State

Persistent state on all servers:

(Updated on stable storage before responding to RPCs)

currentTerm latest term server has seen (initialized to 0

on first boot, increases monotonically)

votedFor candidateId that received vote in current

term (or null if none)

log entries; each entry contains command

for state machine, and term when entry was received by leader (first index is 1)

Volatile state on all servers:

commitIndex index of highest log entry known to be

committed (initialized to 0, increases

monotonically)

lastApplied index of highest log entry applied to state

machine (initialized to 0, increases

monotonically)

Volatile state on leaders:

(Reinitialized after election)

nextIndex[] for each server, index of the next log entry

to send to that server (initialized to leader

last log index + 1)

matchIndex[] for each server, index of highest log entry

known to be replicated on server (initialized to 0, increases monotonically)

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term leader's term

leaderId so follower can redirect clients

new ones

prevLogTerm term of prevLogIndex entry

entries log entries to store (empty for heartbeat;

may send more than one for efficiency)

leaderCommit leader's commitIndex

Results:

term currentTerm, for leader to update itself success true if follower contained entry matching

prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)

- 2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
- 3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
- 4. Append any new entries not already in the log
- If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term candidate's term candidateId candidate requesting vote

lastLogIndex index of candidate's last log entry (§5.4) term of candidate's last log entry (§5.4)

Results:

term currentTerm, for candidate to update itself voteGranted true means candidate received vote

Receiver implementation:

- 1. Reply false if term < currentTerm (§5.1)
- 2. If votedFor is null or candidateld, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

Rules for Servers

All Servers:

- If commitIndex > lastApplied: increment lastApplied, apply log[lastApplied] to state machine (§5.3)
- If RPC request or response contains term T > currentTerm: set currentTerm = T, convert to follower (§5.1)

Followers (§5.2):

- Respond to RPCs from candidates and leaders
- If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate

Candidates (§5.2):

- On conversion to candidate, start election:
 - Increment currentTerm
 - Vote for self
 - Reset election timer
 - Send RequestVote RPCs to all other servers
- · If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- · If election timeout elapses: start new election

Leaders:

- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)
- If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
 - If successful: update nextIndex and matchIndex for follower (§5.3)
 - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)
- If there exists an N such that N > commitIndex, a majority
 of matchIndex[i] ≥ N, and log[N].term == currentTerm:
 set commitIndex = N (§5.3, §5.4).

Figure 2: A condensed summary of the Raft consensus algorithm (excluding membership changes and log compaction). The server behavior in the upper-left box is described as a set of rules that trigger independently and repeatedly. Section numbers such as §5.2 indicate where particular features are discussed. A formal specification [31] describes the algorithm more precisely.