L13 QUORUMS AND FAULT TOLERANCE

2PC can only work if all notes all able to contact with each other.

In distributed system, there is CAP theory, consistency, availability and partition. Partition can never be avoided. How to proceed with read/write transactions in case where not all replicas can be contacted?

QUORUM BASED PROTOCOLS

Quorum is designed to prevent *split-brain* scenarios which can happen when there is a partition in the network and subsets of nodes cannot communicate with each other.This can cause both subsets of nodes to try to own the workload and write to the same disk which can lead to numerous problems.

Idea: Tell client that a file's version is updated after a majority of SurfStoreServers get the update

1. Form a "read quorum" of size N_R • Contact $N_R$servers and read all their versions • Select highest version as the "correct" version
2. Form a "write quorum" of size N_W • Contact N_W servers • Increment the highest version from that set • Write out that new version to the servers in the write quorum

Constraints

- N_R + N_W > N

  Drawer theory: There are N_W servers with correct version and we need to at least read $N - N_W + 1$ in order to get the correct version.

  Every read will see at least one copy of the latest value written

- N_W > N/2

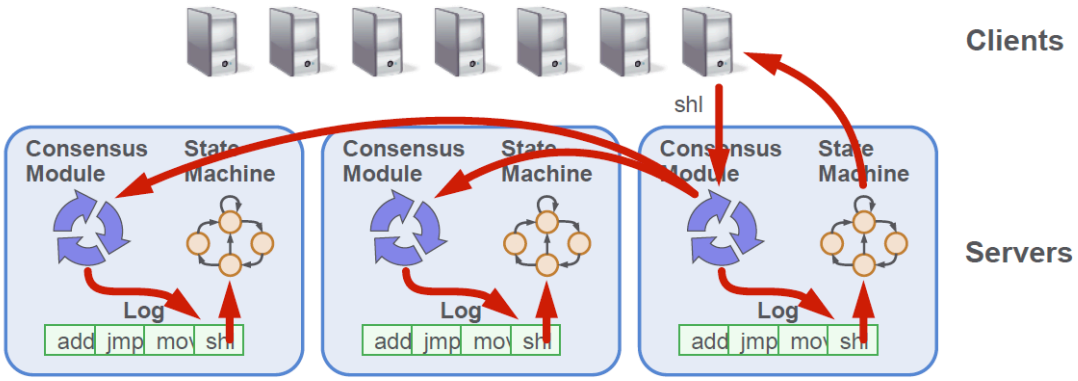
  Only one write quorum at a time can be assembled

FaultTolerence via RAFT

Replicated Logs

It means replicate sate machine ==> All machines execute in the same order

Goal: Replicated Log



- Replicated log => replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication

Server state

- Leader : handles all client interactions, log replication
- Follower : completely passive
- Candidate : used to elect a new leader

Normal operation: 1 leader, N 1 followers

Operation

AppendEntries

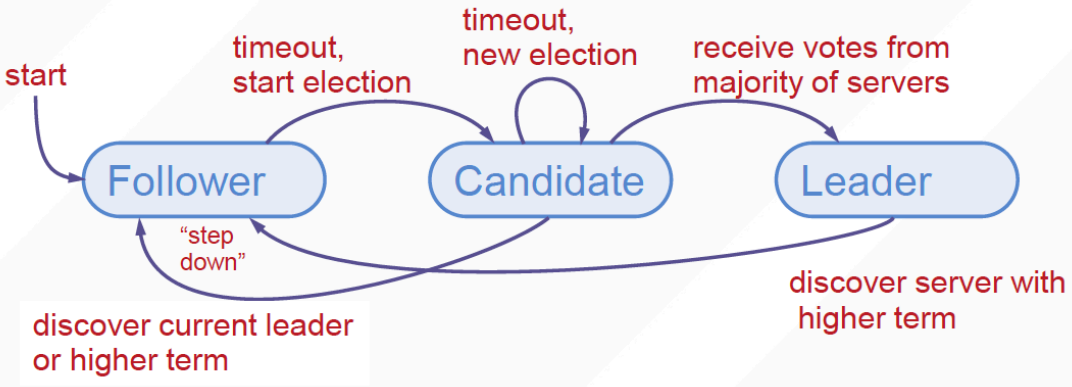
• The TC (leader) uses this to “push” new operations to the replicated state machines • Also used by the TC to tell the other nodes it is the TC/leader

RequestVote

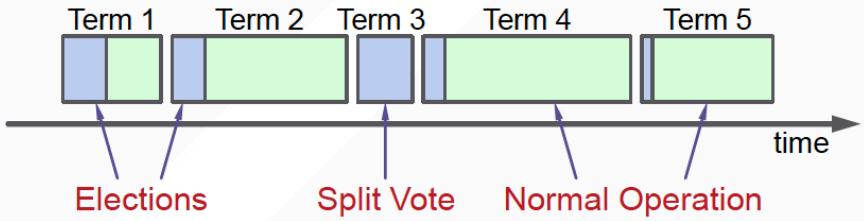
• Used when the system starts up to select a leader • Used when the leader fails to elect a new leader • Used when the leader is unreachable due to a network partition to elect a new leader

LIVENESS VALIDATION

- Servers start as followers
- Leaders send heartbeats (empty AppendEntries RPCs) to maintain authority
- If electionTimeout elapses with no RPCs (100-500ms), follower assumes leader has crashed and starts new election



TERMS (AKA EPOCHS)



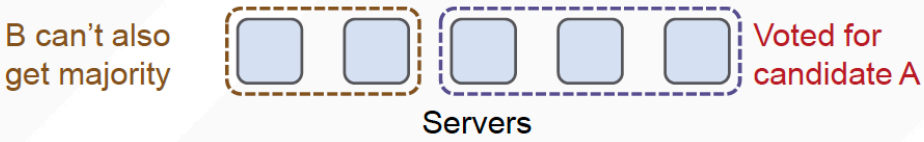
- Time divided into terms
  - Election (either failed or resulted in 1 leader)
  - Normal operation under a single leader
- Each server maintains current term value
- Key role of terms: identify obsolete information

ELECTIONS

- Start election:
  - Increment current term, change to candidate state, vote for self
- Send RequestVote to all other servers, retry until either:
  1. Receive votes from majority of servers:
     - Become leader
     - Send AppendEntries heartbeats to all other servers
  2. Receive RPC from valid leader:
     - Return to follower state
  3. No-one wins election (election timeout elapses):
     - Increment term, start new election

ELECTION PROPERTIES

- Safety: allow at most one winner per term
  - Each server votes only once per term (persists on disk)
  - Two different candidates can't get majorities in same term



- Liveness: some candidate must eventually win
  - Each choose election timeouts randomly in [T, 2T]
  - One usually initiates and wins election before others start
  - Works well if T >> network RTT