Leader election in Raft

Alessandro Monticelli, Jóhannes Helgi Tómasson, Jonas Stahl April 3, 2026

1 Implementation of the Election Algorithm

1.1 System Initialization and Heartbeats

In the Raft consensus algorithm, servers operate in one of the following states [1, 2]:

- Follower
- Candidate
- Leader

All servers are initialized as Followers [1, 2]. A Leader maintains its authority by broadcasting periodic heartbeat messages, through empty `AppendEntries` RPCs [1, 2].

1.2 Initiating an Election

If a Follower does not receive a heartbeat within a specific timeframe, known as the *"election timeout"*, it assumes the Leader has failed [1, 2]. At this point, the Follower transitions to the Candidate state, increments its current `term number`, votes for itself, and broadcasts `RequestVote` RPCs to the other servers in the cluster [1, 2].

1.3 Processing Vote Requests

Normally, a `RequestVote` RPC contains the Candidate's `lastLogIndex` and `lastLogTerm` to allow voters to determine which Candidate's log is more up-to-date [1]. However, the assignment description specifies that *"all AppendEntries RPC carry empty log messages and no node ever appends anything to its log"*. In this particular case, the evaluation of the `RequestVote` RPC will be based only on the `term number`, rejecting the request if the Candidate's term is lower than its own, and giving the vote if it has not already voted in the current term [2].

2 Election Mechanics with Empty Logs

So in this case the Leader election follows a first-come-first-served logic, based on `term numbers` [2]. The Raft protocol dictates that a Candidate may only be elected if its log is at least as up-to-date as any other log in that majority [1]. In our example, the logs are empty, so they are identical, it follows that this condition is automatically respected.

2.1 Securing Leadership

As said, a Follower will grant its vote to the first Candidate that requests it during a new term [2]. Upon securing votes from a majority of the servers, including its own vote, a Candidate promotes itself to the Leader state, it then immediately dispatches heartbeat messages to assert its Leadership and suppress further elections [2]. If a Candidate receives a heartbeat from an entity claiming to be Leader with a term number equal to or greater than its own, it acknowledges the Leader and reverts to the Follower state [2].

3 Vulnerability to Election Failures

3.1 Split Votes

Despite the simplified state of the logs, elections can still fail due to split votes [2]. If multiple Followers experience an election timeout simultaneously, they will transition into Candidates, increment their term numbers, and cast votes for themselves at the exact same time [2]. Because the remaining servers allocate their votes on a first-come-first-served basis, the votes may be distributed in such a way that no single Candidate achieves the requisite majority [2].

3.2 Mitigation with Randomized Timeouts

This results in a failed election and a stalemate that persists until a Candidate timeout occurs [2]. At this point, the Candidates increment their term numbers and initiate a subsequent election round [2]. To mitigate the risk of indefinite split-vote loops, Raft employs randomized election timeouts per-server [2]. By assigning timeouts randomly from a fixed interval, such as 150 to 300 milliseconds, the protocol ensures that it is highly probable for one server to time out before its peers [2]. This early Candidate can then secure the majority vote and broadcast heartbeats to establish its Leadership before a split vote can manifest [2].

References

[1] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association.

[2] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, 2nd edition, 2007. 686 pages.