Raft

Raft Background

A consensus algorithm that displays fault tolerance! Used to build a replicated log, such that each node will eventually have the exact same log. As a result, this can be used to achieve total order broadcast. Many similarities to Multi-Paxos.

Op	A	B	C
Term	1	1	2

	A	B	C
	1	1	2

	A	B	
	1	1	

Review Terms

Quorum: A majority of the nodes in the cluster - it is impossible for two different quorums of nodes to make conflicting decisions about anything as they must overlap in at least one node

Fencing Token: An increasing number used in situations like split brain in order to resolve a conflict (higher token number wins), in Raft we will call this the term number

Leader: A single node through which all writes are sent and propagated - if the leader fails we must find a way to switch to a new leader

Raft Overview

- One leader sends all writes to the follower nodes
- Once a leader receives responses from a majority of nodes accepting the write, it can tell them to commit said write to the above database layer
- If a leader is presumed to be dead, a follower node will begin an election to become a new leader with a higher term number, and will only consider itself the new leader if it receives responses from a majority of nodes

Leader Election

In Raft, each follower node periodically receives “heartbeats” from the leader to inform the follower that the leader is still up and running.

If a follower node does not receive a heartbeat within some timeout, it will assume the leader to be dead and declare itself to be a candidate and start a new leader election process, where the candidate votes for itself (increase internal term number by 1).

So that we don’t have every follower starting new elections at the same time (because then no one can get a majority of votes), the heartbeat timeout is randomized over a reasonable range on each follower.

Leader Election Continued

At this point, the other nodes will be informed that an election is ongoing:

- If the candidate term number is higher than the local term number, they once again become a follower (even the current leader) and change their local term number to the candidate term number
- If the candidate log is more up to date than the local log and the node hasn’t voted for any other candidate this term, the node will reply saying that it will vote for the candidate
- Else, it replies saying it does not vote for the candidate

On receiving these vote responses, the candidate node will keep track of the set of nodes that have voted for it for its local term, and if the number of votes reaches a majority, will declare itself the leader!

Broadcasting Messages

When the leader receives a message to broadcast, add it to the local log (don’t commit it yet), and send it to all the other nodes via some function which I will for now call ReplicateLog.

Additionally, periodically call the ReplicateLog function for each node to act as a heart beat, keep logs in sync, and also alert other nodes to commit messages that should be committed to the database layer.

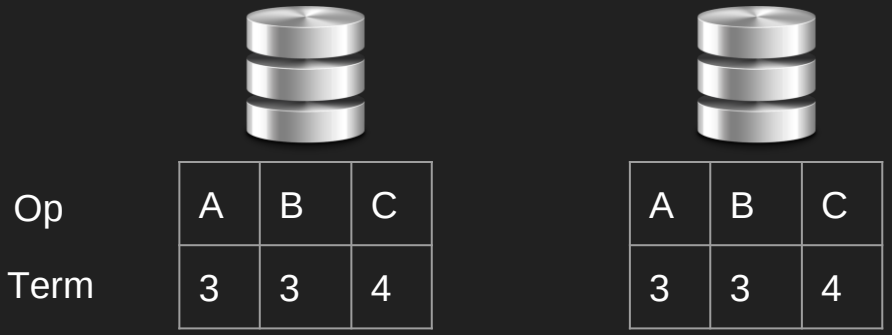
Replicate Log Function

For each node, leader keeps track of how many messages it has sent. It uses this to try and split its local log into a prefix and suffix, where the prefix is all the local messages that the remote node has already seen, and the suffix is all the local messages that need to be appended to the remote log. It then sends over the suffix messages. It also sends over the term number of the last message in the prefix.

Note: the leader may be wrong here about what the prefix and suffix are and have to readjust later, I'll explain soon!

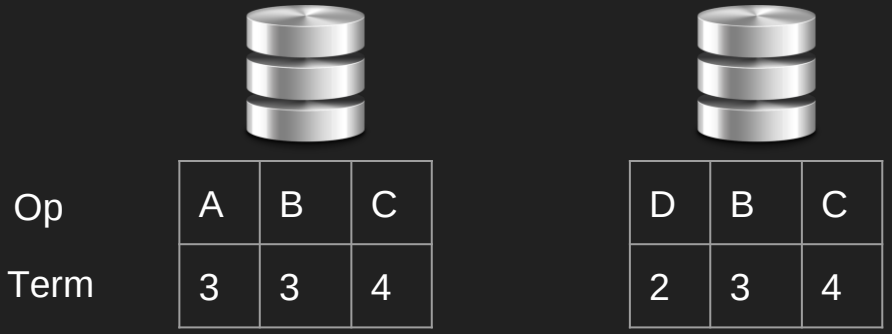
Prefix Invariant

Raft Invariant: If the logs on two replicas have the same term number at the same index in the log, they MUST be the same up to and including that index.



Prefix Invariant

Raft Invariant: If the logs on two replicas have the same term number at the same index in the log, they MUST be the same up to and including that index.



Not possible: if the first log had (D, 2) in index 0, the leader would not have been able to append (B, 3) to the follower node because the entries at index 0 would have different term numbers, and as a result the leader would have to change its guess for the prefix for the follower node and overwrite that entry of the follower. We know the leader is right here because the only reason that it was elected in the first place was that it was equal to or more updated than a majority of nodes, which means that the leader itself must be up to date. (I know this is a lot, just focus on the invariant)

Replicate Log Function Continued

Because of the invariant, if a follower node has the same term number at the end of its prefix as the leader, we can just go and copy over the rest of the suffix nodes because the follower was up to date (overwriting any log entries from a different term)! Respond to the leader saying that the new messages were acknowledged. Additionally, check for newly committed messages from the leader and commit those.

If the prefixes of the logs were not the same (or the follower has a higher term number than the leader), reject the write and report back.

Receiving Write Acknowledgements on the Leader

If the leader receives word from a follower that a write was successful, it keeps track of this and waits for a quorum of positive responses to commit the message and subsequently alert a follower.

If it hears that a write was not successful, it must have been because the prefixes of the leader and follower node were not the same, and as the result it must try the write again with a smaller prefix (and overwrite more incorrect entries on the follower log). Also possible that the follower it sent the write to now has a higher term number in which case the leader gives up being a leader and becomes follower.

Raft Conclusion

Even though the original Raft paper is from 2014, it is already hugely popular, mainly due to how easy it is to understand!

Still takes a lot of network calls, and compared to something like 2 phase commit, is good for making replicated logs but not so great for cross partition atomic transactions. Nonetheless, it is far more fault tolerant as it can support a leader failure while also being able to only write to a quorum of nodes at a time.

In a future video, I will do a comparison of Raft with other consensus algorithms, while they are generally similar, there are some subtle differences that can affect real world performance!