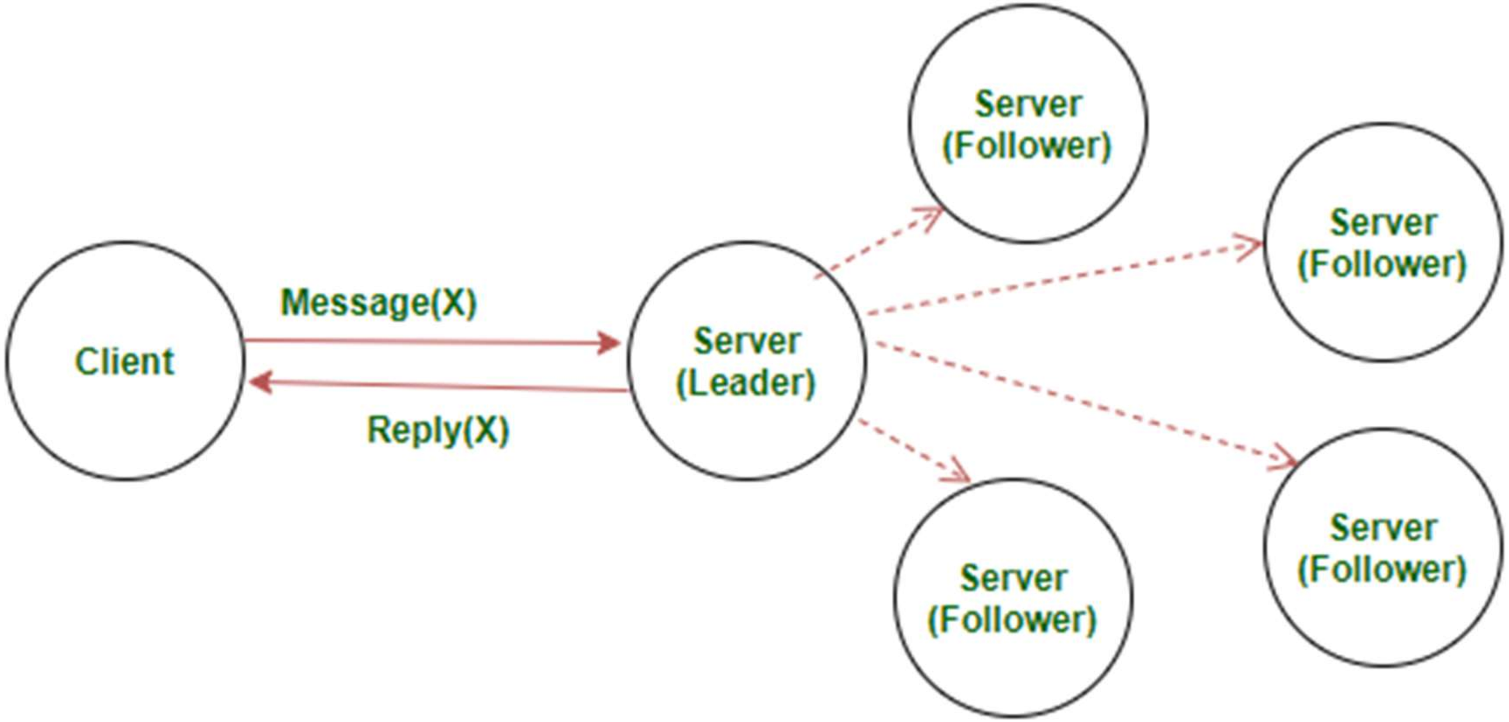
KOENIG
step forward

The Raft Consensus Algorithm

Interactive Guide

- http://thesecretlivesofdata.com/raft/
  - - Let's play around with this interactive guide to understand.

What is RAFT?


Client
Message(X)
Reply(X)
Server
(Leader)
Server
(Follower)
Server
(Follower)
Server
(Follower)
Server
(Follower)

What is Consensus?

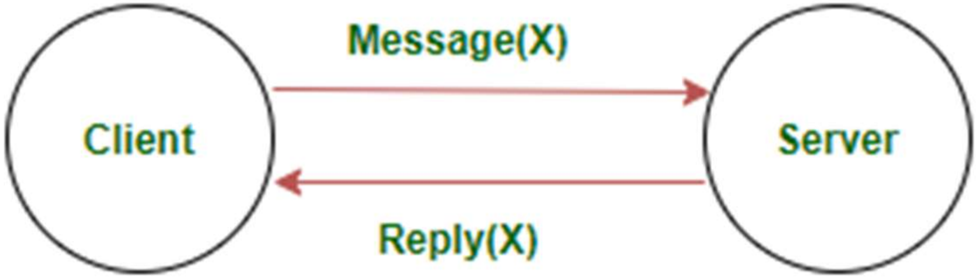
- Which problem Raft protocol tries to solve?
  - That is achieving Consensus
- Consensus means multiple servers agreeing on same information

Client server interaction Process

- The client sends a message to the server and the server responds back with a reply.
- A consensus protocol tolerating failures must have the following features :
  - Validity : If a process decides(read/write) a value, then it must have been proposed by some other correct process
  - Agreement : Every correct process must agree on the same value
  - Termination : Every correct process must terminate after a finite number of steps.
  - Integrity : If all correct processes decide on the same value, then any process has the said value

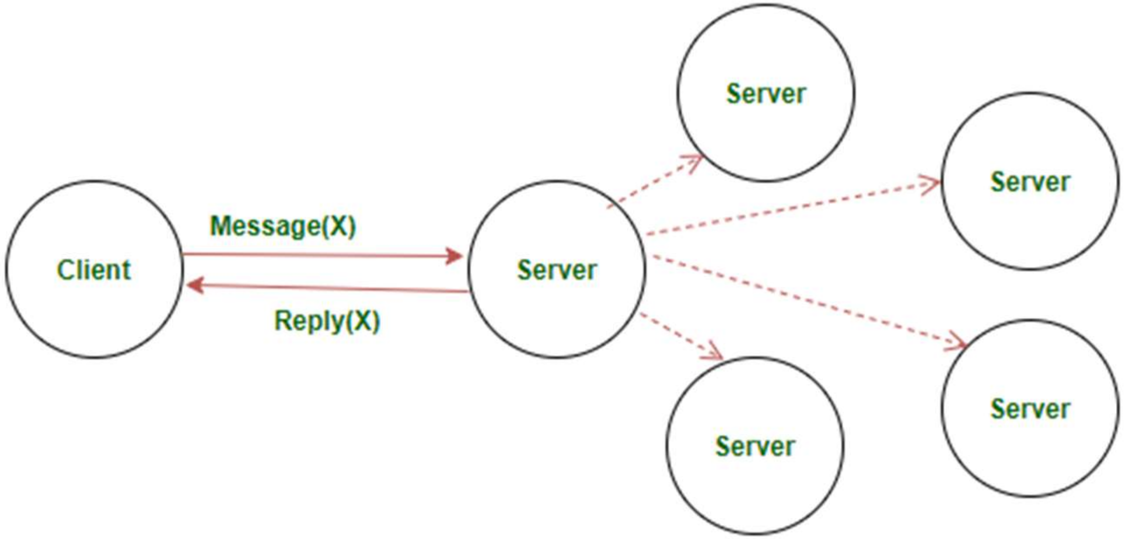
Single Server system

- The client interacts with a system having only one server with no backup
- There is no problem in achieving consensus in such a system.



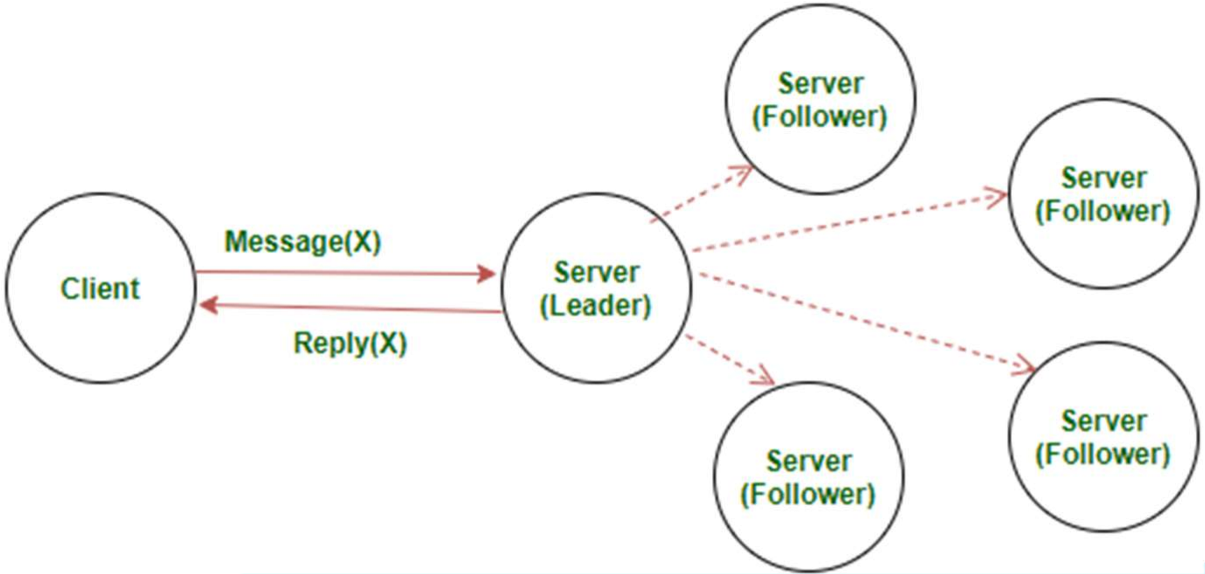
Multiple Server system

- The client interacts with a system having multiple servers
- Such systems can be of two types :
  - Symmetric :- Any of the multiple servers can respond to the client and all the other servers are supposed to sync up with the server that responded to the client's request, and
  - Asymmetric :- Only the elected leader server can respond to the client. All other servers then sync up with the leader server.
- Given below is an example of an asymmetric multiple server system



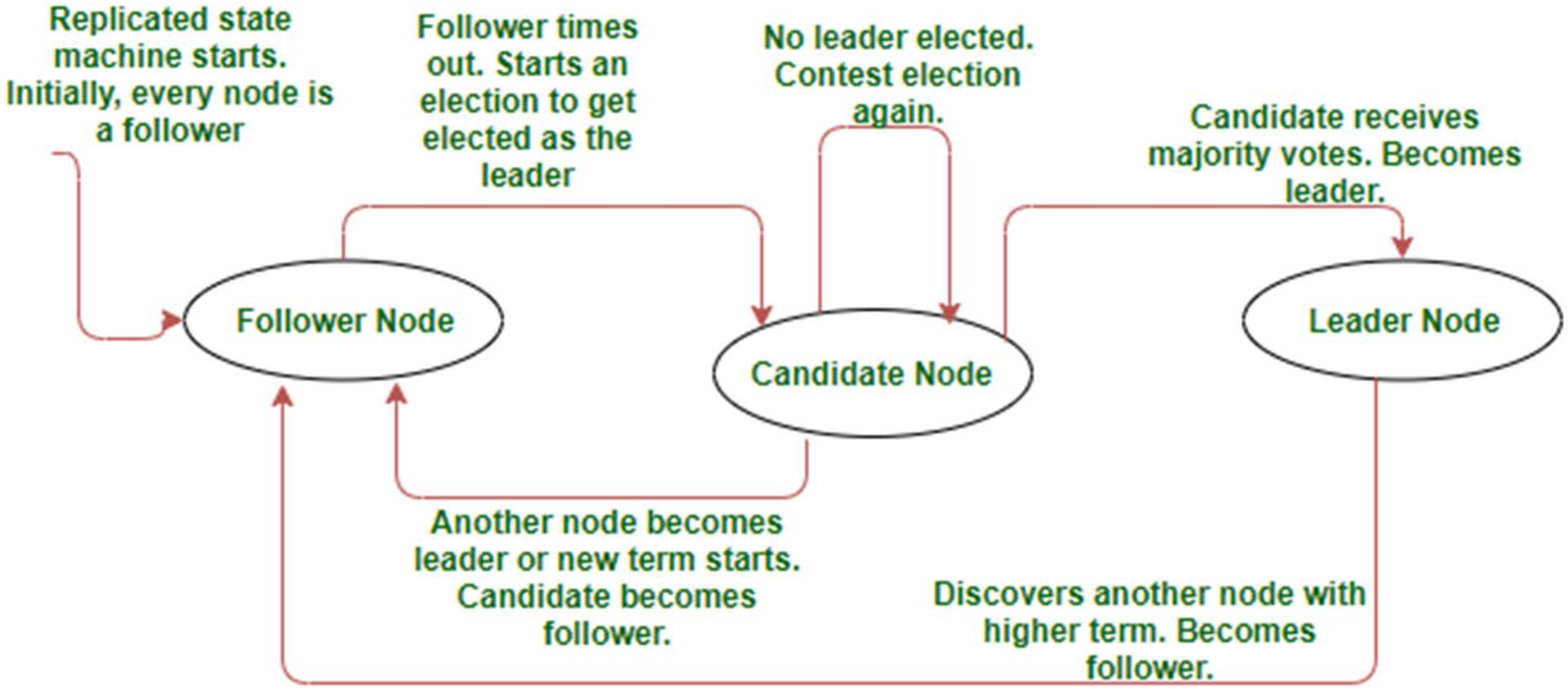
Replicated state machine

- System in which all the servers maintain similar data across time
- Terms used to refer individual servers in a distributed system
  - Leader – Only the server elected as leader can interact with the client
  - Follower – Follower servers sync up their copy of data with that of the leader’s
  - Candidate – At the time of contesting an election to choose the leader server, the servers can ask other servers for votes. Hence, they are called candidates when they have requested votes.



What is the Raft protocol

- A consensus algorithm
- Raft states that each node in cluster can stay in any of the three states, namely, leader, candidate, follower



Replicated state machine starts. Initially, every node is a follower

Follower times out. Starts an election to get elected as the leader

No leader elected. Contest election again.

Candidate receives majority votes. Becomes leader.

Another node becomes leader or new term starts. Candidate becomes follower.

Discovers another node with higher term. Becomes follower.

What is the Raft protocol

- To maintain these server status(es), the Raft algorithm divides time into small terms of arbitrary length
- Each term is identified by a monotonically increasing number, called term number.

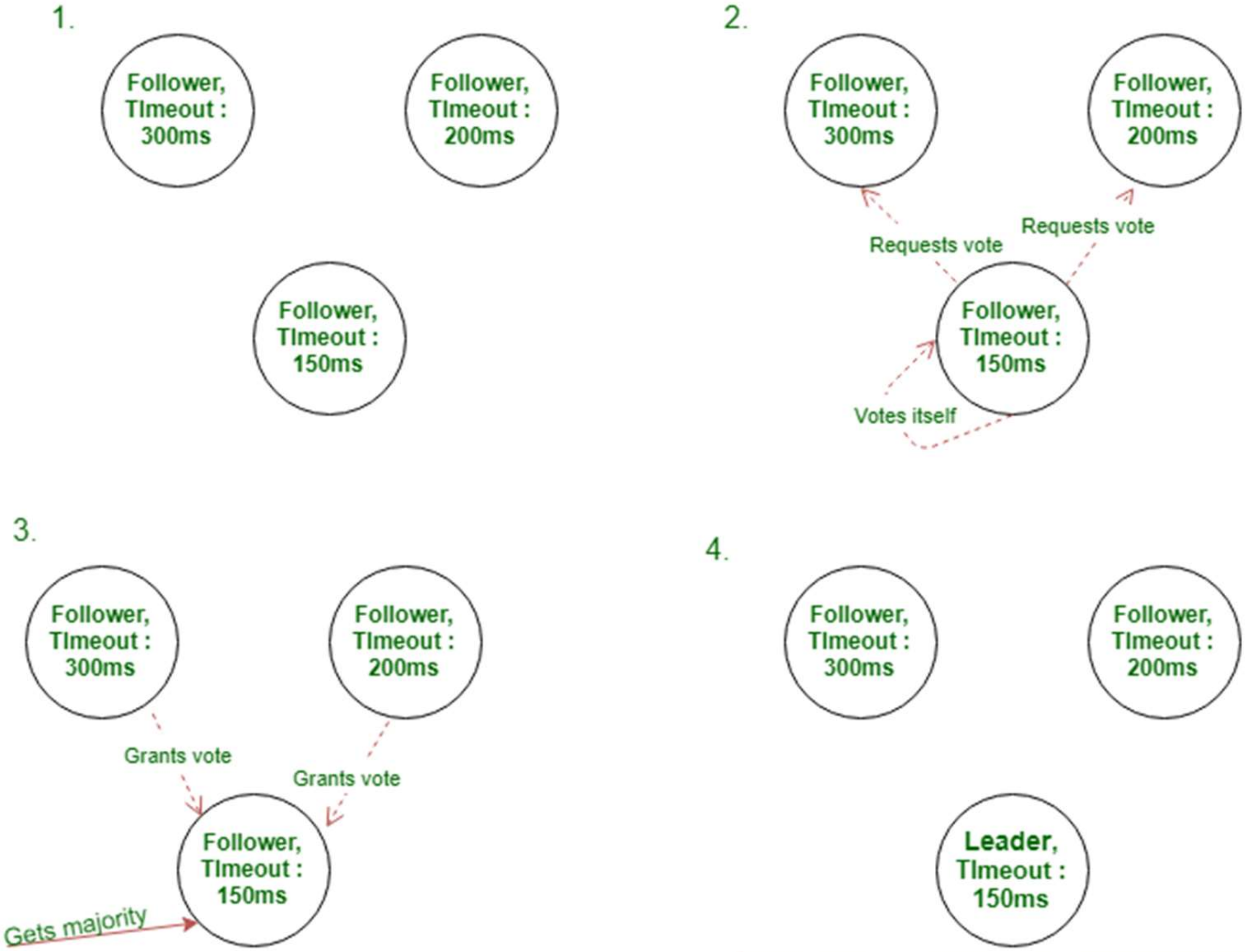
Term number

- Is maintained by every node
- Is passed while communications between nodes
- Every term starts with an election to determine the new leader
- The candidates ask for votes from other server nodes(followers) to gather majority
- If the majority is gathered, the candidate becomes the leader for the current term
- If no majority is established, the situation is called a split vote and the term ends with no leader
- Hence, a term can have at most one leader

Leader election

- Leader node sends heartbeat to express dominion to other Follower nodes
- A leader election takes place when a Follower node times out while waiting for a heartbeat from the Leader node
- At this point of time, the timed out node
  - Changes it state to Candidate state,
  - Votes for itself and
  - Issues RequestVotes RPC to establish majority and attempt to become the Leader

Leader election



1.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Follower, TImeout : 150ms
2.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Requests vote
Requests vote
Follower, TImeout : 150ms
Votes itself
3.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Grants vote
Grants vote
Follower, TImeout : 150ms
Gets majority
4.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Leader, TImeout : 150ms

Log Replication

- Each request made by the client is stored in the Logs of the Leader
- This log is then replicated to other nodes(Followers)
- Typically, a log entry contains the following three information :
  - Command/Data specified by the client to execute
  - Index to identify the position of entry in the log of the node
  - Term Number to ascertain the time of entry of the command.
- The Leader node fires AppendEntries RPCs to all other servers(Followers) to sync/match up their logs with the current Leader
- The Leader keeps sending the RPCs until all the Followers safely replicate the new entry in their logs.

Log Replication

- When the majority of the servers in the cluster successfully copy the new entries in their logs, it is considered committed
- At this point, the Leader also commits the entry in its log to show that it has been successfully replicated.
- After the entry is committed, the leader executes the entry and responds back with the result to the client.
- Leader handles inconsistencies by forcing the followers' logs to duplicate its own.
- This means that conflicting entries in follower logs will be overwritten with entries from the leader's log.

Safety

- Leader election safety
  - At most one leader per term
- Log Matching safety
  - If multiple logs have an entry with the same index and term, then those logs are guaranteed to be identical in all entries up through to the given index
- Leader completeness
  - The log entries committed in a given term will always appear in the logs of the leaders
- State Machine safety
  - If a server has applied a particular log entry to its state machine, then no other server in the server cluster can apply a different command for the same log

Safety

- Leader is Append-only
  - A leader node(server) can only append(no other operations like overwrite, delete, update are permitted) new commands to its log
- Follower node crash
  - All the requests sent to the crashed node are ignored
  - Further, the crashed node can’t take part in the leader election
  - When the node restarts, it syncs up its log with the leader node

Cluster membership

- When the status of nodes in the cluster changes(cluster configuration changes), the system becomes susceptible to faults which can break the system
- So, to prevent this, Raft uses what is known as a two phase approach to change the cluster membership
- So, in this approach, the cluster first changes to an intermediate state(known as joint consensus) before achieving the new cluster membership configuration.
- Joint consensus makes the system available to respond to client requests even when the transition between configurations is taking place. Thus, increasing the availability of the distributed system, which is a main aim.