Building a Fault-Tolerant Online Store Using Raft Consensus Algorithm

Raft was designed as a replacement to Leslie Lamport's Paxos consensus algorithm. The goal was to break down the features of Paxos into modularized and easily recognizable parts. The features of Raft that are unique to it are its strong leader where in the leadership of a server is more important than in other consensus algorithms. An example is that log entries only flow from the leader to the other servers which simplifies the management of the log replication. Raft also uses randomized timers in its leader election. This assures that several servers are unlikely to turn into candidates for election at the same time. Additionally, Raft breaks down periods of time in the algorithm into arbitrary units of time called terms. Terms can change if a leader dies, a follower becomes a candidate, or no candidate wins an election. Raft also decomposes the consensus problem into three independent subproblems: leader election, log replication, and safety. For this algorithm to work, a majority of the servers must stay online.

Leader election is carried out at the very beginning. After a leader is elected, it sends out periodic heartbeats to all the followers, so that they recognize it is still alive. If a follower does not receive a heartbeat in the period that it's randomized timer set, it will become a candidate and start a new election. A candidate remains a candidate until it or another server wins the election or a period of time goes by with no winner. If no server wins the election, then the candidate increments its term, and starts a new election. If a leader or a candidate are operating normally and receive a message from a server with a higher term, then they update their term to that number and revert to follower status.

Once a leader has been elected it sends a notification to every other server in the cluster. It will then start accepting requests from clients. It stores the request in its log and then sends sends the updated log to all the followers. These followers will overwrite any wrong entries that they have in their logs and continue to add to their logs as the leader updates them from then on out. Once a majority of the servers have accepted the leader's updated log, the leader commits the changes to its log and applies it to its state machine. Then every heartbeat the leader sends out after that will also tell followers to apply the updates to their state machine. This is how client requests get serviced and become permanent.

The safety of the algorithm is enforced by ensuring that a majority of servers need to agree on a value before it is applied to the state machine. There are situations where two leaders could be operating in the algorithm such as in the situation where a leader has not noticed its followers had stopped receiving its heartbeats and elected a new leader. However, this is not an issue because a faulty leader cannot make any permanent changes since it needs a majority of servers to agree with it. However, these servers do not recognize it as the leader and will ignore any messages from it. Therefore, only one server can be applying permanent changes to the state machine at any given time.