MODULE *Server*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*, *Messages*

The set of server *IDs*
CONSTANTS *Server*

Server states.
CONSTANTS *Follower*, *Candidate*, *Leader*

Server *log* entry types.
CONSTANTS *OpenSessionEntry*,
CloseSessionEntry,
CommandEntry

Message types:
CONSTANTS *PollRequest*,
PollResponse,
VoteRequest,
VoteResponse,
AppendRequest,
AppendResponse,
OpenSessionRequest,
OpenSessionResponse,
CloseSessionRequest,
CloseSessionResponse,
CommandRequest,
CommandResponse

The following variables are all per server (functions with domain *Server*).

The server's term number.
VARIABLE *currentTerm*

The server's state (Follower, *Candidate*, or *Leader*).
VARIABLE *state*

The candidate the server voted for in its current term, or
Nil if it hasn't voted for any.
VARIABLE *votedFor*

$serverVars \triangleq \langle currentTerm, state, votedFor \rangle$

The last applied index
VARIABLE *lastApplied*

All registered sessions
VARIABLE *session*

$stateVars \triangleq \langle lastApplied,\ session \rangle$

A Sequence of *log* entries. The index into this sequence is the index of the *log* entry. Unfortunately, the Sequence module defines $Head(s)$ as the entry with index 1, so be careful not to use that!

VARIABLE log

The index of the latest entry in the *log* the state machine may apply.

VARIABLE $commitIndex$

$logVars \triangleq \langle log,\ commitIndex \rangle$

The following variables are used only on followers:

VARIABLE $preVotesGranted$

$followerVars \triangleq \langle preVotesGranted \rangle$

The set of servers from which the candidate has received a vote in its *currentTerm*.

VARIABLE $votesGranted$

$candidateVars \triangleq \langle votesGranted \rangle$

The following variables are used only on leaders:
The next entry to send to each follower.

VARIABLE $nextIndex$

The latest entry that each follower has acknowledged is the same as the leader's. This is used to calculate *commitIndex* on the leader.

VARIABLE $matchIndex$

$leaderVars \triangleq \langle nextIndex,\ matchIndex \rangle$

End of per server variables.

Helpers

The set of all quorums. This just calculates simple majorities, but the only important property is that every quorum overlaps with every other.

$Quorum \triangleq \{i \in \text{SUBSET } (Server) : Cardinality(i) * 2 > Cardinality(Server)\}$

The term of the last entry in a *log*, or 0 if the *log* is empty.

$LastTerm(xlog) \triangleq \text{IF } Len(xlog) = 0 \text{ THEN } 0 \text{ ELSE } xlog[Len(xlog)].term$

Return the minimum value from a set, or undefined if the set is empty.

$Min(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s : x \leq y$

Return the maximum value from a set, or undefined if the set is empty.

$Max(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s : x \geq y$

Define initial values for all variables

$InitServerVars \triangleq$
$\land\ currentTerm = [i \in Server \mapsto 1]$
$\land\ state = [s1 \mapsto Leader,\ s2 \mapsto Follower,\ s3 \mapsto Follower]$
$\land\ votedFor = [i \in Server \mapsto Nil]$
$\land\ preVotesGranted = [i \in Server \mapsto \{\}]$
$\land\ votesGranted = [i \in Server \mapsto \{\}]$
$\land\ nextIndex = [i \in Server \mapsto [j \in Server \mapsto 1]]$
$\land\ matchIndex = [i \in Server \mapsto [j \in Server \mapsto 0]]$
$\land\ log = [i \in Server \mapsto \langle\rangle]$
$\land\ commitIndex = [i \in Server \mapsto 0]$
$\land\ lastApplied = [i \in Server \mapsto 0]$
$\land\ session = [i \in Server \mapsto [j \in \{\} \mapsto [id \mapsto Nil]]]$

Define state transitions

Server i restarts from stable storage.
It loses everything but its *currentTerm*, *votedFor*, and *log*.
$Restart(i) \triangleq$
$\land\ state' = [state \text{ EXCEPT } ![i] = Follower]$
$\land\ preVotesGranted' = [preVotesGranted \text{ EXCEPT } ![i] = \{\}]$
$\land\ votesGranted' = [votesGranted \text{ EXCEPT } ![i] = \{\}]$
$\land\ nextIndex' = [nextIndex \text{ EXCEPT } ![i] = [j \in Server \mapsto 1]]$
$\land\ matchIndex' = [matchIndex \text{ EXCEPT } ![i] = [j \in Server \mapsto 0]]$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = 0]$
$\land\ \text{UNCHANGED } \langle messages,\ currentTerm,\ votedFor,\ stateVars,\ log\rangle$

$TimeoutFollower(i) \triangleq$
$\land\ state[i] = Follower$
$\land\ preVotesGranted' = [preVotesGranted \text{ EXCEPT } ![i] = \{\}]$
$\land\ \text{UNCHANGED } \langle messages,\ serverVars,\ stateVars,\ candidateVars,\ leaderVars,\ logVars\rangle$

Server i times out and starts a new election.
$TimeoutCandidate(i) \triangleq$
$\land\ state[i] = Candidate$
$\land\ currentTerm' = [currentTerm \text{ EXCEPT } ![i] = currentTerm[i] + 1]$
$\land\ votedFor' = [votedFor \text{ EXCEPT } ![i] = Nil]$
$\land\ votesGranted' = [votesGranted \text{ EXCEPT } ![i] = \{\}]$
$\land\ \text{UNCHANGED } \langle messages,\ state,\ stateVars,\ followerVars,\ leaderVars,\ logVars\rangle$

Follower i sends j a pre-vote request.
$RequestPreVote(i,\ j) \triangleq$
$\land\ state[i] = Follower$
$\land\ Send([mtype \mapsto PollRequest,$

$mterm \mapsto currentTerm[i],$
$mlastLogTerm \mapsto LastTerm(log[i]),$
$mlastLogIndex \mapsto Len(log[i]),$
$msource \mapsto i,$
$mdest \mapsto j])$
$\land$ UNCHANGED $\langle serverVars, stateVars, followerVars, candidateVars, leaderVars, logVars\rangle$

Candidate i sends j a *RequestVote* request.

$RequestVote(i, j) \triangleq$
$\land state[i] = Candidate$
$\land Send([mtype \mapsto VoteRequest,$
$mterm \mapsto currentTerm[i],$
$mlastLogTerm \mapsto LastTerm(log[i]),$
$mlastLogIndex \mapsto Len(log[i]),$
$msource \mapsto i,$
$mdest \mapsto j])$
$\land$ UNCHANGED $\langle serverVars, stateVars, followerVars, candidateVars, leaderVars, logVars\rangle$

Leader i sends j an *AppendEntries* request containing up to 1 entry.
While implementations may want to send more than 1 at a time, this spec uses
just 1 because it minimizes atomic regions without loss of generality.

$AppendEntries(i, j) \triangleq$
$\land i \neq j$
$\land state[i] = Leader$
$\land Len(log[i]) \geq nextIndex[i][j]$ Limit empty *AppendEntries RPCs*
$\land$ LET $prevLogIndex \triangleq nextIndex[i][j] - 1$
$prevLogTerm \triangleq$ IF $prevLogIndex > 0$ THEN
$log[i][prevLogIndex].term$
ELSE
0

Send up to 1 entry, constrained by the end of the *log*.

$lastEntry \triangleq Min(\{Len(log[i]), nextIndex[i][j]\})$
$entries \triangleq SubSeq(log[i], nextIndex[i][j], lastEntry)$
IN $Send([mtype \mapsto AppendRequest,$
$mterm \mapsto currentTerm[i],$
$mprevLogIndex \mapsto prevLogIndex,$
$mprevLogTerm \mapsto prevLogTerm,$
$mentries \mapsto entries,$

mlog is used as a history variable for the proof.
It would not exist in a real implementation.

$mlog \mapsto log[i],$
$mcommitIndex \mapsto Min(\{commitIndex[i], lastEntry\}),$
$msource \mapsto i,$
$mdest \mapsto j])$
$\land$ UNCHANGED $\langle serverVars, stateVars, followerVars, candidateVars, leaderVars, logVars\rangle$

```
 Follower i transitions to candidate.
BecomeCandidate(i) ≜
    ∧ state[i] = Follower
    ∧ state' = [state EXCEPT ![i] = Candidate]
    ∧ currentTerm' = [currentTerm EXCEPT ![i] = currentTerm[i] + 1]
    ∧ votedFor' = [votedFor EXCEPT ![i] = Nil]
    ∧ votesGranted'   = [votesGranted EXCEPT ![i] = {}]
    ∧ UNCHANGED ⟨messages, stateVars, followerVars, leaderVars, logVars⟩

 Candidate i transitions to leader.
BecomeLeader(i) ≜
    ∧ state[i] = Candidate
    ∧ votesGranted[i] ∈ Quorum
    ∧ state'          = [state EXCEPT ![i] = Leader]
    ∧ nextIndex'  = [nextIndex EXCEPT ![i] = [j ∈ Server ↦ Len(log[i]) + 1]]
    ∧ matchIndex' = [matchIndex EXCEPT ![i] = [j ∈ Server ↦ 0]]
    ∧ UNCHANGED ⟨messages, currentTerm, votedFor, stateVars, followerVars, candidateVars, logVars⟩

 Leader i advances its commitIndex.
 This is done as a separate step from handling AppendEntries responses,
 in part to minimize atomic regions, and in part so that leaders of
 single-server clusters are able to mark entries committed.
AdvanceCommitIndex(i) ≜
    ∧ state[i] = Leader
    ∧ LET  The set of servers that agree up through index.
           Agree(index) ≜ {i} ∪ {k ∈ Server : matchIndex[i][k] ≥ index}
            The maximum indexes for which a quorum agrees
           agreeIndexes ≜ {index ∈ 1 .. Len(log[i]) : Agree(index) ∈ Quorum}
            New value for commitIndex'[i]
           newCommitIndex ≜
              IF  ∧ agreeIndexes ≠ {}
                  ∧ log[i][Max(agreeIndexes)].term = currentTerm[i]
               THEN
                  Max(agreeIndexes)
               ELSE
                  commitIndex[i]
      IN
          ∧ commitIndex' = [commitIndex EXCEPT ![i] = newCommitIndex]
    ∧ UNCHANGED ⟨messages, serverVars, stateVars, followerVars, candidateVars, leaderVars, log⟩

ApplyEntry(i) ≜
    ∧ commitIndex[i] > lastApplied[i]
    ∧ LET entry ≜ log[i][lastApplied[i] + 1]
      IN
          ∧ ∨ ∧ entry.type = OpenSessionEntry
              ∧ session' = [session EXCEPT ![i] = session[i] @@ (entry.index :> [id ↦ entry.index])]
```

```
                    ∧ Send([mtype     ↦ OpenSessionResponse,
                            msession  ↦ entry.index,
                            msource   ↦ i,
                            mdest     ↦ entry.client])
                ∨ ∧ entry.type = CloseSessionEntry
                  ∧ ∨ ∧ entry.session ∈ DOMAIN session
                      ∧ session' = [session EXCEPT ![i] = [j ∈ DOMAIN session[i] \ entry.session ↦ session[i][j
                      ∧ Send([mtype ↦ CloseSessionResponse,
                              msource ↦ i,
                              mdest ↦ session[entry.session].client])
                    ∨ ∧ entry.session ∉ DOMAIN session
                      ∧ UNCHANGED ⟨messages⟩
                ∨ ∧ entry.type = CommandEntry
                  ∧ ∨ ∧ entry.session ∈ DOMAIN session
                      ∧ Send([mtype ↦ CommandResponse,
                              msource ↦ i,
                              mdest ↦ session[entry.session].client])
                    ∨ ∧ entry.session ∉ DOMAIN session
                      ∧ UNCHANGED ⟨messages, session⟩
              ∧ lastApplied' = [lastApplied EXCEPT ![i] = lastApplied[i] + 1]
              ∧ UNCHANGED ⟨serverVars, followerVars, candidateVars, leaderVars, logVars⟩
```

Message handlers
$i =$ recipient, $j =$ sender, $m =$ message

```
HandlePollRequest(i, j, m) ≜
    LET logOk ≜ ∨ m.mlastLogTerm > LastTerm(log[i])
                ∨ ∧ m.mlastLogTerm = LastTerm(log[i])
                  ∧ m.mlastLogIndex ≥ Len(log[i])
        grant ≜ ∧ m.mterm = currentTerm[i]
                ∧ logOk
    IN  ∧ m.mterm ≤ currentTerm[i]
        ∧ Reply([mtype       ↦ PollResponse,
                 mterm       ↦ currentTerm[i],
                 mvoteGranted ↦ grant,
                 msource     ↦ i,
                 mdest       ↦ j],
                 m)
        ∧ UNCHANGED ⟨serverVars, stateVars, followerVars, candidateVars, leaderVars, logVars⟩

HandlePollResponse(i, j, m) ≜
    ∧ m.mterm = currentTerm[i]
    ∧ ∨ ∧ m.mvoteGranted
        ∧ preVotesGranted' = [preVotesGranted EXCEPT ![i] = preVotesGranted[i] ∪ {j}]
      ∨ ∧ ¬m.mvoteGranted
```

```
        ∧ UNCHANGED ⟨preVotesGranted⟩
    ∧ Discard(m)
    ∧ UNCHANGED ⟨serverVars, stateVars, candidateVars, leaderVars, logVars⟩

Server i receives a RequestVote request from server j with
m.mterm ≤ currentTerm[i].
HandleRequestVoteRequest(i, j, m) ≜
    LET logOk ≜ ∨ m.mlastLogTerm > LastTerm(log[i])
                ∨ ∧ m.mlastLogTerm = LastTerm(log[i])
                  ∧ m.mlastLogIndex ≥ Len(log[i])
          grant ≜ ∧ m.mterm = currentTerm[i]
                  ∧ logOk
                  ∧ votedFor[i] ∈ {Nil, j}
    IN  ∧ m.mterm ≤ currentTerm[i]
        ∧ ∨ grant  ∧ votedFor' = [votedFor EXCEPT ![i] = j]
          ∨ ¬grant ∧ UNCHANGED votedFor
        ∧ Reply([mtype          ↦ VoteResponse,
                 mterm          ↦ currentTerm[i],
                 mvoteGranted   ↦ grant,
                 msource        ↦ i,
                 mdest          ↦ j],
                 m)
        ∧ UNCHANGED ⟨state, currentTerm, stateVars, followerVars, candidateVars, leaderVars, logVars⟩

Server i receives a RequestVote response from server j with
m.mterm = currentTerm[i].
HandleRequestVoteResponse(i, j, m) ≜
    This tallies votes even when the current state is not Candidate, but
    they won't be looked at, so it doesn't matter.
    ∧ m.mterm = currentTerm[i]
    ∧ ∨ ∧ m.mvoteGranted
        ∧ votesGranted' = [votesGranted EXCEPT ![i] = votesGranted[i] ∪ {j}]
      ∨ ∧ ¬m.mvoteGranted
        ∧ UNCHANGED ⟨votesGranted⟩
    ∧ Discard(m)
    ∧ UNCHANGED ⟨serverVars, votedFor, stateVars, followerVars, leaderVars, logVars⟩

Server i receives an AppendEntries request from server j with
m.mterm ≤ currentTerm[i]. This just handles m.entries of length 0 or 1, but
implementations could safely accept more by treating them the same as
multiple independent requests of 1 entry.
HandleAppendEntriesRequest(i, j, m) ≜
    LET logOk ≜ ∨ m.mprevLogIndex = 0
                ∨ ∧ m.mprevLogIndex > 0
                  ∧ m.mprevLogIndex ≤ Len(log[i])
                  ∧ m.mprevLogTerm = log[i][m.mprevLogIndex].term
```

```
IN  ∧ m.mterm ≤ currentTerm[i]
    ∧ ∨ ∧  reject request
        ∨ m.mterm < currentTerm[i]
        ∨ ∧ m.mterm = currentTerm[i]
          ∧ state[i] = Follower
          ∧ ¬logOk
      ∧ Reply([mtype       ↦ AppendResponse,
               mterm       ↦ currentTerm[i],
               msuccess    ↦ FALSE,
               mmatchIndex ↦ 0,
               msource     ↦ i,
               mdest       ↦ j],
               m)
      ∧ UNCHANGED ⟨serverVars, logVars⟩
    ∨  return to follower state
      ∧ m.mterm = currentTerm[i]
      ∧ state[i] = Candidate
      ∧ state' = [state EXCEPT ![i] = Follower]
      ∧ UNCHANGED ⟨currentTerm, votedFor, logVars, messages⟩
    ∨  accept request
      ∧ m.mterm = currentTerm[i]
      ∧ state[i] = Follower
      ∧ logOk
      ∧ LET index ≜ m.mprevLogIndex + 1
        IN  ∨  already done with request
            ∧ ∨ Len(m.mentries) = 0
              ∨ ∧ Len(m.mentries) > 0  Raft spec fix
                ∧ Len(log[i]) ≥ index
                ∧ log[i][index].term = m.mentries[1].term
              This could make our commitIndex decrease (for
              example if we process an old, duplicated request),
              but that doesn't really affect anything.
            ∧ commitIndex' = [commitIndex EXCEPT ![i] = m.mcommitIndex]
            ∧ Reply([mtype       ↦ AppendResponse,
                     mterm       ↦ currentTerm[i],
                     msuccess    ↦ TRUE,
                     mmatchIndex ↦ m.mprevLogIndex + Len(m.mentries),
                     msource     ↦ i,
                     mdest       ↦ j],
                     m)
            ∧ UNCHANGED ⟨serverVars, logVars⟩
          ∨  conflict: remove 1 entry
            ∧ m.mentries ≠ ⟨⟩
            ∧ Len(log[i]) ≥ index
            ∧ log[i][index].term     ≠ m.mentries[1].term
```

$\land$ LET $new \triangleq [index2 \in 1 \,..\, (Len(log[i]) - 1) \mapsto log[i][index2]]$
IN $log' = [log$ EXCEPT $![i] = new]$
$\land$ UNCHANGED $\langle serverVars,\ commitIndex,\ messages\rangle$
$\lor$ no conflict: append entry
$\land\ m.mentries \neq \langle\rangle$
$\land\ Len(log[i]) = m.mprevLogIndex$
$\land\ log' = [log$ EXCEPT $![i] = log[i] \circ m.mentries]$ Raft spec fix
$\land$ UNCHANGED $\langle serverVars,\ commitIndex,\ messages\rangle$
$\land$ UNCHANGED $\langle stateVars,\ followerVars,\ candidateVars,\ leaderVars\rangle$

Server i receives an *AppendEntries* response from server j with $m.mterm = currentTerm[i]$.

$HandleAppendEntriesResponse(i,\ j,\ m) \triangleq$
$\land\ m.mterm = currentTerm[i]$
$\land\ \lor\ \land\ m.msuccess$ successful
$\land\ nextIndex' = [nextIndex$ EXCEPT $![i][j] = m.mmatchIndex + 1]$
$\land\ matchIndex' = [matchIndex$ EXCEPT $![i][j] = m.mmatchIndex]$
$\lor\ \land\ \neg m.msuccess$ not successful
$\land\ nextIndex' = [nextIndex$ EXCEPT $![i][j] =$
$Max(\{nextIndex[i][j] - 1,\ 1\})]$
$\land$ UNCHANGED $\langle matchIndex\rangle$
$\land\ Discard(m)$
$\land$ UNCHANGED $\langle serverVars,\ stateVars,\ followerVars,\ candidateVars,\ logVars\rangle$

$HandleOpenSessionRequest(s,\ c,\ m) \triangleq$
$\land\ state[s] = Leader$
$\land\ log' = [log$ EXCEPT $![s] = Append(log[s], [index \mapsto Len(log[s]) + 1,$
$term \mapsto currentTerm[s],$
$type \mapsto OpenSessionEntry,$
$client \mapsto c])]$
$\land\ Discard(m)$
$\land$ UNCHANGED $\langle serverVars,\ stateVars,\ followerVars,\ candidateVars,\ leaderVars,\ commitIndex\rangle$

$HandleCloseSessionRequest(s,\ c,\ m) \triangleq$
$\land\ state[s] = Leader$
$\land\ log' = [log$ EXCEPT $![s] = Append(log[s], [index \mapsto Len(log[s]) + 1,$
$term \mapsto currentTerm[s],$
$type \mapsto CloseSessionEntry,$
$session \mapsto m.msession])]$
$\land\ Discard(m)$
$\land$ UNCHANGED $\langle serverVars,\ stateVars,\ followerVars,\ candidateVars,\ leaderVars,\ commitIndex\rangle$

$HandleCommandRequest(s,\ c,\ m) \triangleq$
$\land\ state[s] = Leader$
$\land\ log' = [log$ EXCEPT $![s] = Append(log[s], [index \mapsto Len(log[s]) + 1,$
$term \mapsto currentTerm[s],$

```
                                                    type      ↦ CommandEntry,
                                                    session   ↦ m.msession,
                                                    command ↦ m.mcommand])]
    ∧ Discard(m)
    ∧ UNCHANGED ⟨serverVars, stateVars, followerVars, candidateVars, leaderVars, commitIndex⟩

 Any RPC with a newer term causes the recipient to advance its term first.
UpdateTerm(i, j, m) ≜
    ∧ "mterm" ∈ DOMAIN m
    ∧ m.mterm > currentTerm[i]
    ∧ currentTerm' = [currentTerm EXCEPT ![i] = m.mterm]
    ∧ state'       = [state       EXCEPT ![i] = Follower]
    ∧ votedFor'    = [votedFor    EXCEPT ![i] = Nil]
       messages is unchanged so m can be processed further.
    ∧ UNCHANGED ⟨messages, stateVars, followerVars, candidateVars, leaderVars, logVars⟩

 Responses with stale terms are ignored.
DropStaleResponse(i, j, m) ≜
    ∧ m.mterm < currentTerm[i]
    ∧ Discard(m)
    ∧ UNCHANGED ⟨serverVars, stateVars, followerVars, candidateVars, leaderVars, logVars⟩

 Receive a message.
Receive(m) ≜
    LET i ≜ m.mdest
        j ≜ m.msource
    IN  Any RPC with a newer term causes the recipient to advance
        its term first. Responses with stale terms are ignored.
        ∨ UpdateTerm(i, j, m)
        ∨ ∧ m.mtype = PollRequest
          ∧ HandlePollRequest(i, j, m)
        ∨ ∧ m.mtype = PollResponse
          ∧ HandlePollResponse(i, j, m)
        ∨ ∧ m.mtype = VoteRequest
          ∧ HandleRequestVoteRequest(i, j, m)
        ∨ ∧ m.mtype = VoteResponse
          ∧ ∨ DropStaleResponse(i, j, m)
            ∨ HandleRequestVoteResponse(i, j, m)
        ∨ ∧ m.mtype = AppendRequest
          ∧ HandleAppendEntriesRequest(i, j, m)
        ∨ ∧ m.mtype = AppendResponse
          ∧ ∨ DropStaleResponse(i, j, m)
            ∨ HandleAppendEntriesResponse(i, j, m)
        ∨ ∧ m.mtype = OpenSessionRequest
          ∧ HandleOpenSessionRequest(i, j, m)
        ∨ ∧ m.mtype = CloseSessionRequest
```

$\land$ $HandleCloseSessionRequest(i,\ j,\ m)$
$\lor$ $\land$ $m.mtype = CommandRequest$
$\land$ $HandleCommandRequest(i,\ j,\ m)$

\ * Modification History
\ * Last modified Sat *Feb* 03 13:30:59 *PST* 2018 by *jordanhalterman*
\ * Created *Tue Jan* 30 15:04:21 *PST* 2018 by *jordanhalterman*