MODULE *curp*

EXTENDS *FiniteSets*, *Naturals*, *Sequences*

CONSTANTS *COMMANDS*, set of records like [key: STRING, value: STRING]
REPLICAS set of replicas

ASSUME $IsFiniteSet(REPLICAS)$

VARIABLES *msgPool*, messages in transit
leader, current leader
epoch, current epoch
specPools, *specPool* of each replica
unsyncedCmds, unsynced commands (backend)
syncedCmds, synced commands (backend)
requestedCmds, client requested commands
committedCmds commands that client believes are stable in the *RSM*

$$
\begin{aligned}
vars \triangleq\ & \langle msgPool, \\
& leader, \\
& epoch, \\
& specPools, \\
& syncedCmds, \\
& unsyncedCmds, \\
& requestedCmds, \\
& committedCmds \rangle
\end{aligned}
$$

The initial state of the system.

$$
\begin{aligned}
&Init \triangleq \\
&\quad \land msgPool = \{\} \\
&\quad \land leader \in REPLICAS \\
&\quad \land epoch = 1 \\
&\quad \land specPools = [r \in REPLICAS \mapsto \{\}] \\
&\quad \land unsyncedCmds = \langle\rangle \\
&\quad \land syncedCmds = \langle\rangle \\
&\quad \land requestedCmds \ = \{\} \\
&\quad \land committedCmds = \{\}
\end{aligned}
$$

Helper function for converting a set to a sequence.

$$
\begin{aligned}
&SetToSeqs(set) \triangleq \\
&\quad \text{LET } len \triangleq 1 \,..\, Cardinality(set) \\
&\qquad seqs \triangleq \{f \in [len \to set] : \forall\, i, j \in len : i \neq j \Rightarrow f[i] \neq f[j]\} \\
&\quad \text{IN } seqs
\end{aligned}
$$

Helper function for checking if a target exists in a sequence.

$$
\begin{aligned}
&ExistInSeq(seq, target) \triangleq \\
&\quad \exists\, i \in 1 \,..\, Len(seq) : seq[i] = target
\end{aligned}
$$

SuperQuorums: In $N = 2 * f + 1$ replicas, a *SuperQuorum* is a set of replicas that contains at least $f + (f + 1) \ / \ 2 + 1$ replicas (including the leader).

The client can consider a command as committed if and only if it receives positive responses from a *SuperQuorum*.

This defines the set consisting of all *SuperQuorums*.

$$
\begin{array}{l}
SuperQuorums \triangleq \\
\quad \{q \in \text{SUBSET } REPLICAS : \\
\qquad \land\ Cardinality(q) \geq (Cardinality(REPLICAS) * 3) \div 4 + 1 \\
\qquad \land\ leader \in q\}
\end{array}
$$

LeastQuorums: In $N = 2 * f + 1$ replicas, a *LeastQuorum* is a set of replicas that contains at least $(f + 1) \ / \ 2 + 1$ replicas.

When a replica becomes a leader, it must recover the command if and only if the command is a *LeastQuorum* of replicas' *specPool*.

This defines the set consisting of all *LeastQuorums*.

$$
\begin{array}{l}
LeastQuorums \triangleq \\
\quad \{q \in \text{SUBSET } REPLICAS : \\
\qquad Cardinality(q) \geq Cardinality(REPLICAS) \div 4 + 1\}
\end{array}
$$

RecoveryQuorums: In $N = 2 * f + 1$ replicas, a *RecoveryQuorum* is a set of replicas that contains at least $f + 1$ replicas.

A replica must gather a *RecoveryQuorum* of replicas' *specPool* to recover the commands that need to be recovered.

This defines the set consisting of all *RecoveryQuorums*.

$$
\begin{array}{l}
RecoveryQuorums \triangleq \\
\quad \{q \in \text{SUBSET } REPLICAS : \\
\qquad Cardinality(q) \geq Cardinality(REPLICAS) \div 2 + 1\}
\end{array}
$$

Client sends a request to all replicas.

$$
\begin{array}{l}
ClientSendRequest(cmd) \triangleq \\
\quad \land\ requestedCmds' = requestedCmds \cup \{cmd\} \\
\quad \land\ msgPool' = msgPool \cup \\
\qquad [type : \{\text{“request”}\}, \\
\qquad\ cmd : \{cmd\}, \\
\qquad\ dst\ \ : REPLICAS] \\
\quad \land \text{UNCHANGED } \langle leader, \\
\qquad\qquad\qquad\qquad epoch, \\
\qquad\qquad\qquad\qquad specPools, \\
\qquad\qquad\qquad\qquad syncedCmds, \\
\qquad\qquad\qquad\qquad unsyncedCmds, \\
\qquad\qquad\qquad\qquad committedCmds\rangle
\end{array}
$$

Replica receives a request from the client.

If there is no conflict command (command on the same key) in the *specPool*, the replica adds the command to its *specPool*.

The leader will always add the command to *unsyncedCmds*.

$$
\begin{array}{l}
ReplicaReceiveRequest(r,\ msg)\ \triangleq \\
\quad \text{IF } \neg ExistInSeq(syncedCmds,\ msg.cmd) \text{ THEN} \\
\qquad \text{LET } conflict \triangleq (\exists\, cmd \in specPools[r] : cmd.key = msg.cmd.key) \text{IN} \\
\qquad\quad \land\ specPools' = [specPools \text{ EXCEPT } ![r] = \\
\qquad\qquad \text{IF } conflict \text{ THEN } @ \text{ ELSE } @ \cup \{msg.cmd\}] \\
\qquad\quad \land\ unsyncedCmds' = \\
\qquad\qquad \text{IF } r = leader \text{ THEN } Append(unsyncedCmds,\ msg.cmd) \\
\qquad\qquad \text{ELSE } unsyncedCmds \\
\qquad\quad \land\ msgPool' = (msgPool \setminus \{msg\}) \cup \\
\qquad\qquad \{[type \mapsto \text{“response”}, \\
\qquad\qquad\ cmd \mapsto msg.cmd, \\
\qquad\qquad\ ok \mapsto \neg conflict, \\
\qquad\qquad\ src \mapsto r]\} \\
\qquad\quad \land \text{ UNCHANGED } \langle leader, \\
\qquad\qquad\qquad\qquad epoch, \\
\qquad\qquad\qquad\qquad syncedCmds, \\
\qquad\qquad\qquad\qquad requestedCmds, \\
\qquad\qquad\qquad\qquad committedCmds\rangle \\
\qquad \text{ELSE} \quad \text{If the command is already synced, the replica does nothing.} \\
\qquad\quad \land\ msgPool' = msgPool \setminus \{msg\} \\
\qquad\quad \land \text{ UNCHANGED } \langle leader, \\
\qquad\qquad\qquad\qquad epoch, \\
\qquad\qquad\qquad\qquad specPools, \\
\qquad\qquad\qquad\qquad syncedCmds, \\
\qquad\qquad\qquad\qquad unsyncedCmds, \\
\qquad\qquad\qquad\qquad requestedCmds, \\
\qquad\qquad\qquad\qquad committedCmds\rangle
\end{array}
$$

Client receives a response from a replica.

If the client got positive responses from a *SuperQuorum*, the client considers the command as committed.

If the client got negative responses from a *LeastQuorum*, the command can never accepted by a *SuperQuorum*. Thus the client stops waiting for it.

$$
\begin{array}{l}
ClientReceiveResponse(msg)\ \triangleq \\
\quad \text{LET } sameCmdResp \triangleq \\
\qquad \{resp \in msgPool : resp.type = \text{“response”} \land resp.cmd = msg.cmd\} \\
\quad \text{IN} \\
\qquad \lor\ \land \{m.src : m \in \{resp \in sameCmdResp : resp.ok\}\} \in SuperQuorums \\
\qquad\quad \land\ committedCmds' = committedCmds \cup \{msg.cmd\} \\
\qquad\quad \land\ msgPool' = msgPool \setminus sameCmdResp \\
\qquad\quad \land \text{ UNCHANGED } \langle leader,
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\quad epoch, \\
\qquad\qquad\qquad\quad specPools, \\
\qquad\qquad\qquad\quad syncedCmds, \\
\qquad\qquad\qquad\quad unsyncedCmds, \\
\qquad\qquad\qquad\quad requestedCmds\rangle \\
\lor\ \land \{m.src : m \in \{resp \in sameCmdResp : \neg resp.ok\}\} \in LeastQuorums \\
\quad\ \land\ msgPool' = msgPool \setminus sameCmdResp \\
\quad\ \land\ \text{UNCHANGED}\ \langle leader, \\
\qquad\qquad\qquad\quad epoch, \\
\qquad\qquad\qquad\quad specPools, \\
\qquad\qquad\qquad\quad syncedCmds, \\
\qquad\qquad\qquad\quad unsyncedCmds, \\
\qquad\qquad\qquad\quad requestedCmds, \\
\qquad\qquad\qquad\quad committedCmds\rangle
\end{array}
$$

Client Actions

$$
\begin{array}{l}
ClientAction\ \triangleq \\
\quad \lor \exists\, cmd \in (COMMANDS \setminus requestedCmds) : ClientSendRequest(cmd) \\
\quad \lor \exists\, msg \in msgPool : \land\ msg.type = \text{“response”} \\
\qquad\qquad\qquad\qquad\qquad\ \land\ ClientReceiveResponse(msg)
\end{array}
$$

Replica Actions

$$
\begin{array}{l}
ReplicaAction\ \triangleq \\
\quad \exists\, msg \in msgPool : \\
\qquad \land\ msg.type = \text{“request”} \\
\qquad \land\ ReplicaReceiveRequest(msg.dst,\ msg)
\end{array}
$$

Syncing an *unsyncedCmd*

$$
\begin{array}{l}
SyncAction\ \triangleq \\
\quad \land\ unsyncedCmds \neq \langle\rangle \\
\quad \land\ specPools' = [r \in REPLICAS \mapsto specPools[r] \setminus \{Head(unsyncedCmds)\}] \\
\quad \land\ syncedCmds' = Append(syncedCmds,\ Head(unsyncedCmds)) \\
\quad \land\ unsyncedCmds' = Tail(unsyncedCmds) \\
\quad \land\ \text{UNCHANGED}\ \langle msgPool, \\
\qquad\qquad\qquad\quad leader, \\
\qquad\qquad\qquad\quad epoch, \\
\qquad\qquad\qquad\quad requestedCmds, \\
\qquad\qquad\qquad\quad committedCmds\rangle
\end{array}
$$

Leader Change Action

The new leader should gather at least a RecoveryQuorum of replicas specPool to recover the commands.

Commands occurring in the *specPool* of a *LeastQuorum* of replicas need to be recovered.

$$
\begin{array}{l}
\mathit{LeaderChangeAction} \triangleq \\
\quad \exists\, \mathit{newLeader} \in \mathit{REPLICAS} : \\
\qquad \land\ \mathit{newLeader} \neq \mathit{leader} \\
\qquad \land\ \exists\, \mathit{recoveryQuorum} \in \mathit{RecoveryQuorums} : \\
\qquad\quad \text{LET } \mathit{specPoolCmds} \triangleq \text{UNION } \{(\{ \\
\qquad\qquad\qquad [\mathit{cmd} \mapsto \mathit{cmd},\ r \mapsto r] : \mathit{cmd} \in \mathit{specPools}[r] \\
\qquad\qquad \}) : r \in \mathit{recoveryQuorum}\} \\
\qquad\qquad \mathit{filteredSpecPoolCmds} \triangleq \{c1 \in \mathit{specPoolCmds} : \{ \\
\qquad\qquad\qquad c.r : c \in \{c2 \in \mathit{specPoolCmds} : c1.\mathit{cmd} = c2.\mathit{cmd}\} \\
\qquad\qquad \} \in \mathit{LeastQuorums}\} \\
\qquad\qquad \mathit{newSpecPool} \triangleq \{c.\mathit{cmd} : c \in \mathit{filteredSpecPoolCmds}\} \\
\qquad\quad \text{IN} \\
\qquad\qquad \land\ \mathit{specPools}' = [\mathit{specPools} \text{ EXCEPT } ![\mathit{newLeader}] = \mathit{newSpecPool}] \\
\qquad\qquad \land\ \text{LET } \mathit{newUnsyncedCmds} \triangleq \\
\qquad\qquad\qquad\quad \text{CHOOSE } s \in \mathit{SetToSeqs}(\mathit{newSpecPool}) : \text{TRUE} \\
\qquad\qquad\quad \text{IN} \quad \mathit{unsyncedCmds}' = \mathit{unsyncedCmds} \circ \mathit{newUnsyncedCmds} \\
\qquad\qquad \land\ \mathit{leader}' = \mathit{newLeader} \\
\qquad\qquad \land\ \mathit{epoch}' = \mathit{epoch} + 1 \\
\qquad\qquad \land\ \text{UNCHANGED } \langle \mathit{msgPool}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{syncedCmds}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{requestedCmds}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{committedCmds}\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{Next} \triangleq \\
\quad \lor\ \mathit{LeaderChangeAction} \\
\quad \lor\ \mathit{ClientAction} \\
\quad \lor\ \mathit{ReplicaAction} \\
\quad \lor\ \mathit{SyncAction}
\end{array}
$$

$$
\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}} \land \mathrm{WF}_{\mathit{vars}}(\mathit{Next})
$$

Type Check

$$
\begin{array}{l}
\mathit{TypeOK} \triangleq \\
\quad \land\quad \mathit{msgPool} \subseteq \\
\qquad\quad [\mathit{type} : \{\text{“request”}\}, \\
\qquad\quad\ \mathit{cmd} : \mathit{COMMANDS}, \\
\qquad\quad\ \mathit{dst}\ \ : \mathit{REPLICAS}] \cup \\
\qquad\quad [\mathit{type} : \text{“response”}, \\
\qquad\quad\ \mathit{cmd} : \mathit{COMMANDS}, \\
\qquad\quad\ \mathit{ok} : \{\text{TRUE, FALSE}\}, \\
\qquad\quad\ \mathit{src} : \mathit{REPLICAS}] \\
\quad \land\quad \mathit{leader} \in \mathit{REPLICAS} \\
\quad \land\quad \mathit{epoch} \in \mathit{Nat} \\
\quad \land\quad \forall\, r \in \mathit{REPLICAS} : \\
\qquad\quad \text{LET } \mathit{specPool} \triangleq \mathit{specPools}[r] \text{IN}
\end{array}
$$

$$
\begin{array}{ll}
 & \land\ specPool \subseteq COMMANDS \\
 & \land\ \forall\, cmd1,\ cmd2 \in specPool : \\
 & \quad cmd1.key \neq cmd2.key \\
\land & syncedCmds \in \{SetToSeqs(s) : s \in \text{SUBSET } COMMANDS\} \\
\land & unsyncedCmds \in \{SetToSeqs(s) : s \in \text{SUBSET } COMMANDS\} \\
\land & requestedCmds \subseteq COMMANDS \\
\land & committedCmds \subseteq COMMANDS
\end{array}
$$

Stability Property

If the client considers a command has been committed, the command must eventually be synced.

$$
\begin{array}{l}
Stability \triangleq \\
\quad \forall\, cmd \in COMMANDS : \\
\qquad cmd \in committedCmds \leadsto cmd \in \{syncedCmds[i] : i \in \text{DOMAIN } syncedCmds\}
\end{array}
$$

THEOREM $Spec \Rightarrow (\Box\, TypeOK) \land Stability$