol can handle Byzantine failures.

Bitcoin's transaction information is stored as blocks. The blocks are ordered, and each block has a unique hash. This hash depends on the transactions in that block and the hash of the preceding block. A block is added to the chain when a specific amount of CPU power is spent on the block. This spending is easily verifiable.

When a block is to be added to the transaction chain, it is announced and verified by a majority of other nodes. After being added, if a block has to be changed, every block after that block has to be rehashed and reworked upon. This makes it essentially impossible for a block to be modified.

Bitcoin has more than 5000 nodes in its network at any time. Nodes are not meant to be permanent and so this number keeps changing throughout the day. A transaction takes around 8 minutes to be verified.

4. METHODOLOGY

As mentioned before, the major difference between Paxos and Raft is the practicality of the protocol. The pseudo code for the basic version of the Paxos algorithm is in a figure on the next page [9].

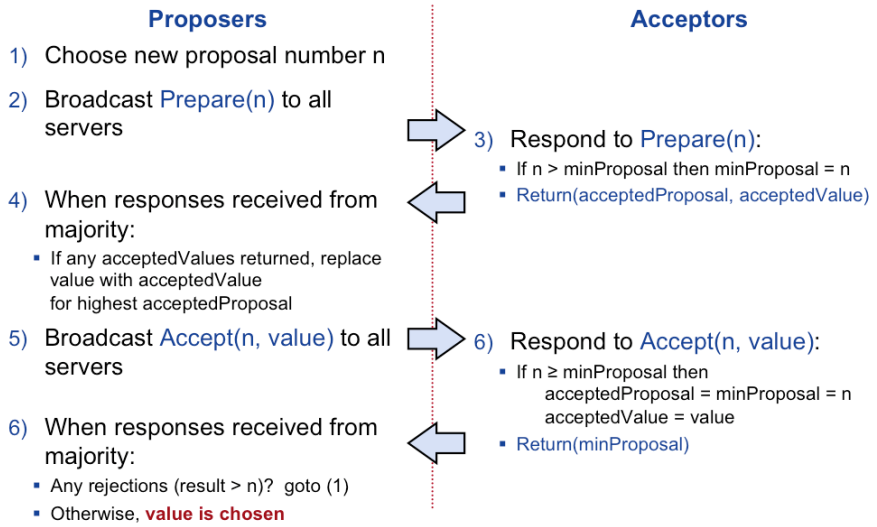
I will start by talking about Multi-Paxos, and how it makes Paxos implementable.

4.1 Multi-Paxos

The Paxos algorithm, described by the author, only works for a single value. The simplest way to use Paxos for a sequence of values, say a log, is to run the Paxos algorithm separately for each value. While this will work, it gives rise to some problems that are explained here.

4.1.1 Presence of a leader

The original Paxos allows any node in the network to be a proposer. A definite improvement would be to select a leader and allow only the leader to act as the proposer while the rest of the network acts as acceptors. Any client request would be routed to the leader and only the leader would reply to the client. Selecting a leader has two advantages:



1. Conflicts could occur in the basic Paxos protocol. A node could propose a value and wait for the value to be accepted. Before this value is accepted, another node could propose a different value. As this second value has a higher n, the nodes reject the former value. Before the second value gets accepted, the former node would realize that its proposal was rejected and so it will try again with a larger n. This cycle could go on resulting in a system lock. A single leader proposing values would prevent this situation.

2. If there were only a single leader, the first Prepare notifies the leader if any values with a higher n reside in any of the acceptors. After the first Prepare, there is no way the acceptors could have a more updated value, as the proposals originate from only the leader. This removes the need for further Prepare requests after the first Prepare, which makes the protocol much faster by effectively cutting down the number of requests by half.

4.1.2 Epoch Numbers

As Multi-Paxos runs the basic Paxos algorithm once every client request, these requests need to be numbered so the protocol knows which request the RPCs (Prepare and Accept calls) are referring to. Each request is given a unique epoch number that is constantly increasing. Having a unique number also helps in solving some other issues:

1. Ordering: Since the leader can accept multiple requests from different clients concurrently, the order in which the state machines apply those requests are decided by the epoch number.
2. Replication: The basic Paxos protocol does not talk about how values are replicated through all the nodes. It only guarantees that a value exists in a majority of the nodes. But in a real world system, replication of values is desirable. Extending the protocol with additional messages could perform replication. Epoch numbers guarantee the replication order and state.
3. Leader changes: Nodes send heartbeat messages among themselves to get notified of the current leader in a system. When a leader fails, another node takes its place after a heartbeat timeout. The new node should not execute a request that has already been executed. Every node stores the epoch number of the last executed request and this gives the system "exact one" semantics.

4.1.3 Cluster membership

The basic Paxos protocol defines a committed value as one that is present in a majority of the nodes. Real world systems require a constant addition and removal of nodes from a system. This would break the basic Paxos protocol if done arbitrarily.

Paxos describes theoretically how this problem can be solved. This algorithm ignores practical issues, and a proof of correctness is missing from the paper [3].

4.2 Raft

Raft is as efficient as Multi-Paxos and effectively gives the same result, but is designed to be more understandable and practical [1]. Described below are some of the differences between Multi-Paxos and Raft, and why they make the protocol clearer and easier to implement.

4.2.1 Node states

Paxos has two node states, proposers and acceptors. Multi-Paxos requires one proposer (called leader). Raft divides nodes further into three states:

1. Leader: Like Multi-Paxos, only leaders respond to client requests. Also similarly, there can be only one leader at a time. Each request corresponds to a term, which is similar to the epoch number in Multi-Paxos. Raft also has a single leader for a term, similar to Multi-Paxos.

2. Candidates: Raft calls nodes that could become a leader as candidate nodes. Leaders are selected among the candidates by an election. An election takes place at the beginning of a term. This is in contrast with Multi-Paxos, which is unclear about how leaders are selected when a leader fails.

3. Followers: Followers are equivalent to Acceptors in Paxos.

This division between node states and the clear definition of when a node changes states removes the ambiguity in the Multi-Paxos leader choice. This includes an explanation of how leader choice affects the safety requirements. The client protocol with respect to leader change is also defined.

The RPC calls in Raft were designed for a log rather than for a single value. So Raft does not have the obsolete calls that Multi-Paxos optimizes over the Paxos protocol (the unnecessary Prepare RPCs). This makes Raft as fast as Multi-Paxos by default.

4.2.2 RPC calls

The main RPC call in Raft is the AppendEntries RPC call. The leader uses the call to inform followers of new values. But the same call is also used for other purposes that needed separate calls in the Multi-Paxos protocol. This makes Raft more efficient.

1. Heartbeat: While Multi-Paxos had separate heartbeat RPCs, in Raft the AppendEntries call doubles up as a heartbeat message. When other nodes have not heard of a leader for a specific time, they initiate an election for a new leader.

2. Log replication: Log replication was done in Multi-Paxos through a non-trivial extension to the protocol. In Raft, the AppendEntries RPC has the last value and term number that is used to continuously guarantee full replication on all nodes.

4.2.3 Client Protocol

The client protocol in Raft is the same as Multi-Paxos. The clients embed an equivalent of epoch number in Raft that guarantees "exactly once" semantics. Again, while this was ambiguous in the Paxos protocol (Google came up with their own implementation for Chubby), it is clearly defined in the Raft protocol.

4.2.4 Configuration changes

When dealing with configuration changes in Paxos, Google faced non-trivial problems like disk failures that were not explained in Paxos.

Raft proposes a proven, two phase algorithm for dealing with configuration changes [10]. The change is done in switching between three states of the system: an old state, where majority is decided based on the old configuration, a new state, where it depends on the new configuration, and an in between joint consensus phase, where majority depends on both the old and the new configuration.

5. EVALUATION

From the discussion, the two advantages of Raft are obvious.

5.1 Understandability

Raft simplifies the node states and the RPC calls. Also, parts of Multi-Paxos that was left to the readers to figure out themselves, which led to non-trivial extensions to the core algorithm, are clearly defined in Raft.

The authors of Raft ran a user study to quantify this metric. They found statistically significant results that a majority of participants not only found Raft easier to understand and got higher test scores; they also found Paxos more understandable with prior experience in Raft.

5.2 Implementation

The same user study also proved readers find Raft easier to implement. This is because Raft was not designed to be a theoretical protocol or a proof of concept like Paxos was. The authors themselves implemented Raft and that made sure the problems that could potentially arise in a real world system were solved.

Readers also had a reference implementation to work with (Paxos did not have an open source implementation of the algorithm for a real world system even for years after the paper was published). This led to a huge number of open source implementations of Raft even though the algorithm is relatively new.

5.3 Correctness

While the core algorithm of Paxos was proven correct, the extensions in Multi-Paxos like full replication and Membership changes were not proven correct. Raft has proven these correct, as they are a part of the core protocol in Raft [1].

6. CONCLUSION

Through simple and coherent structure, Raft has proven itself to be an easier, faster and more practical alternative to Paxos.

7. REFERENCES

[1] In Search of an Understandable Consensus Algorithm. https://ramcloud.stanford.edu/raft.pdf

[2] Paxos made simple. http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf

[3] Paxos Made Live - An Engineering Perspective http://www.cs.utexas.edu/users/lorenzo/corsi/cs380d/papers/paper2-1.pdf

[4] ZooKeeper: Wait-free coordination for Internet-scale systems http://www.bruzgys.eu/files/ZooKeeper%20-%20Wait-free%20coordination%20for%20Internet-scale%20systems.pdf

[5] https://cwiki.apache.org/confluence/display/ZOOKEEPER/Zab+vs.+Paxos

[6] Replication in the Harp File System http://pdos.csail.mit.edu/6.824-2004/papers/harp.pdf

[7] Vive La Difference: Paxos vs. Viewstamped Replication vs. Zab https://www.cs.cornell.edu/fbs/publications/viveLaDifference.pdf

[8] Bitcoin: A Peer-to-Peer Electronic Cash System https://bitcoin.org/bitcoin.pdf

[9] https://ramcloud.stanford.edu/~ongaro/userstudy/paxos.pdf

[10] https://ramcloud.stanford.edu/~ongaro/userstudy/raft.pdf