MODULE *JustInTimePaxos*

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

The set of *Paxos* replicas
CONSTANT *Replicas*

The set of *Paxos* clients
CONSTANT *Clients*

The set of possible values
CONSTANT *Values*

An empty value
CONSTANT *Nil*

Request/response types
CONSTANTS
MClientRequest,
MClientReply,
MRepairRequest,
MRepairReply,
MAbortRequest,
MAbortReply,
MViewChangeRequest,
MViewChangeReply,
MStartViewRequest

Replica statuses
CONSTANTS
SNormal,
SAborting,
SViewChange

Entry types
CONSTANTS
TValue,
TNoOp

Message schemas

$ViewIDs \triangleq [viewID \mapsto n \in (1 \,..\,)]$

ClientRequest
$[src \mapsto c \in Clients,$
$dest \mapsto r \in Replicas,$
$type \mapsto MClientRequest,$
$viewID \mapsto i \in ViewIDs,$
$value \mapsto v \in Values,$
$seqNum \mapsto s \in (1 \,..\,),$

$timestamp \mapsto t \in (1 \mathinner{..})]$

ClientReply
$[src \mapsto r \in Replicas,$
$dest \mapsto c \in Clients,$
$type \mapsto MClientReply,$
$viewID \mapsto i \in ViewIDs,$
$value \mapsto v \in Values,$
$seqNum \mapsto s \in (1 \mathinner{..}),$
$index \mapsto i \in (1 \mathinner{..}),$
$succeeded \mapsto \text{TRUE} \lor \text{FALSE}]$

RepairRequest
$[src \mapsto r \in Replicas,$
$dest \mapsto r \in Replicas,$
$type \mapsto MRepairRequest,$
$viewID \mapsto i \in ViewIDs,$
$client \mapsto c \in Clients,$
$seqNum \mapsto s \in (1 \mathinner{..}),$
$timestamp \mapsto t \in (1 \mathinner{..})]$

RepairReply
$[src \mapsto r \in Replicas,$
$dest \mapsto r \in Replicas,$
$type \mapsto MRepairReply,$
$viewID \mapsto i \in ViewIDs,$
$client \mapsto c \in Clients,$
$seqNum \mapsto s \in (1 \mathinner{..}),$
$value \mapsto v \in Values,$
$timestamp \mapsto t \in (1 \mathinner{..})]$

AbortRequest
$[src \mapsto r \in Replicas,$
$dest \mapsto r \in Replicas,$
$type \mapsto MAbortRequest,$
$viewID \mapsto i \in ViewIDs,$
$client \mapsto c \in Clients,$
$seqNum \mapsto s \in (1 \mathinner{..}),$
$timestamp \mapsto t \in (1 \mathinner{..})]$

AbortReply
$[src \mapsto r \in Replicas,$
$dest \mapsto r \in Replicas,$
$type \mapsto MAbortReply,$
$viewID \mapsto i \in ViewIDs,$
$client \mapsto c \in Clients,$
$seqNum \mapsto s \in (1 \mathinner{..})]$

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

$$
\begin{array}{l}
\textit{ViewChangeReply} \\
\quad [\textit{src} \quad \mapsto r \in \textit{Replicas}, \\
\quad\ \textit{dest} \quad \mapsto r \in \textit{Replicas}, \\
\quad\ \textit{type} \quad \mapsto \textit{MViewChangeReply}, \\
\quad\ \textit{viewID} \quad \mapsto i \in \textit{ViewIDs}, \\
\quad\ \textit{lastViewID} \mapsto i \in (1\,..\,), \\
\quad\ \textit{log} \quad \mapsto [c \in \textit{Clients} \mapsto [\, i \in 1\,..\,(1\,..\,) \mapsto [\\
\qquad\qquad\qquad \textit{value} \mapsto v \in \textit{Values}, \textit{timestamp} \mapsto (1\,..\,)]]]] \\
\\
\textit{StartViewRequest} \\
\quad [\textit{src} \quad \mapsto r \in \textit{Replicas}, \\
\quad\ \textit{dest} \quad \mapsto r \in \textit{Replicas}, \\
\quad\ \textit{type} \quad \mapsto \textit{MStartViewRequest}, \\
\quad\ \textit{viewID} \mapsto i \in \textit{ViewIDs}, \\
\quad\ \textit{timestamp} \mapsto t \in (1\,..\,), \textit{log} \quad \mapsto [c \in \textit{Clients} \mapsto [\\
\qquad\qquad i \in 1\,..\,(1\,..\,) \mapsto [\, \textit{value} \mapsto v \in \textit{Values}, \textit{timestamp} \mapsto (1\,..\,)]]]]
\end{array}
$$

A sequence of replicas used for deterministic primary election

VARIABLE $replicas$

$globalVars \triangleq \langle replicas \rangle$

The set of all messages on the network

VARIABLE $messages$

The total number of messages sent

VARIABLE $messageCount$

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

Local client state

Strictly increasing representation of synchronized time

VARIABLE $cTime$

The highest known view ID for a client

VARIABLE $cViewID$

The current sequence number for a client

VARIABLE $cSeqNum$

A client response buffer

VARIABLE $cReps$

A set of all commits - used for model checking

VARIABLE $cCommits$

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

Local replica state

The current status of a replica
VARIABLE $rStatus$

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

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

The current sequence number for each session
VARIABLE $rSeqNum$

The highest known timestamp for all sessions
VARIABLE $rTimestamp$

The last known normal view
VARIABLE $rLastViewID$

The set of received view change responses
VARIABLE $rViewChanges$

The point (*client* + sequence number) in the *log* currently being aborted
VARIABLE $rAbortPoint$

The set of abort responses received
VARIABLE $rAbortReps$

$$replicaVars \triangleq \langle rStatus,\ rLog,\ rViewID,\ rSeqNum,\ rTimestamp,\ rLastViewID,\ rViewChanges,\ rAbortPoint,\ rAbortReps\rangle$$

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

This section provides helpers for the spec.

Creates a sequence from set 'S'
RECURSIVE $SeqFromSet(\_)$

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

Selects an element of set 'S'

$$Pick(S) \triangleq \text{CHOOSE } s \in S : \text{TRUE}$$

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

$$SetReduce(Op(\_, \_),\ S,\ value) \triangleq \text{IF } S = \{\} \text{ THEN}$$

```
        value
    ELSE
        LET s ≜ Pick(S)
        IN  SetReduce(Op, S \ {s}, Op(s, value))

 Computes the greatest vlue in set 'S'
Max(S) ≜ CHOOSE x ∈ S : ∀ y ∈ S : x ≥ y

 Computes the sum of numbers in set 'S'
Sum(S) ≜ LET _op(a, b) ≜ a + b
         IN  SetReduce(_op, S, 0)

 A boolean indicating whether the given set is a quorum
IsQuorum(s) ≜ Cardinality(s) * 2 ≥ Cardinality(Replicas)

 The set of all quorums
Quorums ≜ {r ∈ SUBSET Replicas : IsQuorum(r)}

 The primary for view 'v'
Primary(v) ≜ replicas[(v%Len(replicas)) + (IF v ≥ Len(replicas) THEN 1 ELSE 0)]

 A boolean indicating whether replica 'r' is the primary for the current view
IsPrimary(r) ≜ Primary(rViewID[r]) = r
```

This section models the network.

```
 Send a set of messages
Sends(ms) ≜
    ∧ messages'       = messages ∪ ms
    ∧ messageCount'   = messageCount + Cardinality(ms)

 Send a message
Send(m) ≜ Sends({m})

 Reply to a message with a set of responses
Replies(req, reps) ≜
    ∧ messages'       = (messages ∪ reps) \ {req}
    ∧ messageCount'   = messageCount + Cardinality(reps)

 Reply to a message
Reply(req, resp) ≜ Replies(req, {resp})

 Discard a message
Discard(m) ≜
    ∧ messages'       = messages \ {m}
    ∧ messageCount'   = messageCount + 1
```

This section models client requests.

Client 'c' sends value 'v' to all replicas
Client requests are ordered globally using physical timestamps and locally (within the client) using client sequence numbers. Sequence numbers are sequential and unique within each view.
When the client sends a request is generates a new timestamp. Physical timestamps are modeled here as a strictly increasing global clock simulating synchronized system clocks. The sequence number for the client is also incremented and sent with the request.

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

Client 'c' handles a response 'm' from replica 'r'
When a response is received by the client, if the client is still in the request view it can process the response. The client is responsible for determining commitment by counting responses for each sequence number. Once a response is received from a majority of the replicas including the primary replica, the response is committed. Committed responses are stored in a history variable for checking against invariants.

$$
\begin{array}{l}
HandleClientReply(c,\ r,\ m)\ \triangleq \\
\quad \land\ \lor\ \land\ m.viewID = cViewID[c] \\
\qquad\quad \land\ cReps' = [cReps\ \text{EXCEPT}\ ![c] = cReps[c] \cup \{m\}] \\
\qquad\quad \land\ \text{LET} \\
\qquad\qquad\quad seqNumReps \triangleq \{n \in cReps[c] : n.seqNum = m.seqNum\} \\
\qquad\qquad\quad goodReps \triangleq \{n \in seqNumReps : n.viewID = cViewID[c] \land n.succeeded\} \\
\qquad\qquad\quad isCommitted \triangleq \land \exists\, n \in goodReps : n.src = Primary(n.viewID) \\
\qquad\qquad\qquad\qquad\qquad\quad \land \{n.src : n \in goodReps\} \in Quorums \\
\qquad\qquad \text{IN} \\
\qquad\qquad\quad \land\ \lor\ \land\ isCommitted \\
\qquad\qquad\qquad\qquad \land\ cCommits' = [cCommits\ \text{EXCEPT}\ ![c] = cCommits[c] \cup \\
\qquad\qquad\qquad\qquad\qquad\quad \{\text{CHOOSE}\ n \in goodReps : n.src = Primary(n.viewID)\}] \\
\qquad\qquad\qquad\quad\ \lor\ \land\ \neg isCommitted \\
\qquad\qquad\qquad\qquad \land\ \text{UNCHANGED}\ \langle cCommits\rangle \\
\qquad\qquad\quad \land\ \text{UNCHANGED}\ \langle cViewID,\ cSeqNum\rangle \\
\qquad\quad \lor\ \land\ m.viewID > cViewID[c] \\
\qquad\qquad\quad \land\ cViewID' = [cViewID\ \text{EXCEPT}\ ![c] = m.viewID] \\
\qquad\qquad\quad \land\ cSeqNum' = [cSeqNum\ \text{EXCEPT}\ ![c] = 0]
\end{array}
$$

$\land\ cReps' = [cReps \text{ EXCEPT } ![c] = \{\}]$
$\land$ UNCHANGED $\langle cCommits \rangle$
$\lor\ \land\ m.viewID < cViewID[c]$
$\land$ UNCHANGED $\langle cViewID,\ cSeqNum,\ cReps,\ cCommits \rangle$
$\land\ Discard(m)$
$\land$ UNCHANGED $\langle globalVars,\ replicaVars,\ cTime \rangle$

This section models the replica protocol.

Replica 'r' requests a repair of the client 'c' request 'm'

$$Repair(r,\ c,\ m) \triangleq$$
$$\land\ Replies(m,\ \{[src \mapsto r,\ dest \mapsto d,\ type \mapsto MRepairRequest,\ viewID \mapsto rViewID[r],\ client \mapsto c,\ seqNum \mapsto m.seqNum,\ timestamp \mapsto m.timestamp] : d \in Replicas\})$$

Replica 'r' aborts the client 'c' request 'm'

$$Abort(r,\ c,\ m) \triangleq$$
$$\land\ IsPrimary(r)$$
$$\land\ rStatus[r] = SNormal$$
$$\land\ rStatus' = [rStatus \text{ EXCEPT } ![r] = SAborting]$$
$$\land\ rAbortReps' = [rAbortReps \text{ EXCEPT } ![r] = \{\}]$$
$$\land\ rAbortPoint' = [rAbortPoint \text{ EXCEPT } ![r] = [client \mapsto c,\ seqNum \mapsto m.seqNum]]$$
$$\land\ Replies(m,\ \{[src \mapsto r,\ dest \mapsto d,\ type \mapsto MAbortRequest,\ viewID \mapsto rViewID[r],\ client \mapsto c,\ seqNum \mapsto m.seqNum,\ timestamp \mapsto m.timestamp] : d \in Replicas\})$$

Replica 'r' handles client 'c' request 'm'
Client requests with a view *ID* not matching the replica's view are rejected.
Clients reset their sequence number of
For requests in the correct view, the request must be sequential and linear
to be appended to the *log*. That is, the request must have a 'seqNum' that is
1 + the prior 'seqNum' for the client, and the 'timestamp' must be greater
than all prior timestamps in the *log*. This is necessary to ensure the primary
log does not change when requests are reordered. The client can retry requests
that are reordered with a new sequence number and timestamp.
To maintain consistency within the *log*, a separate sequence is maintained for
each session (client), and each sequence number is assigned to a unique

position in the session *log*. Session logs are logically merged into a totally ordered *log* using the request timestamps.

When a sequence number is skipped, the primary must commit a *TNoOp* entry to the *log*. It does so by running the *AbortRequest* protocol.

When a sequence number is skipped on a non-primary replica, the replica attempts to recover the request using the *RepairRequest* protocol.

$$
\begin{array}{l}
HandleClientRequest(r,\ c,\ m) \triangleq \\
\quad \land\ rStatus[r] = SNormal \\
\quad \land\ \lor\ \land\ m.viewID = rViewID[r] \\
\qquad \land\ \text{LET} \\
\qquad\quad lastIndex \triangleq Sum(\{Len(rLog[r][i]) : i \in Clients\}) \\
\qquad\quad index \triangleq lastIndex + 1 \\
\qquad\quad lastTimestamp \triangleq rTimestamp[r] \\
\qquad\quad isSequential \triangleq m.seqNum = rSeqNum[r][c] + 1 \\
\qquad\quad isLinear \triangleq m.timestamp > lastTimestamp \\
\qquad\quad entry \triangleq [type \mapsto TValue, \\
\qquad\qquad\qquad\qquad index \mapsto index, \\
\qquad\qquad\qquad\qquad value \mapsto m.value, \\
\qquad\qquad\qquad\qquad timestamp \mapsto m.timestamp] \\
\qquad\quad append(e) \triangleq [rLog\ \text{EXCEPT}\ ![r] = [rLog[r]\ \text{EXCEPT} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ![c] = Append(rLog[r][c],\ e)]] \\
\qquad \text{IN} \\
\qquad\quad \lor\ \land\ isSequential \\
\qquad\qquad\quad \land\ isLinear \\
\qquad\qquad\quad \land\ rLog' = append(entry) \\
\qquad\qquad\quad \land\ rSeqNum' = [rSeqNum\ \text{EXCEPT}\ ![r] = \\
\qquad\qquad\qquad\qquad\qquad\qquad [rSeqNum[r]\ \text{EXCEPT}\ ![c] = m.seqNum]] \\
\qquad\qquad\quad \land\ rTimestamp' = [rTimestamp\ \text{EXCEPT}\ ![r] = m.timestamp] \\
\qquad\qquad\quad \land\ Reply(m, [src \mapsto r, \\
\qquad\qquad\qquad\qquad\qquad\qquad dest \mapsto c, \\
\qquad\qquad\qquad\qquad\qquad\qquad type \mapsto MClientReply, \\
\qquad\qquad\qquad\qquad\qquad\qquad viewID \mapsto rViewID[r], \\
\qquad\qquad\qquad\qquad\qquad\qquad seqNum \mapsto m.seqNum, \\
\qquad\qquad\qquad\qquad\qquad\qquad index \mapsto index, \\
\qquad\qquad\qquad\qquad\qquad\qquad value \mapsto m.value, \\
\qquad\qquad\qquad\qquad\qquad\qquad succeeded \mapsto \text{TRUE}]) \\
\qquad\qquad\quad \land\ \text{UNCHANGED}\ \langle rStatus,\ rAbortPoint,\ rAbortReps\rangle \\
\qquad\quad \lor\ \land\ \lor\ \land\ \neg isSequential \\
\qquad\qquad\qquad\qquad\ \land\ m.seqNum > rSeqNum[r][c] + 1 \\
\qquad\qquad\qquad\ \lor\ \neg isLinear \\
\qquad\qquad\quad \land\ \lor\ \land\ IsPrimary(r) \\
\qquad\qquad\qquad\qquad\ \land\ Abort(r,\ c,\ m) \\
\qquad\qquad\qquad\ \lor\ \land\ \neg IsPrimary(r) \\
\qquad\qquad\qquad\qquad\ \land\ Repair(r,\ c,\ m) \\
\qquad\qquad\qquad\qquad\ \land\ \text{UNCHANGED}\ \langle rStatus,\ rAbortPoint,\ rAbortReps\rangle
\end{array}
$$

```
                    ∧ UNCHANGED ⟨rLog, rSeqNum, rTimestamp⟩
        ∨ ∧ m.viewID < rViewID[r]
          ∧ Reply(m, [src       ↦ r,
                      dest      ↦ c,
                      type      ↦ MClientReply,
                      viewID    ↦ rViewID[r],
                      seqNum    ↦ m.seqNum,
                      succeeded ↦ FALSE])
          ∧ UNCHANGED ⟨rStatus, rLog, rSeqNum, rTimestamp, rAbortPoint, rAbortReps⟩
    ∧ UNCHANGED ⟨globalVars, clientVars, rViewID, rLastViewID, rViewChanges⟩

Replica 'r' handles replica 's' repair request 'm'
When a repair request is received, if the requested sequence number is in the session
log, the entry is returned. Otherwise, the primary aborts the request.
HandleRepairRequest(r, s, m) ≜
    ∧ m.viewID = rViewID[r]
    ∧ IsPrimary(r)
    ∧ rStatus[r] = SNormal
    ∧ LET offset ≜ Len(rLog[r][m.client]) − (rSeqNum[r][m.client] − m.seqNum)
      IN
        ∨ ∧ offset ≤ Len(rLog[r][m.client])
          ∧ Reply(m, [src       ↦ r,
                      dest      ↦ s,
                      type      ↦ MRepairReply,
                      viewID    ↦ rViewID[r],
                      client    ↦ m.client,
                      seqNum    ↦ m.seqNum,
                      value     ↦ rLog[r][m.client][offset].value,
                      timestamp ↦ rLog[r][m.client][offset].timestamp])
          ∧ UNCHANGED ⟨rStatus, rAbortPoint, rAbortReps⟩
        ∨ ∧ offset = Len(rLog[r][m.client]) + 1
          ∧ Abort(r, m.client, m)
    ∧ UNCHANGED ⟨globalVars, clientVars, rLog, rSeqNum, rTimestamp, rViewID, rLastViewID, rViewChan

Replica 'r' handles replica 's' repair response 'm'
Repair responses are handled like client requests.
HandleRepairReply(r, s, m) ≜
    HandleClientRequest(r, m.client, [m EXCEPT !.src = m.client])

Replica 'r' handles replica 's' abort request 'm'
If the aborted sequence number is in the session log, the entry is replaced with
a no-op entry. If the sequence number can be appebded to the log, it is.
HandleAbortRequest(r, s, m) ≜
    ∧ m.viewID = rViewID[r]
    ∧ rStatus[r] ∈ {SNormal, SAborting}
    ∧ LET
```

```
        offset          ≜ Len(rLog[r][m.client]) − (rSeqNum[r][m.client] − m.seqNum)
        entry           ≜ [type ↦ TNoOp, timestamp ↦ 0]
        replace(l, i, e) ≜ [j ∈ 1 .. Max({Len(l), i}) ↦ IF j = i THEN e ELSE l[j]]
    IN
        ∧ ∨ ∧ offset ≤ Len(rLog[r][m.client])
            ∧ rLog' = [rLog EXCEPT ![r] = [
                          rLog[r] EXCEPT ![m.client] = [
                          rLog[r][m.client] EXCEPT ![offset] = [
                          rLog[r][m.client][offset] EXCEPT !.type = TNoOp]]]]
            ∧ UNCHANGED ⟨rTimestamp, rSeqNum⟩
          ∨ ∧ offset = Len(rLog[r][m.client]) + 1
            ∧ rLog' = [rLog EXCEPT ![r] = [
                          rLog[r] EXCEPT ![m.client] =
                             Append(rLog[r][m.client], [type ↦ TNoOp, timestamp ↦ 0])]]
            ∧ rTimestamp' = [rTimestamp EXCEPT ![r] = Max({rTimestamp[r], m.timestamp})]
            ∧ rSeqNum'    = [rSeqNum    EXCEPT ![r] = [
                               rSeqNum[r] EXCEPT ![m.client] = m.seqNum]]
        ∧ Replies(m, {[src       ↦ r,
                       dest      ↦ Primary(rViewID[r]),
                       type      ↦ MAbortReply,
                       viewID    ↦ rViewID[r],
                       client    ↦ m.client,
                       seqNum    ↦ m.seqNum],
                      [src       ↦ r,
                       dest      ↦ m.client,
                       type      ↦ MClientReply,
                       viewID    ↦ rViewID[r],
                       seqNum    ↦ m.seqNum,
                       succeeded ↦ FALSE]})
    ∧ UNCHANGED ⟨globalVars, clientVars, rStatus, rAbortPoint,
                 rAbortReps, rViewID, rLastViewID, rViewChanges⟩

 Replica 'r' handles replica 's' repair response 'm'
HandleAbortReply(r, s, m) ≜
    ∧ rStatus[r] = SAborting
    ∧ m.viewID = rViewID[r]
    ∧ IsPrimary(r)
    ∧ m.seqNum = rAbortPoint[r].seqNum
    ∧ rAbortReps' = [rAbortReps EXCEPT ![r] = rAbortReps[r] ∪ {m}]
    ∧ LET reps ≜ {res.src : res ∈ {resp ∈ rAbortReps'[r] :
                       ∧ resp.viewID  = rViewID[r]
                       ∧ resp.client  = rAbortPoint[r].client
                       ∧ resp.seqNum = rAbortPoint[r].seqNum}}
          isQuorum ≜ r ∈ reps ∧ reps ∈ Quorums
      IN
```

$\lor \land isQuorum$
$\land rStatus' = [rStatus \text{ EXCEPT } ![r] = SNormal]$
$\lor \land \neg isQuorum$
$\land \text{UNCHANGED } \langle rStatus \rangle$
$\land \text{UNCHANGED } \langle globalVars, messageVars, clientVars, rLog, rSeqNum, rTimestamp, rAbortPoint, rViewID, rViewChanges, rLastViewID \rangle$

Replica 'r' requests a view change
The view change is requested by sending a *ViewChangeRequest* to each replica.

$ChangeView(r) \triangleq$
$\land Sends(\{[src \mapsto r, dest \mapsto d, type \mapsto MViewChangeRequest, viewID \mapsto rViewID[r] + 1] : d \in Replicas\})$
$\land \text{UNCHANGED } \langle globalVars, clientVars, replicaVars \rangle$

Replica 'r' handles replica 's' view change request 'm'
Replicas respond to *ViewChangeRequests* with the contents of their logs for reconciliation. When a new view change is requested, the replica updates its *ViewID* and transitions to the *ViewChange* status to block writes during the transition.

$HandleViewChangeRequest(r, s, m) \triangleq$
$\land rViewID[r] < m.viewID$
$\land rViewID' = [rViewID \text{ EXCEPT } ![r] = m.viewID]$
$\land rStatus' = [rStatus \text{ EXCEPT } ![r] = SViewChange]$
$\land rViewChanges' = [rViewChanges \text{ EXCEPT } ![r] = \{\}]$
$\land Reply(m, [src \mapsto r, dest \mapsto Primary(m.viewID), type \mapsto MViewChangeReply, viewID \mapsto m.viewID, lastViewID \mapsto rLastViewID[r], log \mapsto rLog[r]])$
$\land \text{UNCHANGED } \langle globalVars, clientVars, rLog, rSeqNum, rTimestamp, rAbortPoint, rAbortReps, rLastViewID \rangle$

Replica 'r' handles replica 's' view change response 'm'
ViewChangeReplys are handled by the primary for the new view. Once responses are received from a majority of the replicas including the new primary, the logs received from each replica are merged together to form the *log* for the new view. For each known session, the logs from each replica are merged by comparing each entry and keeping all non-empty sequential entries in the quorum. An updated timestamp is calculated from the reconciled *log*, and a *StartViewRequest* containing the new logs is sent to each replica.

$HandleViewChangeReply(r, s, m) \triangleq$
$\land IsPrimary(r)$
$\land rViewID[r] = m.viewID$

$$
\begin{array}{l}
\land\ rStatus[r] = SViewChange \\
\land\ rViewChanges' = [rViewChanges \text{ EXCEPT } ![r] = rViewChanges[r] \cup \{m\}] \\
\land \text{ LET } viewChanges \triangleq \{v \in rViewChanges'[r] : v.viewID = rViewID[r]\} \\
\qquad viewSources \triangleq \{v.src : v \in viewChanges\} \\
\qquad isQuorum \triangleq r \in viewSources \land viewSources \in Quorums \\
\qquad lastViewIDs \triangleq \{v.lastViewID : v \in viewChanges\} \\
\qquad lastViewID \triangleq (\text{CHOOSE } v1 \in lastViewIDs : \forall\, v2 \in lastViewIDs : v2 \leq v1) \\
\qquad lastViewChanges \triangleq \{v2 \in viewChanges : v2.lastViewID = lastViewID\} \\
\qquad viewLogs \triangleq [c \in Clients \mapsto \{v1.log[c] : v1 \in lastViewChanges\}] \\
\qquad mergeEnts(es) \triangleq \\
\qquad\quad \text{IF } es = \{\} \lor \exists\, e \in es : e.type = TNoOp \text{ THEN} \\
\qquad\qquad \text{CHOOSE } e \in es : e.type = TNoOp \\
\qquad\quad \text{ELSE} \\
\qquad\qquad \text{CHOOSE } e \in es : e.type \neq TNoOp \\
\qquad range(ls) \triangleq Max(\{Len(l) : l \in ls\}) \\
\qquad entries(ls,\, i) \triangleq \{l[i] : l \in \{k \in ls : i \leq Len(k)\}\} \\
\qquad mergeLogs(ls) \triangleq [i \in 1 \,..\, range(ls) \mapsto mergeEnts(entries(ls,\, i))] \\
\qquad viewLog \triangleq [c \in Clients \mapsto mergeLogs(viewLogs[c])] \\
\qquad viewRange \triangleq Max(\{Len(viewLog[c]) : c \in Clients\}) \\
\qquad viewTimestamp \triangleq \text{IF } viewRange > 0 \text{ THEN} \\
\qquad\qquad\qquad Max(\text{UNION } \{\{l[i].timestamp : i \in \text{DOMAIN } l\} : \\
\qquad\qquad\qquad\qquad l \in \{viewLog[c] : c \in Clients\}\}) \\
\qquad\qquad\qquad\quad \text{ELSE } 0 \\
\text{IN} \\
\qquad \lor \land isQuorum \\
\qquad\quad \land Replies(m, \{[src \mapsto r, \\
\qquad\qquad\qquad dest \mapsto d, \\
\qquad\qquad\qquad type \mapsto MStartViewRequest, \\
\qquad\qquad\qquad viewID \mapsto rViewID[r], \\
\qquad\qquad\qquad timestamp \mapsto viewTimestamp, \\
\qquad\qquad\qquad log \mapsto viewLog] : d \in Replicas\}) \\
\qquad \lor \land \neg isQuorum \\
\qquad\quad \land Discard(m) \\
\land \text{ UNCHANGED } \langle globalVars,\, clientVars,\, rStatus,\, rViewID,\, rLog,\, rSeqNum, \\
\qquad\qquad rTimestamp,\, rAbortPoint,\, rAbortReps,\, rLastViewID\rangle
\end{array}
$$

Replica 'r' handles replica 's' start view request 'm'

If the view is new, the replica updates its logs and session state from the request.

$$
\begin{array}{l}
HandleStartViewRequest(r,\, s,\, m) \triangleq \\
\land\ \lor\ rViewID[r] < m.viewID \\
\qquad \lor \land rViewID[r] = m.viewID \\
\qquad\quad \land rStatus[r] = SViewChange \\
\land\ rLog' = [rLog \text{ EXCEPT } ![r] = m.log] \\
\land\ rSeqNum' = [rSeqNum \text{ EXCEPT } ![r] = [c \in Clients \mapsto 0]] \\
\land\ rTimestamp' = [rTimestamp \text{ EXCEPT } ![r] = m.timestamp]
\end{array}
$$

$\land\ rStatus' = [rStatus \text{ EXCEPT } ![r] = SNormal]$
$\land\ rViewID' = [rViewID \text{ EXCEPT } ![r] = m.viewID]$
$\land\ rLastViewID' = [rLastViewID \text{ EXCEPT } ![r] = m.viewID]$
$\land\ Discard(m)$
$\land$ UNCHANGED $\langle globalVars, clientVars, rAbortPoint, rAbortReps, rViewChanges\rangle$

$InitMessageVars \triangleq$
$\land\ messages = \{\}$
$\land\ messageCount = 0$

$InitClientVars \triangleq$
$\land\ cTime = 0$
$\land\ cViewID = [c \in Clients \mapsto 1]$
$\land\ cSeqNum = [c \in Clients \mapsto 0]$
$\land\ cReps = [c \in Clients \mapsto \{\}]$
$\land\ cCommits = [c \in Clients \mapsto \{\}]$

$InitReplicaVars \triangleq$
$\land\ replicas = SeqFromSet(Replicas)$
$\land\ rStatus = [r \in Replicas \mapsto SNormal]$
$\land\ rLog = [r \in Replicas \mapsto [c \in Clients \mapsto \langle\rangle]]$
$\land\ rSeqNum = [r \in Replicas \mapsto [c \in Clients \mapsto 0]]$
$\land\ rTimestamp = [r \in Replicas \mapsto 0]$
$\land\ rAbortPoint = [r \in Replicas \mapsto [client \mapsto Nil, seqNum \mapsto 0]]$
$\land\ rAbortReps = [r \in Replicas \mapsto \{\}]$
$\land\ rViewID = [r \in Replicas \mapsto 1]$
$\land\ rLastViewID = [r \in Replicas \mapsto 1]$
$\land\ rViewChanges = [r \in Replicas \mapsto \{\}]$

$Init \triangleq$
$\land\ InitMessageVars$
$\land\ InitClientVars$
$\land\ InitReplicaVars$

The type invariant verifies that clients do not receive two commits at the same index with different values.

$TypeOK \triangleq$
$\forall\, c1, c2 \in Clients :$
$\forall\, e1 \in cCommits[c1] :$
$\neg\exists\, e2 \in cCommits[c2] :$
$\land\ e1.index = e2.index$
$\land\ e1.value \neq e2.value$

$$
\begin{array}{l}
Next \triangleq \\
\quad \lor \exists\, c \in Clients : \\
\qquad \exists\, v \in Values : \\
\qquad\quad \land\, ClientRequest(c,\, v) \\
\quad \lor \exists\, r \in Replicas : \\
\qquad \land\, ChangeView(r) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MClientRequest \\
\qquad \land\, HandleClientRequest(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MClientReply \\
\qquad \land\, HandleClientReply(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MRepairRequest \\
\qquad \land\, HandleRepairRequest(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MRepairReply \\
\qquad \land\, HandleRepairReply(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MAbortRequest \\
\qquad \land\, HandleAbortRequest(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MAbortReply \\
\qquad \land\, HandleAbortReply(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MViewChangeRequest \\
\qquad \land\, HandleViewChangeRequest(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MViewChangeReply \\
\qquad \land\, HandleViewChangeReply(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, m.type = MStartViewRequest \\
\qquad \land\, HandleStartViewRequest(m.dest,\, m.src,\, m) \\
\quad \lor \exists\, m \in messages : \\
\qquad \land\, Discard(m) \\
\qquad \land\, \text{UNCHANGED } \langle globalVars,\, clientVars,\, replicaVars \rangle
\end{array}
$$

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

* Modification History
* Last modified *Wed Sep* 23 19:09:39 *PDT* 2020 by *jordanhalterman*
* Created *Fri Sep* 18 22:45:21 *PDT* 2020 by *jordanhalterman*