MODULE *JustInTimePaxos*

Defines the Just-In-Time *Paxos* (*JITPaxos*) protocol. *JITPaxos* is a variant of the *Paxos* consensus protocol designed for environments where process clocks are synchronized with high precision. The protocol relies on synchronized clocks to establish a global total ordering of events, avoiding coordination between replicas when requests arrive in the expected order, and reconciling requests only when they arrive out of order. This allows *JITPaxos* to reach consensus within a single round trip in the normal case, falling back to traditional replication strategies only when required.

JITPaxos uses a view-based approach to elect a primary and reconcile logs across views. Views are identified by a monotonically increasing, globally unique view *ID*. Each view deterministically assigns a quorum, and within the quorum a primary replica responsible for executing client requests and reconciling inconsistencies in the logs of the remaining replicas. *JITPaxos* replicas to not coordinate with each other in the normal case. Clients send timestamped requests in parallel to every replica in the view's quorum. When a replica receives a client request, if the request is received in chronological order, it's appended to the replica's *log*. If a request is received out of order (*i.e.* the request timestamp is less than the last timestamp in the replica's *log*), the request is rejected by the replica. Clients are responsible for identifying inconsistencies in the quorum's logs and initiating the reconciliation protocol. To help clients identify inconsistencies, replicas return a checksum representing the contents of the *log* up to the request point with each client reply. If a client's request is received out of chronological order, or if the checksums provided by the quorum do not match, the client must initiate the reconcilitation protocol to reconcile the inconsistencies in the quorum's logs.

When requests are received out-of-order, the reconciliation protocol works to re-order requests using the view's primary as a reference. When a client initiates the reconciliation protocol for an inconsistent replica, the replica stops accepting client requests and sends a repair request to the primary. The primary responds with the subset of the *log* not yet reconciled on the replica, and the replica replaces the out-of-order entries in its *log*. Once the replica's *log* has been reconciled with the primary, it can acknowledge the reconciled request and begin accepting requests again. Once a client has reconciled all the divergent replicas and has received acknowledgement from each of the replicas in the quorum, the request can be committed.

View primaries and quorums are evenly distributed amongst view *IDs*. View changes can be initiated to change the primary or the set of replicas in the quorum. When a view change is initiated, each replica sends its *log* to the primary for the initiated view. Once the primary has received logs from a majority of replicas, it initializes the view with the *log* from the most recent in-sync replica, broadcasting the *log* to its peers. The use of quorums to determine both the commitment of a request and the initialization of new views ensures that each view *log* contains all prior committed requests.

EXTENDS *Naturals*, *Reals*, *Sequences*, *FiniteSets*, *TLC*

The set of *JITPaxos* replicas
CONSTANT *Replicas*

The set of *JITPaxos* clients
CONSTANT *Clients*

The set of possible values
CONSTANT *Values*

An empty value
CONSTANT *Nil*

Request/response types

CONSTANTS
MClientRequest,
MClientReply,
MReconcileRequest,
MReconcileReply,
MRepairRequest,
MRepairReply,
MViewChangeRequest,
MViewChangeReply,
MStartViewRequest

Replica statuses
CONSTANTS
SInSync,
SRepair,
SViewChange

This section specifies the message types and schemas used in this spec.

$ReqIDs \triangleq [c \in Clients \mapsto i \in (1 \,..\,)]$

$ViewIDs \triangleq [r \in Replicas \mapsto i \in (1 \,..\,)]$

$\text{Logs} \triangleq [r \in Replicas \mapsto [i \in (1 \,..\,) \mapsto \text{Value}]]$

$\text{Indexes} \triangleq [r \in Replicas \mapsto i \in (1 \,..\,)]$

$\text{Timestamps} \triangleq [r \in Replicas \mapsto i \in (1 \,..\,)]$

$\text{Checksums} \triangleq [r \in Replicas \mapsto [i \in (1 \,..\,) \mapsto t \in \text{Timestamps}]]$

ClientRequest
[*src* $\mapsto c \in Clients$,
dest $\mapsto r \in Replicas$,
type $\mapsto MClientRequest$,
viewID $\mapsto i \in ViewIDs$,
reqID $\mapsto i \in ReqIDs$,
value $\mapsto v \in Values$,
timestamp $\mapsto t \in$ Timestamps]

ClientReply
[*src* $\mapsto r \in Replicas$,
dest $\mapsto c \in Clients$,
req $\mapsto$ (*ClientRequest*),
type $\mapsto MClientReply$,
viewID $\mapsto i \in ViewIDs$,
index $\mapsto i \in$ Indexes,
checksum $\mapsto c \in$ Checksums,
value $\mapsto v \in Values$,
timestamp $\mapsto t \in$ Timestamps,
succeeded $\mapsto$ TRUE $\vee$ FALSE]

ReconcileRequest
[$src \mapsto c \in Clients$,
$dest \mapsto r \in Replicas$,
$type \mapsto MReconcileRequest$,
$viewID \mapsto i \in ViewIDs$,
$reqID \mapsto i \in ReqIDs$,
$index \mapsto i \in$ Indexes]

ReconcileReply
[$src \mapsto r \in Replicas$,
$dest \mapsto c \in Clients$,
$req \mapsto (ClientRequest)$,
$type \mapsto MReconcileReply$,
$viewID \mapsto i \in ViewIDs$,
$index \mapsto i \in$ Indexes,
$checksum \mapsto c \in$ Checksums,
$value \mapsto v \in Values$,
$timestamp \mapsto t \in$ Timestamps,
$succeeded \mapsto$ TRUE $\vee$ FALSE]

RepairRequest
[$src \mapsto r \in Replicas$,
$dest \mapsto r \in Replicas$,
$req \mapsto (ClientRequest)$,
$type \mapsto MRepairRequest$,
$viewID \mapsto i \in ViewIDs$,
$index \mapsto i \in$ Indexes]

RepairReply
[$src \mapsto r \in Replicas$,
$dest \mapsto r \in Replicas$,
$req \mapsto (ClientRequest)$,
$type \mapsto MRepairReply$,
$viewID \mapsto i \in ViewIDs$,
$index \mapsto i \in$ Indexes,
$log \mapsto l \in$ Logs]

ViewChangeRequest
[$src \mapsto r \in Replicas$,
$dest \mapsto r \in Replicas$,
$type \mapsto MViewChangeRequest$,
$viewID \mapsto i \in ViewIDs$]

ViewChangeReply
[$src \mapsto r \in Replicas$,
$dest \mapsto r \in Replicas$,
$type \mapsto MViewChangeReply$,
$viewID \mapsto i \in ViewIDs$,
$logViewID \mapsto i \in ViewIDs$,
$log \mapsto l \in$ Logs]

StartViewRequest
[$src \mapsto r \in Replicas$,

$dest \mapsto r \in Replicas$,
$type \mapsto MStartViewRequest$,
$viewID \mapsto i \in ViewIDs$,
$log \mapsto l \in$ Logs]

The set of all messages on the network
VARIABLE $messages$

The total number of messages sent
VARIABLE $messageCount$

The total number of steps executed
VARIABLE $stepCount$

$messageVars \triangleq \langle messages, messageCount, stepCount \rangle$

Local client state

Strictly increasing representation of synchronized time
VARIABLE $cTime$

The highest known view *ID* for a client
VARIABLE $cViewID$

Client request *IDs*
VARIABLE $cReqID$

A client response buffer
VARIABLE $cReps$

A set of all commits - used for model checking
VARIABLE $cCommits$

$clientVars \triangleq \langle cTime, cViewID, cReqID, cReps, cCommits \rangle$

Local replica state

The current status of a replica
VARIABLE $rStatus$

The current view *ID* for a replica
VARIABLE $rViewID$

A replica's commit *log*
VARIABLE $rLog$

A replica's sync index
VARIABLE $rSyncIndex$

The view *ID* for the *log*

VARIABLE $rLogViewID$

The set of view change replies

VARIABLE $rViewChangeReps$

$replicaVars \triangleq \langle rStatus,\ rViewID,\ rLog,\ rSyncIndex,\ rLogViewID,\ rViewChangeReps \rangle$

$vars \triangleq \langle messageVars,\ clientVars,\ replicaVars \rangle$

This section provides utilities for implementing the spec.

Creates a sequence from set 'S'

RECURSIVE $SeqFromSet(\_)$

$SeqFromSet(S) \triangleq$
  IF $S = \{\}$ THEN
    $\langle\rangle$
  ELSE LET $x \triangleq$ CHOOSE $x \in S$: TRUE
    IN $\langle x \rangle \circ SeqFromSet(S \setminus \{x\})$

RECURSIVE $SetReduce(\_, \_, \_)$

$SetReduce(Op(\_, \_),\ S,\ value) \triangleq$
  IF $S = \{\}$ THEN
    $value$
  ELSE
    LET $s \triangleq$ CHOOSE $s \in S$: TRUE
    IN $SetReduce(Op,\ S \setminus \{s\},\ Op(s,\ value))$

Computes the greatest vlue in set 'S'

$Max(S) \triangleq$ CHOOSE $x \in S : \forall\, y \in S : x \geq y$

Computes the sum of numbers in set 'S'

$Sum(S) \triangleq$ LET $\_op(a,\ b) \triangleq a + b$
  IN $SetReduce(\_op,\ S,\ 0)$

The values of a sequence

$Range(s) \triangleq \{s[i] : i \in$ DOMAIN $s\}$

This section provides helpers for the protocol.

A sorted sequence of replicas

$replicas \triangleq SeqFromSet(Replicas)$

The primary index for view 'v'

$PrimaryIndex(v) \triangleq (v\%Len(replicas)) + ($IF $v \geq Len(replicas)$ THEN 1 ELSE $0)$

The primary for view 'v'

$Primary(v) \triangleq replicas[PrimaryIndex(v)]$

Quorum is the quorum for a given view

$Quorum(v) \triangleq$

LET

$quorumSize \triangleq Len(replicas) \div 2 + 1$

$index(i) \triangleq PrimaryIndex(v) + (i - 1)$

$member(i) \triangleq$ IF $index(i) > Len(replicas)$ THEN $replicas[index(i)\%Len(replicas)]$ ELSE $replicas[inde$

IN

$\{member(i) : i \in 1 \,..\, quorumSize\}$

A boolean indicating whether the given set is a quorum

$IsQuorum(S) \triangleq Cardinality(S) * 2 \geq Cardinality(Replicas)$

A boolean indicating whether the given set is a quorum that includes the given replica

$IsLocalQuorum(r,\, S) \triangleq IsQuorum(S) \wedge r \in S$

This section models the network.

Send a set of messages

$Sends(ms) \triangleq$

$\wedge\ messages' = messages \cup ms$

$\wedge\ messageCount' = messageCount + Cardinality(ms)$

$\wedge\ stepCount' = stepCount + 1$

Send a message

$Send(m) \triangleq Sends(\{m\})$

Ack a message

$Ack(m) \triangleq$

$\wedge\ messages' = messages \setminus \{m\}$

$\wedge\ messageCount' = messageCount + 1$

$\wedge\ stepCount' = stepCount + 1$

Ack a message and send a set of messages

$AckAndSends(m,\, ms) \triangleq$

$\wedge\ messages' = (messages \cup ms) \setminus \{m\}$

$\wedge\ messageCount' = messageCount + Cardinality(ms)$

$\wedge\ stepCount' = stepCount + 1$

Ack and send a message

$AckAndSend(m,\, n) \triangleq AckAndSends(m,\, \{n\})$

Reply to a message with a set of responses

$Replies(req,\, reps) \triangleq AckAndSends(req,\, reps)$

Reply to a message

$Reply(req,\ resp) \triangleq AckAndSend(req,\ resp)$

This section models client requests.

Client 'c' sends value 'v' to all replicas

$$
\begin{array}{l}
ClientRequest(c,\ v) \triangleq \\
\quad \land\ cTime' = cTime + 1 \\
\quad \land\ cReqID' = [cReqID \text{ EXCEPT } ![c] = cReqID[c] + 1] \\
\quad \land\ Sends(\{[src \mapsto c, \\
\qquad\qquad dest \mapsto r, \\
\qquad\qquad type \mapsto MClientRequest, \\
\qquad\qquad viewID \mapsto cViewID[c], \\
\qquad\qquad reqID \mapsto cReqID'[c], \\
\qquad\qquad value \mapsto v, \\
\qquad\qquad timestamp \mapsto cTime'] : r \in Quorum(cViewID[c])\}) \\
\quad \land \text{ UNCHANGED } \langle replicaVars,\ cViewID,\ cReps,\ cCommits\rangle
\end{array}
$$

Client 'c' handles a response 'm' from replica 'r'

$$
\begin{array}{l}
HandleClientReply(c,\ r,\ m) \triangleq \\
\qquad \text{If the reply view } ID \text{ does not match the request view } ID\text{, update the client's view.} \\
\quad \land\ \lor\ \land\ m.viewID \neq m.req.viewID \\
\qquad\quad \land\ \lor\ \land\ cViewID[c] < m.viewID \\
\qquad\qquad\qquad \land\ cViewID' = [cViewID \text{ EXCEPT } ![c] = m.viewID] \\
\qquad\qquad \lor\ \land\ cViewID[c] \geq m.viewID \\
\qquad\qquad\qquad \land \text{ UNCHANGED } \langle cViewID\rangle \\
\qquad\quad \land\ Ack(m) \\
\qquad\quad \land \text{ UNCHANGED } \langle cReps,\ cCommits\rangle \\
\qquad \text{If the request and reply views match and the reply view matches the client's view,} \\
\qquad \text{aggregate the replies for the associated client request.} \\
\qquad \lor\ \land\ m.viewID = m.req.viewID \\
\qquad\quad \land\ m.viewID = cViewID[c] \\
\qquad\quad \land\ \lor\ \land\ m.succeeded \\
\qquad\qquad\qquad \land\ cReps' = [cReps \text{ EXCEPT } ![c] = \\
\qquad\qquad\qquad\qquad\qquad (cReps[c] \setminus \{n \in cReps[c] : \land\ n.src = m.src \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land\ n.viewID = cViewID[c] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land\ n.req.reqID = m.req.reqID \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \land\ \neg n.succeeded\}) \cup \{m\}] \\
\qquad\qquad\quad \lor\ \land\ \neg m.succeeded \\
\qquad\qquad\qquad\quad \land\ \neg\exists\, n \in cReps[c] : \land\ n.src = m.src \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ n.viewID = cViewID[c] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ n.req.reqID = m.req.reqID \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ n.succeeded \\
\qquad\qquad\qquad\quad \land\ cReps' = [cReps \text{ EXCEPT } ![c] = cReps[c] \cup \{m\}] \\
\qquad\quad \land \text{ LET } reps \triangleq \{n \in cReps'[c] : \land\ n.viewID = cViewID[c] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\ n.req.reqID = m.req.reqID\}
\end{array}
$$

```
        isQuorum      ≜ {n.src : n ∈ {n ∈ reps : n.succeeded}} = Quorum(cViewID[c])
        isCommitted   ≜ ∧ ∀ n ∈ reps : n.succeeded
                        ∧ Cardinality({n.checksum : n ∈ reps}) = 1
        hasPrimary    ≜ ∃ n ∈ reps : n.src = Primary(cViewID[c]) ∧ n.succeeded
    IN
        If a quorum of successful replies have been received and the checksums
        match, add the primary reply to commits.
      ∨ ∧ isQuorum
        ∧ isCommitted
        ∧ LET commit ≜ CHOOSE n ∈ reps : n.src = Primary(cViewID[c])
          IN   cCommits' = [cCommits EXCEPT ![c] = cCommits[c] ∪ {commit}]
        ∧ Ack(m)
        If some reply failed or was returned with an incorrect checksum,
        send a ReconcileRequest to the inconsistent node to force it to
        reconcile its log with the primary's log.
      ∨ ∧ ¬isCommitted
        ∧ ∨ ∧ hasPrimary
            ∧ LET primaryRep ≜ CHOOSE n ∈ reps : ∧ n.src = Primary(cViewID[c])
                                                 ∧ n.succeeded
                   retryReps ≜ {n ∈ reps :
                                  ∧ n.src      ≠ Primary(cViewID[c])
                                  ∧ n.checksum ≠ primaryRep.checksum}
              IN   AckAndSends(m, {[src      ↦ c,
                                    dest     ↦ r,
                                    type     ↦ MReconcileRequest,
                                    viewID   ↦ cViewID[c],
                                    reqID    ↦ m.req.reqID,
                                    index    ↦ primaryRep.index] : n ∈ retryReps})
          ∨ ∧ ¬hasPrimary
            ∧ Ack(m)
        ∧ UNCHANGED ⟨cCommits⟩
        If a quorum has not yet been reached, wait for more replies.
      ∨ ∧ ¬isQuorum
        ∧ isCommitted
        ∧ Ack(m)
        ∧ UNCHANGED ⟨cCommits⟩
  ∧ UNCHANGED ⟨cViewID⟩
∧ UNCHANGED ⟨replicaVars, cTime, cReqID⟩

HandleReconcileReply(c, r, m) ≜ HandleClientReply(c, r, m)
```

This section models the replica protocol.

```
Replica 'r' handles client 'c' request 'm'
HandleClientRequest(r, c, m) ≜
    Client requests can only be handled if the replica is in-sync.
    ∧ rStatus[r] = SInSync
        If the client's view matches the replica's view, process the client's request.
    ∧ ∨ ∧ m.viewID = rViewID[r]
        ∧ LET lastTimestamp ≜ Max({rLog[r][i].timestamp : i ∈ DOMAIN rLog[r]} ∪ {0})
          IN
              If the request timestamp is greater than the highest log timestamp,
              append the entry to the log and return a successful response with
              the appended entry index.
            ∧ ∨ ∧ m.timestamp > lastTimestamp
              ∧ rLog' = [rLog EXCEPT ![r] =
                           Append(rLog[r], [value     ↦ m.value,
                                            timestamp ↦ m.timestamp])]
              ∧ Reply(m, [src       ↦ r,
                          dest      ↦ c,
                          req       ↦ m,
                          type      ↦ MClientReply,
                          viewID    ↦ rViewID[r],
                          index     ↦ Len(rLog'[r]),
                          checksum  ↦ rLog'[r],
                          value     ↦ m.value,
                          timestamp ↦ m.timestamp,
                          succeeded ↦ TRUE])
                If the request timestamp matches the highest log timestamp, treat the
                request as a duplicate. Return a successful response indicating the
                entry was appended.
              ∨ ∧ m.timestamp = lastTimestamp
                ∧ Reply(m, [src       ↦ r,
                            dest      ↦ c,
                            req       ↦ m,
                            type      ↦ MClientReply,
                            viewID    ↦ rViewID[r],
                            index     ↦ Len(rLog[r]),
                            checksum  ↦ rLog[r],
                            value     ↦ m.value,
                            timestamp ↦ m.timestamp,
                            succeeded ↦ TRUE])
                ∧ UNCHANGED ⟨rLog⟩
                If the request timestamp is less than the highest log timestamp,
                reject the request.
              ∨ ∧ m.timestamp < lastTimestamp
                ∧ Reply(m, [src       ↦ r,
                            dest      ↦ c,
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad req \mapsto m, \\
\qquad\qquad\qquad\qquad type \mapsto MClientReply, \\
\qquad\qquad\qquad\qquad viewID \mapsto rViewID[r], \\
\qquad\qquad\qquad\qquad index \mapsto Len(rLog[r]), \\
\qquad\qquad\qquad\qquad checksum \mapsto rLog[r], \\
\qquad\qquad\qquad\qquad value \mapsto m.value, \\
\qquad\qquad\qquad\qquad timestamp \mapsto m.timestamp, \\
\qquad\qquad\qquad\qquad succeeded \mapsto \text{FALSE}]) \\
\qquad\qquad \land \text{UNCHANGED } \langle rLog \rangle \\
\quad \land \text{UNCHANGED } \langle rViewID, rStatus, rViewChangeReps \rangle
\end{array}
$$

If the client's view is greater than the replica's view, reject the client's request with the outdated view ID and enter the view change protocol.

$$
\begin{array}{l}
\lor \land m.viewID > rViewID[r] \\
\quad \land rViewID' = [rViewID \text{ EXCEPT } ![r] = m.viewID] \\
\quad \land rStatus' = [rStatus \text{ EXCEPT } ![r] = SViewChange] \\
\quad \land rViewChangeReps' = [rViewChangeReps \text{ EXCEPT } ![r] = \{\}] \\
\quad \land Replies(m, \{[src \mapsto r, \\
\qquad\qquad dest \mapsto c, \\
\qquad\qquad req \mapsto m, \\
\qquad\qquad type \mapsto MClientReply, \\
\qquad\qquad viewID \mapsto rViewID[r], \\
\qquad\qquad succeeded \mapsto \text{FALSE}], \\
\qquad\qquad [src \mapsto r, \\
\qquad\qquad dest \mapsto Primary(m.viewID), \\
\qquad\qquad type \mapsto MViewChangeReply, \\
\qquad\qquad viewID \mapsto m.viewID, \\
\qquad\qquad logViewID \mapsto rLogViewID[r], \\
\qquad\qquad log \mapsto rLog[r]]\}) \\
\quad \land \text{UNCHANGED } \langle rLog \rangle
\end{array}
$$

If the client's view is less than the replica's view, reject the client's request with the updated view ID to force the client to retry.

$$
\begin{array}{l}
\lor \land m.viewID < rViewID[r] \\
\quad \land Reply(m, [src \mapsto r, \\
\qquad\qquad dest \mapsto c, \\
\qquad\qquad req \mapsto m, \\
\qquad\qquad type \mapsto MClientReply, \\
\qquad\qquad viewID \mapsto rViewID[r], \\
\qquad\qquad succeeded \mapsto \text{FALSE}]) \\
\quad \land \text{UNCHANGED } \langle rViewID, rStatus, rLog, rViewChangeReps \rangle \\
\land \text{UNCHANGED } \langle clientVars, rLogViewID, rSyncIndex \rangle
\end{array}
$$

$$
\begin{array}{l}
HandleReconcileRequest(r, c, m) \triangleq \\
\quad \land rStatus[r] = SInSync \\
\quad \land rViewID[r] = m.viewID \\
\quad \land \lor \land rSyncIndex[r] \geq m.index
\end{array}
$$

$$
\begin{array}{l}
\quad \land \mathit{Reply}(m, [\mathit{src} \mapsto r, \\
\qquad\qquad \mathit{dest} \mapsto c, \\
\qquad\qquad \mathit{req} \mapsto m, \\
\qquad\qquad \mathit{type} \mapsto \mathit{MReconcileReply}, \\
\qquad\qquad \mathit{viewID} \mapsto \mathit{rViewID}[r], \\
\qquad\qquad \mathit{index} \mapsto m.\mathit{index}, \\
\qquad\qquad \mathit{checksum} \mapsto [i \in 1 \,..\, m.\mathit{index} \mapsto \mathit{rLog}[r][i]], \\
\qquad\qquad \mathit{value} \mapsto \mathit{rLog}[r][m.\mathit{index}].\mathit{value}, \\
\qquad\qquad \mathit{timestamp} \mapsto \mathit{rLog}[r][m.\mathit{index}].\mathit{timestamp}, \\
\qquad\qquad \mathit{succeeded} \mapsto \text{TRUE}]) \\
\quad \land \text{UNCHANGED } \langle \mathit{rStatus} \rangle \\
\lor \land \mathit{rSyncIndex}[r] < m.\mathit{index} \\
\quad \land \mathit{Primary}(\mathit{rViewID}[r]) \neq r \\
\quad \land \mathit{rStatus}' = [\mathit{rStatus} \text{ EXCEPT } ![r] = \mathit{SRepair}] \\
\quad \land \mathit{AckAndSend}(m, [\mathit{src} \mapsto r, \\
\qquad\qquad\qquad\qquad \mathit{dest} \mapsto \mathit{Primary}(\mathit{rViewID}[r]), \\
\qquad\qquad\qquad\qquad \mathit{req} \mapsto m, \\
\qquad\qquad\qquad\qquad \mathit{type} \mapsto \mathit{MRepairRequest}, \\
\qquad\qquad\qquad\qquad \mathit{viewID} \mapsto \mathit{rViewID}[r], \\
\qquad\qquad\qquad\qquad \mathit{index} \mapsto m.\mathit{index}]) \\
\land \text{UNCHANGED } \langle \mathit{clientVars}, \mathit{rViewID}, \mathit{rLog}, \mathit{rLogViewID}, \mathit{rSyncIndex}, \mathit{rViewChangeReps} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{HandleRepairRequest}(r, s, m) \triangleq \\
\quad \land \mathit{rStatus}[r] = \mathit{SInSync} \\
\quad \land \mathit{rViewID}[r] = m.\mathit{viewID} \\
\quad \land \mathit{Primary}(\mathit{rViewID}[r]) = r \\
\quad \land \mathit{Reply}(m, [\mathit{src} \mapsto r, \\
\qquad\qquad \mathit{dest} \mapsto s, \\
\qquad\qquad \mathit{req} \mapsto m.\mathit{req}, \\
\qquad\qquad \mathit{type} \mapsto \mathit{MRepairReply}, \\
\qquad\qquad \mathit{viewID} \mapsto \mathit{rViewID}[r], \\
\qquad\qquad \mathit{index} \mapsto m.\mathit{index}, \\
\qquad\qquad \mathit{log} \mapsto [i \in 1 \,..\, m.\mathit{index} \mapsto \mathit{rLog}[r][i]]]) \\
\quad \land \text{UNCHANGED } \langle \mathit{clientVars}, \mathit{replicaVars} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{HandleRepairReply}(r, s, m) \triangleq \\
\quad \land \mathit{rStatus}[r] = \mathit{SRepair} \\
\quad \land \mathit{rViewID}[r] = m.\mathit{viewID} \\
\quad \land \mathit{rStatus}' = [\mathit{rStatus} \text{ EXCEPT } ![r] = \mathit{SInSync}] \\
\quad \land \mathit{rLog}' = [\mathit{rLog} \text{ EXCEPT } ![r] = m.\mathit{log} \circ \mathit{SubSeq}(\mathit{rLog}[r], \mathit{Len}(m.\mathit{log}), \mathit{Len}(\mathit{rLog}[r]))] \\
\quad \land \mathit{rSyncIndex}' = [\mathit{rSyncIndex} \text{ EXCEPT } ![r] = \mathit{Len}(\mathit{rLog}'[r])] \\
\quad \land \mathit{Reply}(m, [\mathit{src} \mapsto r, \\
\qquad\qquad \mathit{dest} \mapsto m.\mathit{req}.\mathit{src}, \\
\qquad\qquad \mathit{req} \mapsto m.\mathit{req}, \\
\qquad\qquad \mathit{type} \mapsto \mathit{MReconcileReply},
\end{array}
$$

```
                    viewID     ↦ rViewID[r],
                    index      ↦ m.index,
                    checksum   ↦ m.log,
                    value      ↦ m.log[m.index].value,
                    timestamp  ↦ m.log[m.index].timestamp,
                    succeeded  ↦ TRUE])
    ∧ UNCHANGED ⟨clientVars, rViewID, rLogViewID, rViewChangeReps⟩

 Replica 'r' requests a view change
ChangeView(r) ≜
    ∧ Sends({[src      ↦ r,
              dest     ↦ d,
              type     ↦ MViewChangeRequest,
              viewID   ↦ rViewID[r] + 1] : d ∈ Replicas})
    ∧ UNCHANGED ⟨clientVars, replicaVars⟩

 Replica 'r' handles replica 's' view change request 'm'
HandleViewChangeRequest(r, s, m) ≜
    ∧ ∨ ∧ rViewID[r] < m.viewID
        ∧ rViewID'          = [rViewID          EXCEPT ![r] = m.viewID]
        ∧ rStatus'          = [rStatus          EXCEPT ![r] = SViewChange]
        ∧ rViewChangeReps'  = [rViewChangeReps  EXCEPT ![r] = {}]
        ∧ Reply(m, [src        ↦ r,
                    dest       ↦ Primary(m.viewID),
                    type       ↦ MViewChangeReply,
                    viewID     ↦ m.viewID,
                    logViewID  ↦ rLogViewID[r],
                    log        ↦ rLog[r]])
      ∨ ∧ rViewID[r] ≥ m.viewID
        ∧ Ack(m)
        ∧ UNCHANGED ⟨rViewID, rStatus, rViewChangeReps⟩
    ∧ UNCHANGED ⟨clientVars, rLog, rLogViewID, rSyncIndex⟩

 Replica 'r' handles replica 's' view change reply 'm'
HandleViewChangeReply(r, s, m) ≜
     The view change protocol is run by the primary for the view.
    ∧ Primary(m.viewID) = r
    ∧ rViewID[r] = m.viewID
    ∧ rStatus[r]   = SViewChange
    ∧ rViewChangeReps' = [rViewChangeReps EXCEPT ![r] = rViewChangeReps[r] ∪ {m}]
    ∧ LET viewChanges ≜ {v ∈ rViewChangeReps'[r] : v.viewID = rViewID[r]}
      IN
          In order to ensure the new view is initialized with the latest view,
          a quorum of view change replies must be received to guarantee the last
          activated view is present in the set of replies.
          If view change replies have been received from a majority of the replicas,
```

```
                initialize the view using the log from the highest activated view.
            ∨ ∧ IsLocalQuorum(r, {v.src : v ∈ viewChanges})
              ∧ LET latestViewID ≜ Max({v.logViewID : v ∈ viewChanges})
                    latestChange ≜ CHOOSE v ∈ viewChanges :
                                     ∧ v.logViewID = latestViewID
                                     ∧ v.src ∈ Quorum(latestViewID)
                IN  AckAndSends(m, {[src    ↦ r,
                                     dest   ↦ d,
                                     type   ↦ MStartViewRequest,
                                     viewID ↦ rViewID[r],
                                     log    ↦ latestChange.log] : d ∈ Replicas})
                If view change replies have not yet been received from a quorum, record
                the view change reply and discard the message.
            ∨ ∧ ¬IsLocalQuorum(r, {v.src : v ∈ viewChanges})
              ∧ Ack(m)
        ∧ UNCHANGED ⟨clientVars, rStatus, rViewID, rLog, rLogViewID, rSyncIndex⟩

    Replica 'r' handles replica 's' start view request 'm'
HandleStartViewRequest(r, s, m) ≜
    To activate a view, the replica must either not know of the view or already
    be participating in the view change protocol for the view.
    ∧ ∨ rViewID[r] < m.viewID
      ∨ ∧ rViewID[r] = m.viewID
        ∧ rStatus[r]  = SViewChange
    If the replica is part of the quorum for the activated view, update the log
    and record the activated view for use in the view change protocol.
    ∧ ∨ ∧ r ∈ Quorum(m.viewID)
        ∧ rLog'          = [rLog          EXCEPT ![r]  = m.log]
        ∧ rLogViewID'    = [rLogViewID    EXCEPT ![r] = m.viewID]
        ∧ rSyncIndex'    = [rSyncIndex    EXCEPT ![r]  = Len(m.log)]
      ∨ ∧ r ∉ Quorum(m.viewID)
        ∧ UNCHANGED ⟨rLog, rLogViewID, rSyncIndex⟩
    Update the replica's view ID and status and clean up view change state.
    ∧ rViewID' = [rViewID EXCEPT ![r] = m.viewID]
    ∧ rStatus' = [rStatus EXCEPT ![r]  = SInSync]
    ∧ LET viewChanges ≜ {v ∈ rViewChangeReps[r] : v.viewID = rViewID[r]}
      IN  rViewChangeReps' = [rViewChangeReps EXCEPT ![r] = rViewChangeReps[r] \ viewChanges]
    ∧ Ack(m)
    ∧ UNCHANGED ⟨clientVars⟩

------------------------------------------------------------------------

InitMessageVars ≜
    ∧ messages     = {}
    ∧ messageCount = 0
    ∧ stepCount    = 0
```

$$
\begin{aligned}
&InitClientVars \triangleq \\
&\quad \land\ cTime \quad\ = 0 \\
&\quad \land\ cViewID \ = [c \in Clients \mapsto 1] \\
&\quad \land\ cReqID \quad = [c \in Clients \mapsto 0] \\
&\quad \land\ cReps \quad\ = [c \in Clients \mapsto \{\}] \\
&\quad \land\ cCommits = [c \in Clients \mapsto \{\}]
\end{aligned}
$$

$$
\begin{aligned}
&InitReplicaVars \triangleq \\
&\quad \land\ rStatus \quad\ = [r \in Replicas \mapsto SInSync] \\
&\quad \land\ rViewID \quad\ = [r \in Replicas \mapsto 1] \\
&\quad \land\ rLog \quad\ = [r \in Replicas \mapsto \langle\rangle] \\
&\quad \land\ rSyncIndex \quad\ = [r \in Replicas \mapsto 0] \\
&\quad \land\ rLogViewID \quad\ = [r \in Replicas \mapsto 1] \\
&\quad \land\ rViewChangeReps = [r \in Replicas \mapsto \{\}]
\end{aligned}
$$

$$
\begin{aligned}
&Init \triangleq \\
&\quad \land\ InitMessageVars \\
&\quad \land\ InitClientVars \\
&\quad \land\ InitReplicaVars
\end{aligned}
$$

This section specifies the invariants for the protocol.

The linearizability invariant verifies that once a client has received matching acks from a quorum and committed a value, thereafter the value is always present at the committed index on all in-sync replicas.

$$
\begin{aligned}
&Linearizability \triangleq \\
&\quad \forall\, c \in Clients : \\
&\qquad \forall\, e \in cCommits[c] : \\
&\qquad\quad \neg\exists\, r \in Replicas : \\
&\qquad\qquad \land\ rStatus[r] \ \ = SInSync \\
&\qquad\qquad \land\ rViewID[r] \geq e.viewID \\
&\qquad\qquad \land\ r \in Quorum(rViewID[r]) \\
&\qquad\qquad \land\ rLog[r][e.index].value \neq e.value
\end{aligned}
$$

$$
\begin{aligned}
&NextClientRequest \triangleq \\
&\quad \exists\, c \in Clients : \\
&\qquad \exists\, v \in Values : \\
&\qquad\quad ClientRequest(c,\, v)
\end{aligned}
$$

$$
\begin{aligned}
&NextChangeView \triangleq \\
&\quad \exists\, r \in Replicas : \\
&\qquad ChangeView(r)
\end{aligned}
$$

$$
NextHandleClientRequest \triangleq
$$

$\exists\, m \in messages :$
$\land\ m.type = MClientRequest$
$\land\ HandleClientRequest(m.dest,\ m.src,\ m)$

$NextHandleClientReply \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MClientReply$
$\land\ HandleClientReply(m.dest,\ m.src,\ m)$

$NextHandleReconcileRequest \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MReconcileRequest$
$\land\ HandleReconcileRequest(m.dest,\ m.src,\ m)$

$NextHandleReconcileReply \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MReconcileReply$
$\land\ HandleReconcileReply(m.dest,\ m.src,\ m)$

$NextHandleRepairRequest \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MRepairRequest$
$\land\ HandleRepairRequest(m.dest,\ m.src,\ m)$

$NextHandleRepairReply \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MRepairReply$
$\land\ HandleRepairReply(m.dest,\ m.src,\ m)$

$NextHandleViewChangeRequest \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MViewChangeRequest$
$\land\ HandleViewChangeRequest(m.dest,\ m.src,\ m)$

$NextHandleViewChangeReply \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MViewChangeReply$
$\land\ HandleViewChangeReply(m.dest,\ m.src,\ m)$

$NextHandleStartViewRequest \triangleq$
$\exists\, m \in messages :$
$\land\ m.type = MStartViewRequest$
$\land\ HandleStartViewRequest(m.dest,\ m.src,\ m)$

$NextDropMessage \triangleq$
$\exists\, m \in messages :$
$\land\ Ack(m)$
$\land\ \text{UNCHANGED}\ \langle clientVars,\ replicaVars \rangle$

$$
\begin{aligned}
Next \triangleq\ & \\
& \lor NextClientRequest \\
& \lor NextChangeView \\
& \lor NextHandleClientRequest \\
& \lor NextHandleClientReply \\
& \lor NextHandleReconcileRequest \\
& \lor NextHandleReconcileReply \\
& \lor NextHandleRepairRequest \\
& \lor NextHandleRepairReply \\
& \lor NextHandleViewChangeRequest \\
& \lor NextHandleViewChangeReply \\
& \lor NextHandleStartViewRequest \\
& \lor NextDropMessage
\end{aligned}
$$

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

\ * Modification History
\ * Last modified *Wed Sep* 30 18:02:32 *PDT* 2020 by *jordanhalterman*
\ * Created *Fri Sep* 18 22:45:21 *PDT* 2020 by *jordanhalterman*