In Search of an Understandable Consensus Algorithm

Key Takeaways

- Raft is a consensus algorithm to implement replicated state machine.
- Compared with Paxos, Raft has the advantage of easier to be understood.
- Raft ensures the state of all machines to be the same by having a leader and multiple followers that replicate the leader's state. The leader controls the operation sequence of the full cluster while these followers are used to ensure cluster availability while the leader fails.
- Raft ensure the followers have the same state as the leader by having the leader broadcast the operation logs to all followers, and the implementation ensures that the followers execute the same operations as the leader.

Replicated State Machine

- In replicated state machine, state machines on a collection of servers compute identical copies of the same state and can continue operating even if some of the servers are down.
- The replicated state machine is useful in distributed systems, because the server can be implemented as a set of machines maintaining exactly the same state, so that if one machine fails the other machines can still act as server.
- In systems such HDFS, a separate replicated state machine is used to manage leader election and store configuration information.
- Replicated state machines are typically implemented using a replicated log. Each machine has identical log on the order of command it processed.
- A consensus algorithm is needed to keep the replicated logs consistent between different machines.
- Different machines might receive the log message in different order due to network delay, package loss et al, and the consensus algorithm is needed so that all machines agree on the content of the replicated log.
- The log message is processed by the consensus module which chooses the state of the log, then write the log to their own logs. In this way, the logs from each machine are always consistent with each other.

Problem with Paxos

- Paxos first defines a protocol capable of reaching agreement on a single decision, such as a single replicated log entry. We refer to this subset as single-decree Paxos.
- Paxos then combines multiple instances of this protocol to facilitate a series of decisions such as a log (multi-Paxos).
- One drawback of Paxos is it is extremely difficult to understand.
- The other drawback of Paxos it does not provide a good foundation for building practical implementations. There is no widely agreedupon algorithm for multi-Paxos.

Techniques for Creating Understandable Algorithms

- Problem decomposition: The problems are divided into separate pieces that could be solved, explained, and understood relatively independently whenever possible.
- Simplify the state space: reduce the size of state space and make the system more coherent and eliminate nondeterminism where possible. For example, logs are not allowed to have holes, which is a smaller state space compared with Paxos's state space that the log can be arbitrary.

Raft Consensus Algorithm - Basics

- A leader is chosen first from all machines. The leader is given full responsibility to manage all logs.
- The leader accepts log entries from clients, replicates them on other servers, and tells servers when it is safe to apply log entries to their state machines.
- A leader can fail, at which time a new leader is elected.
- During the time the leader replicates the logs to the cluster, the leader must be able to force the other logs to agree with its own log.
- There is always exactly one leader at a time.
- All other machines are followers. They are passive and only respond to the requests from the leader.
- When the leader fails, a follower is converted to candidate to be selected as new leader.
- The time is divided into terms with arbitrary length. Term is used as a logical clock and allow servers to detect obsolete information such as stale leaders.
- Each term is started with an election to select a candidate as the leader. If no candidates get enough vote due to the existence of multiple candidates, then the current term ends without leader, and another term with a new election starts.
- Raft guarantee there is at most one leader for each term.
- The term is used to detect obsolete information: the term observed for each server are exchanged during communication. If a server receives a message containing larger term, then it updates its own term. If a leader observe a term larger than its own, then it converts into follower immediately. If a server receives a message with stale term, then this message is ignored.

Leader Election

- Current leader sends periodic heartbeat information to followers to maintain its authority.
- If a follower does not receive heartbeat for a long time, then it assumes the leader is dead and starts an election.
- At this time, this follower increases its term by 1, converts itself to candidate, and sends message to other followers to start election and it votes for itself.
- A candidate wins the election if it receives majority vote from all servers. Then it sends a heartbeat signal to other servers to establish its authority.
- Each server votes at most one candidate, on a first-come-first-serve principle.
- If a candidate receives a heartbeat signal during election, and the heartbeat signal has a term equal or large than this server's term, then this candidate recognize this heartbeat as legitmate and recognize this server as leader. Otherwise the candidate rejects the signal and remains in candidate state.
- If no candidates get a majority of vote, for example when many followers are converted into candidates independently at close time and each gets the vote from some followers. In that case, a new term begins and the election restarts.
- In order to prevent repeated restart of election due to split of votes, each candidate has a random timeout after which it restarts the election. In this case, at each time there is a single candidate that restarts the election, and it will win the election and send heartbeat signals out before other candidates restart the election.

Log Replication

- When a client requests the leader to execute some command, the leader appends this command to its log, then asks the followers to append this command to their logs.
- After the log entry has been added to the log of a majority of followers, the leader considers it safe to commit the command, then it executes the command and return the result to the client. Committing an entry also commits all entries before this entry.
- If some followers do not respond to the append log request, the leader keeps retrying until all followers append this entry to their logs.

- Each log entry contains the command, and also the term number when the command is received, and an index about the position of the entry in the log.
- When the leader in the future asks the followers to append an entry, it also includes the highest index of the entry that has been committed, so the followers can know which entries have been committed.
- Raft ensures if two entries in different logs have the same index and term, then they store the same command.
- Raft ensures if two entries in different logs have the same index and term, then the logs are identical in all preceding entries.
- Possible discrepancy between logs of leader and follower can be caused when leader crashes before it synchronizes log with the follower. A follower may be missing entries that are present on the leader because the leader crashes before sending the log entry out, It may have extra entries that are not present on the leader because some packs might be duplicated and such extra entries are not committed, or both. Missing and extraneous entries in a log may span multiple terms because the leader can crash, get reelected, then crash again.
- The log consistency is achieved by forcing the followers to copy the log of the leader. So a conflicting log will result in the follower's log get overwritten by the leader.
- During the AppendEntries RPC call sent from the leader to the followers, the goal is to have the follower append a new entry to their log. The message also contains the last log before the current log entry, which is used to consistent check. If the check fails, the log position is moved to the previous log, and this continues until the consistent point is found. Then the leader sends all the log from this point to the latest log.
- The leader only appends in its log, and never overwrites or deletes entries.

Safety

- Raft guarantees that all the committed entries from previous terms are present on each new leader from the moment of its election, without the need to transfer those entries to the new leader.
- Raft uses the voting process to prevent a candidate from winning an election unless its log contains all committed entries.
- This is ensured by the vote mechanism: The RequestVote RPC includes information about the candidate's log. If the voters finds its own log is more up-to-date than the candidate's log, it denies the vote. Here the up-to-date comparison is done by comoparing the index and term of the last entries in the logs.
- Raft only commit logs from the leader's current term, which leads to indirectly committing logs from older terms, but does not try to directly commit logs from older terms by replicating these logs on followers.

Follower and Candidate Crashes

- If a follower or candidate crashes, then future RequestVote and AppendEntries RPCs sent to it will fail.
- Raft handles these failures by retrying indefinitely: if the crashed server restarts, then the RPC will complete successfully. If a server crashes after completing an RPC but before responding, then it will receive the same RPC again after it restarts.
- It is OK for the same message to be received twice, because Raft RPCs are idempotent.

Timing and Availability

- Safety must not depend on timing: the system must not produce incorrect results just because some event happens more quickly or slowly than expected.
- Raft will be able to elect and maintain a steady leader as long as the system satisfies the timing requirement: broadcastTime << electionTimeout << MTBF where broadcastTime is the average time it takes a server to send RPCs in parallel to every server in the cluster and receive their responses,

electionTimeout is timeout for each candidate to wait before they start the next round of election, MTBF is the average time between failures for a single server.
- The broadcast time should be an order of magnitude less than the election timeout so that leaders can reliably send the heartbeat messages required to keep followers from starting elections.
- The election timeout should be a few orders of magnitude less than MTBF so that the system makes steady progress.

Dynamic Configuration Change

- The configuration change refers to things like replace servers when they fail or to change the degree of replication.
- For the configuration change mechanism to be safe, there must be no point during the transition where it is possible for two leaders to be elected for the same term. However, this is impossible, as it is atomically switch all of the servers at once.
- In Raft, the configuration change first changes the configuration to a transitional configuration called joint consensus, then to the final configuration after the joint consensus has been committed.
- At the beginning of configuration change, the leader broadcasts the joint configuration which is a combination of old and new configuration as a new log entry to all machines, and when the majority of the machines receive the new log the joint configuration is regarded as committed. The machines that receives the log starts their transition to the joint consensus configuration.
- After the joint consensus configuration is stored, the leader now creates the log message containing the new configuration and broadcasts to all machines, here all machines start to transition to the new configuration.

Performance of Raft Algorithm

- The most important case for performance is when an established leader is replicating new log entries. Raft achieves this using the minimal number of messages (a single round-trip from the leader to half the cluster).
- The leader election time is measured by repeatedly crashed the leader of a cluster of five servers and timed how long it took to detect the crash and elect a new leader.
- The measured election time shows a small amount of randomization in the election timeout is enough to avoid split votes in elections, and higher randomness improves the election time.
- The election time can be reduced by reducing the election timeout, which leads to more rapidly restarting the election. However, there is a lower limit of the election time to make the system safe, as we need the election time to be much larger than heartbeat broadcast time.