MODULE *curp*

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

Constants: commands: set of records like [*key* $\mapsto$ "key", value $\mapsto$ "value"].
each command should be unique in the set.
replicas: set of replicas.

CONSTANTS *commands*, *replicas*

Variables: leader: the current leader. epoch: the current epoch (the number of leader changes).
specPools: the spec pool of each replica.
requested: the set of requested commands (by client).
committed: the set of committed commands by *CURP*. It also records the index of the last same-key command in the *synced* sequence at the time.
unsynced: the sequence of *unsynced* commands. *synced*: the sequence of *synced* commands.

VARIABLES *leader*, *epoch*, *specPools*, *requested*, *committed*, *unsynced*, *synced*

The initial state of the system.

$$
\begin{aligned}
&Init \triangleq \\
&\quad \land leader \in replicas \\
&\quad \land epoch = 1 \\
&\quad \land specPools = [r \in replicas \mapsto \{\}] \\
&\quad \land requested = \{\} \\
&\quad \land committed = \{\} \\
&\quad \land unsynced = \langle\rangle \\
&\quad \land synced = \langle\rangle
\end{aligned}
$$

Helper function for converting a set to a set containing all sequences containing the elements of the set exactly once and no other elements.

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

Helper function for getting the index of the last element in a sequence satisfying the predicate.

$$
\begin{aligned}
&GetIdxInSeq(seq,\ Test(\_)) \triangleq \\
&\quad \text{LET } I \triangleq \{i \in 1 \,..\, Len(seq) : Test(seq[i])\} \text{ IN} \\
&\qquad \text{IF } I \neq \{\} \text{ THEN CHOOSE } i \in I : \forall\, j \in I : j \leq i \text{ ELSE } 0
\end{aligned}
$$

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

The client can consider a command as committed if and only if it receives positive responses from a set of replicas larger then a *SuperQuorum*.

$$
\begin{aligned}
&IsSuperQuorum(S) \triangleq \\
&\quad \text{LET } f \triangleq (Cardinality(replicas) - 1) \div 2 \\
&\qquad size \triangleq f + (f + 1) \div 2 + 1 \\
&\quad \text{IN} \quad Cardinality(S) \geq size
\end{aligned}
$$

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

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

$$
\begin{array}{l}
IsLeastQuorum(S) \triangleq \\
\quad \text{LET } f \triangleq (Cardinality(replicas) - 1) \div 2 \\
\qquad size \triangleq (f + 1) \div 2 + 1 \\
\quad \text{IN } \quad Cardinality(S) \geq size
\end{array}
$$

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*.

$$
\begin{array}{l}
recoveryQuorums \triangleq \\
\quad \text{LET } f \triangleq (Cardinality(replicas) - 1) \div 2 \\
\qquad size \triangleq f + 1 \\
\quad \text{IN } \quad \{q \in \text{SUBSET } replicas : Cardinality(q) = size\}
\end{array}
$$

The Abstraction of the normal procedure of *CURP*.

$$
\begin{array}{l}
Request \triangleq \\
\quad \exists\, cmd \in commands \setminus requested : \\
\qquad \land \ requested' = requested \cup \{cmd\}
\end{array}
$$

To simulate the unreliability of the network,
only a subset of replicas could receive the request.

$$
\begin{array}{l}
\qquad \land \exists\, received \in \text{SUBSET } replicas : \\
\end{array}
$$

The set of replicas that got no conflict in the spec pool.

$$
\begin{array}{l}
\qquad\quad \land \text{LET } acceptedReplicas \triangleq \\
\qquad\qquad \{r \in received : \\
\qquad\qquad\quad \forall\, specCmd \in specPools[r] : \\
\qquad\qquad\qquad specCmd.key \neq cmd.key\} \\
\qquad\quad \text{IN}
\end{array}
$$

Update the *specPool*.

$$
\begin{array}{l}
\qquad\qquad \land\ specPools' = [r \in replicas \mapsto \\
\qquad\qquad\quad \text{IF } r \in acceptedReplicas \\
\qquad\qquad\quad\ \text{THEN } specPools[r] \cup \{cmd\} \\
\qquad\qquad\quad\ \text{ELSE } specPools[r]]
\end{array}
$$

If there is at least a superquorum set of replicas that accepted
the request, and the leader can execute the command,
the request is committed.

$$
\begin{array}{l}
\qquad\qquad \land \text{LET } CompareKey(elem) \triangleq elem.key = cmd.key \text{IN} \\
\qquad\qquad\quad \text{IF} \\
\qquad\qquad\qquad \land\ IsSuperQuorum(acceptedReplicas)
\end{array}
$$

$\land\ leader \in acceptedReplicas$
$\land\ GetIdxInSeq(unsynced,\ CompareKey) = 0$
THEN
The previous state of the key is also recorded.
This is used to check the correctness of the property.
LET $prevIdx \triangleq GetIdxInSeq(synced,\ CompareKey)$ IN
$committed' = committed \cup \{[$
$cmd \mapsto cmd,$
$prevIdx \mapsto prevIdx]\}$
ELSE $committed' = committed$

No matter if the request is committed or not,
as long as the leader is in the received set,
the command should be *synced* afterward.
$\land$ IF $leader \in received$
THEN $unsynced' = Append(unsynced,\ cmd)$
ELSE $unsynced' = unsynced$

$\land$ UNCHANGED $\langle leader,\ epoch,\ synced\rangle$

Syncing a command using the back-end protocol like Raft. The implementation details of the back-end protocol are omitted.

$Sync \triangleq$
$\land\ unsynced \neq \langle\rangle$
$\land\ specPools' = [r \in replicas \mapsto specPools[r] \setminus \{Head(unsynced)\}]$
$\land\ synced' = Append(synced,\ Head(unsynced))$
$\land\ unsynced' = Tail(unsynced)$
$\land$ UNCHANGED $\langle leader,\ epoch,\ requested,\ committed\rangle$

Leader Change Action

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

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

$LeaderChange \triangleq$
$\exists\, newLeader \in (replicas \setminus \{leader\}) :$
$\land\ leader' = newLeader$
$\land\ epoch' = epoch + 1$
$\land\ \exists\, recoveryQuorum \in recoveryQuorums\ :$
LET $specCmds \triangleq$ UNION $\{specPools[r]\ : r \in recoveryQuorum\}$
$newSpecPool \triangleq \{cmd \in specCmds : IsLeastQuorum(\{r \in replicas : cmd \in specPools[r]\})\}$
IN
$\land\ specPools' = [specPools$ EXCEPT $![newLeader] = newSpecPool]$
$\land\ unsynced' \in SetToSeqs(newSpecPool)$
$\land$ UNCHANGED $\langle requested,\ committed,\ synced\rangle$

$Next \triangleq$
$\lor\ Request$
$\lor\ Sync$
$\lor\ LeaderChange$

$Spec \triangleq Init \land \Box[Next]_{\langle leader,\ epoch,\ specPools,\ requested,\ committed,\ unsynced,\ synced\rangle}$

Type Check

$TypeOK \triangleq$
$\land\ leader \in replicas$
$\land\ epoch \in Nat$
$\land\ \forall\, r \in replicas : specPools[r] \subseteq commands$
$\land\ requested \subseteq commands$
$\land\ \forall\, committedCmd \in committed :$
$\quad \land\ committedCmd.cmd \in commands$
$\quad \land\ committedCmd.prevIdx \in 0\,..\,Len(synced)$
$\land\ synced \in \{SetToSeqs(s) : s \in \text{SUBSET } commands\}$
$\land\ unsynced \in \{SetToSeqs(s) : s \in \text{SUBSET } commands\}$

Stability Property

This is the key property of *CURP*. There are two parts of the property.

1. If a command is committed by *CURP*, command will eventually be *synced* by the back-end protocol.

2. If a command is committed by *CURP*, when the command is *synced* be the back-end protocol, there will never be a command with the same key between the command and the recorded previous same-key command in the *synced* sequence.

$Stability \triangleq$
$\forall\, committedCmd \in committed :$
$\quad \text{LET } CompareExact(elem) \triangleq elem = committedCmd.cmd$
$\qquad syncedIdx \triangleq GetIdxInSeq(synced,\ CompareExact)$
$\quad \text{IN}$
$\qquad \land\ syncedIdx \neq 0$
$\qquad \land \forall\, j \in (committedCmd.prevIdx + 1)\,..\,(syncedIdx - 1) :$
$\qquad\quad synced[j].key \neq committedCmd.cmd.key$

THEOREM $Spec \Rightarrow \Box\, TypeOK \land \Diamond Stability$