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.

Summary:

* View-based protocol
* Views are identified by a monotonically increasing, globally unique identifier
* Each view assigns a primary plus a set of replicas that form the quorum
* Clients timestamp each request and send in parallel to all replicas in the quorum
* Each replica appends requests to a log in chronological order, and the primary executes requests
* If a request is received out of chronological order it is rejected
* Replies to clients include a checksum of the log on each replica
* If the client receives a reply indicating the request was received out of chronological order or if a checksum does not match the primary's checksum, the client initiates a reconciliation protocol
* To reconcile inconsistencies in the log, replicas pull logs from the primary, and once logs have been reconciled the original request is acknowledged
* Once the client receives matching acknowledgements from all the replicas in the quorum a request is committed
* View changes select the most recent log from a majority of the replicas to ensure the initial view log contains all committed requests from prior views

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.

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

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

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

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

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

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

ClientRequest

$$\begin{array}{l} [\ \mathit{src} \mapsto c \in \mathit{Clients}, \\ \quad \mathit{dest} \mapsto r \in \mathit{Replicas}, \\ \quad \mathit{type} \mapsto \mathit{MClientRequest}, \\ \quad \mathit{viewID} \mapsto i \in \mathit{ViewIDs}, \\ \quad \mathit{reqID} \mapsto i \in \mathit{ReqIDs}, \\ \quad \mathit{value} \mapsto v \in \mathit{Values}, \\ \quad \mathit{timestamp} \mapsto t \in \text{Timestamps}\ ] \end{array}$$

ClientReply

$$\begin{array}{l} [\ \mathit{src} \mapsto r \in \mathit{Replicas}, \\ \quad \mathit{dest} \mapsto c \in \mathit{Clients}, \\ \quad \mathit{req} \mapsto (\mathit{ClientRequest}), \\ \quad \mathit{type} \mapsto \mathit{MClientReply}, \\ \quad \mathit{viewID} \mapsto i \in \mathit{ViewIDs}, \\ \quad \mathit{index} \mapsto i \in \text{Indexes}, \\ \quad \mathit{checksum} \mapsto c \in \text{Checksums}, \\ \quad \mathit{value} \mapsto v \in \mathit{Values}, \\ \quad \mathit{timestamp} \mapsto t \in \text{Timestamps}, \\ \quad \mathit{succeeded} \mapsto \textsc{true} \lor \textsc{false}\ ] \end{array}$$

ReconcileRequest

$$\begin{array}{l} [\ \mathit{src} \mapsto c \in \mathit{Clients}, \\ \quad \mathit{dest} \mapsto r \in \mathit{Replicas}, \\ \quad \mathit{type} \mapsto \mathit{MReconcileRequest}, \\ \quad \mathit{viewID} \mapsto i \in \mathit{ViewIDs}, \\ \quad \mathit{reqID} \mapsto i \in \mathit{ReqIDs}, \\ \quad \mathit{index} \mapsto i \in \text{Indexes}\ ] \end{array}$$

ReconcileReply

$$\begin{array}{l} [\ \mathit{src} \mapsto r \in \mathit{Replicas}, \\ \quad \mathit{dest} \mapsto c \in \mathit{Clients}, \\ \quad \mathit{req} \mapsto (\mathit{ClientRequest}), \\ \quad \mathit{type} \mapsto \mathit{MReconcileReply}, \\ \quad \mathit{viewID} \mapsto i \in \mathit{ViewIDs}, \\ \quad \mathit{index} \mapsto i \in \text{Indexes}, \\ \quad \mathit{checksum} \mapsto c \in \text{Checksums}, \\ \quad \mathit{value} \mapsto v \in \mathit{Values}, \\ \quad \mathit{timestamp} \mapsto t \in \text{Timestamps}, \\ \quad \mathit{succeeded} \mapsto \textsc{true} \lor \textsc{false}\ ] \end{array}$$

RepairRequest

$$\begin{array}{l} [\ \mathit{src} \mapsto r \in \mathit{Replicas}, \\ \quad \mathit{dest} \mapsto r \in \mathit{Replicas}, \\ \quad \mathit{req} \mapsto (\mathit{ClientRequest}), \\ \quad \mathit{type} \mapsto \mathit{MRepairRequest}, \\ \quad \mathit{viewID} \mapsto i \in \mathit{ViewIDs}, \\ \quad \mathit{index} \mapsto i \in \text{Indexes}\ ] \end{array}$$

RepairReply

$$[\ \mathit{src} \mapsto r \in \mathit{Replicas},$$

$dest \mapsto r \in Replicas$,
$req \mapsto (ClientRequest)$,
$type \mapsto MRepairReply$,
$viewID \mapsto i \in ViewIDs$,
$index \mapsto i \in$ 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'

$$Max(S) \triangleq \textsc{choose}\ x \in S : \forall\, y \in S : x \geq y$$

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

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

This section provides helpers for the protocol.

A sorted sequence of replicas

$$replicas \triangleq SeqFromSet(Replicas)$$

The primary index for view 'v'

$$PrimaryIndex(v) \triangleq (v \% Len(replicas)) + (\textsc{if}\ v \geq Len(replicas)\ \textsc{then}\ 1\ \textsc{else}\ 0)$$

The primary for view 'v'

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

Quorum is the quorum for a given view

```
$Quorum(v) \triangleq$
    LET
        $quorumSize \triangleq Len(replicas) \div 2 \quad + 1$
        $index(i) \triangleq PrimaryIndex(v) + (i - 1)$
        $member(i) \triangleq$ 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

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

This section models the network.

Messages between processes are unordered and can be dropped by the network at any time.

Send a set of messages

```
$Sends(ms) \triangleq$
    $\land\ messages' = messages \cup ms$
    $\land\ messageCount' = messageCount + Cardinality(ms)$
    $\land\ stepCount' = stepCount + 1$
```

Send a message

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

Ack a message

```
$Ack(m) \triangleq$
    $\land messages' = messages \setminus \{m\}$
    $\land messageCount' = messageCount + 1$
    $\land stepCount' = stepCount + 1$
```

Ack a message and send a set of messages

```
$AckAndSends(m, ms) \triangleq$
    $\land messages' = (messages \cup ms) \setminus \{m\}$
    $\land messageCount' = messageCount + Cardinality(ms)$
    $\land stepCount' = stepCount + 1$
```

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 *JITPaxos* clients.

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

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

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

```
$HandleClientReply(c, r, m) \triangleq$
        If the reply view ID does not match the request view ID, update the client's view.
    $\land \lor \land m.viewID \neq m.req.viewID$
        $\land \lor \land cViewID[c] < m.viewID$
                $\land cViewID' = [cViewID \text{ EXCEPT } ![c] = m.viewID]$
            $\lor \land cViewID[c] \geq m.viewID$
        $\land$ UNCHANGED $\langle cViewID \rangle$
    $\land$ $Ack(m)$
    $\land$ UNCHANGED $\langle cReps,\ cCommits \rangle$
If the request and reply views match and the reply view matches the client's view,
aggregate the replies for the associated client request.
$\lor$  $\land$ $m.viewID = m.req.viewID$
    $\land$ $m.viewID = cViewID[c]$
    $\land$ $\lor$ $\land$ $m.succeeded$
           $\land$ $cReps' = [cReps$ EXCEPT $![c] =$
                  $(cReps[c] \setminus \{n \in cReps[c] :$ $\land$ $n.src = m.src$
                                                        $\land$ $n.viewID = cViewID[c]$
                                                        $\land$ $n.req.reqID = m.req.reqID$
                                                        $\land$ $\neg n.succeeded\}) \cup \{m\}]$
       $\lor$ $\land$ $\neg m.succeeded$
           $\land$ $\neg \exists\, n \in cReps[c] :$ $\land$ $n.src = m.src$
                                        $\land$ $n.viewID = cViewID[c]$
                                        $\land$ $n.req.reqID = m.req.reqID$
                                        $\land$ $n.succeeded$
           $\land$ $cReps' = [cReps$ EXCEPT $![c] = cReps[c] \cup \{m\}]$
    $\land$ LET $reps \triangleq \{n \in cReps'[c] :$ $\land$ $n.viewID = cViewID[c]$
                                          $\land$ $n.req.reqID = m.req.reqID\}$
          $isQuorum \triangleq \{n.src : n \in \{n \in reps : n.succeeded\}\} = Quorum(cViewID[c])$
          $isCommitted \triangleq$ $\land$ $\forall\, n \in reps : n.succeeded$
                          $\land$ $Cardinality(\{n.checksum : n \in reps\}) = 1$
          $hasPrimary \triangleq \exists\, n \in reps : n.src = Primary(cViewID[c]) \land n.succeeded$
      IN
          If a quorum of successful replies have been received and the checksums
          match, add the primary reply to commits.
          $\lor$ $\land$ $isQuorum$
             $\land$ $isCommitted$
             $\land$ LET $commit \triangleq$ CHOOSE $n \in reps : n.src = Primary(cViewID[c])$
                 IN $cCommits' = [cCommits$ EXCEPT $![c] = cCommits[c] \cup \{commit\}]$
             $\land$ $Ack(m)$
          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*.
          $\lor$ $\land$ $\neg isCommitted$
             $\land$ $\lor$ $\land$ $hasPrimary$
                   $\land$ LET $primaryRep \triangleq$ CHOOSE $n \in reps :$ $\land$ $n.src = Primary(cViewID[c])$
                                                            $\land$ $n.succeeded$
                           $retryReps \triangleq \{n \in reps :$
                                          $\land$ $n.src \neq Primary(cViewID[c])$
                                          $\land$ $n.checksum \neq primaryRep.checksum\}$
                       IN  $AckAndSends(m, \{[src \mapsto c,$
                                          $dest \mapsto r,$
                                        $type \mapsto MReconcileRequest,$
                                        $viewID \mapsto cViewID[c],$
                                        $reqID \mapsto m.req.reqID,$
                                        $index \mapsto primaryRep.index] : n \in retryReps\})$
                    $\lor \land \neg hasPrimary$
                      $\land Ack(m)$
                  $\land$ UNCHANGED $\langle cCommits \rangle$
                  If a quorum has not yet been reached, wait for more replies.
                $\lor \land \neg isQuorum$
                  $\land isCommitted$
                  $\land Ack(m)$
                  $\land$ UNCHANGED $\langle cCommits \rangle$
            $\land$ UNCHANGED $\langle cViewID \rangle$
        $\land$ UNCHANGED $\langle replicaVars, cTime, cReqID \rangle$

$HandleReconcileReply(c, r, m) \triangleq HandleClientReply(c, r, m)$
```

This section models *JITPaxos* replicas.

```
Replica 'r' handles client 'c' request 'm'
$HandleClientRequest(r, c, m) \triangleq$
    Client requests can only be handled if the replica is in-sync.
    $\land rStatus[r] = SInSync$
      If the client's view matches the replica's view, process the client's request.
    $\land \lor \land m.viewID = rViewID[r]$
        $\land$ LET $lastTimestamp \triangleq Max(\{rLog[r][i].timestamp : i \in$ DOMAIN $rLog[r]\} \cup \{0\})$
          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.
            $\land \lor \land m.timestamp > lastTimestamp$
                  $\land rLog' = [rLog$ EXCEPT $![r] =$
                          $Append(rLog[r], [value \mapsto m.value,$
                                          $timestamp \mapsto m.timestamp])]$
                  $\land Reply(m, [src \mapsto r,$
                              $dest \mapsto c,$
                              $req \mapsto m,$
                              $type \mapsto MClientReply,$
                              $viewID \mapsto rViewID[r],$
                              $index \mapsto Len(rLog'[r]),$
                              $checksum \mapsto rLog'[r],$
                              $value \mapsto m.value,$
                              $timestamp \mapsto m.timestamp,$
                              $succeeded \mapsto$ 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.
          $\lor \land m.timestamp = lastTimestamp$
            $\land Reply(m, [src \mapsto r,$
                            $dest \mapsto c,$
                            $req \mapsto m,$
                            $type \mapsto MClientReply,$
                            $viewID \mapsto rViewID[r],$
                            $index \mapsto Len(rLog[r]),$
                            $checksum \mapsto rLog[r],$
                            $value \mapsto m.value,$
                            $timestamp \mapsto m.timestamp,$
                            $succeeded \mapsto \text{TRUE}])$
            $\land \text{UNCHANGED } \langle rLog \rangle$
          If the request timestamp is less than the highest log timestamp,
          reject the request.
          $\lor \land m.timestamp < lastTimestamp$
            $\land Reply(m, [src \mapsto r,$
                            $dest \mapsto c,$
                            $req \mapsto m,$
                            $type \mapsto MClientReply,$
                            $viewID \mapsto rViewID[r],$
                            $index \mapsto Len(rLog[r]),$
                            $checksum \mapsto rLog[r],$
                            $value \mapsto m.value,$
                            $timestamp \mapsto m.timestamp,$
                            $succeeded \mapsto \text{FALSE}])$
            $\land \text{UNCHANGED } \langle rLog \rangle$
      $\land \text{UNCHANGED } \langle rViewID, rStatus, rViewChangeReps \rangle$
      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.
      $\lor \land m.viewID > rViewID[r]$
        $\land rViewID' = [rViewID \text{ EXCEPT } ![r] = m.viewID]$
        $\land rStatus' = [rStatus \text{ EXCEPT } ![r] = SViewChange]$
        $\land rViewChangeReps' = [rViewChangeReps \text{ EXCEPT } ![r] = \{\}]$
        $\land Replies(m, \{[src \mapsto r,$
                            $dest \mapsto c,$
                            $req \mapsto m,$
                            $type \mapsto MClientReply,$
                            $viewID \mapsto rViewID[r],$
                            $succeeded \mapsto \text{FALSE}],$
                           $[src \mapsto r,$
                            $dest \mapsto Primary(m.viewID),$
                            $type \mapsto MViewChangeReply,$
                            $viewID \mapsto m.viewID,$
```

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

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

$HandleRepairRequest(r, s, m) \triangleq$
    $\land rStatus[r] = SInSync$
    $\land rViewID[r] = m.viewID$
    $\land Primary(rViewID[r]) = r$
    $\land\ Reply(m,\ [src \mapsto r,$
                    $dest \mapsto s,$
                    $req \mapsto m.req,$
                    $type \mapsto MRepairReply,$
                    $viewID \mapsto rViewID[r],$
                    $index \mapsto m.index,$
                    $log \mapsto [i \in 1 \,..\, m.index \mapsto rLog[r][i]]])$
    $\land$ UNCHANGED $\langle clientVars,\ replicaVars \rangle$

$HandleRepairReply(r,\ s,\ m)\ \triangleq$
    $\land\ rStatus[r] = SRepair$
    $\land\ rViewID[r] = m.viewID$
    $\land\ rStatus' = [rStatus$ EXCEPT $![r] = SInSync]$
    $\land\ rLog' = [rLog$ EXCEPT $![r] = m.log \circ SubSeq(rLog[r],\ Len(m.log),\ Len(rLog[r]))]$
    $\land\ rSyncIndex' = [rSyncIndex$ EXCEPT $![r] = Len(rLog'[r])]$
    $\land\ Reply(m,\ [src \mapsto r,$
                    $dest \mapsto m.req.src,$
                    $req \mapsto m.req,$
                    $type \mapsto MReconcileReply,$
                    $viewID \mapsto rViewID[r],$
                    $index \mapsto m.index,$
                    $checksum \mapsto m.log,$
                    $value \mapsto m.log[m.index].value,$
                    $timestamp \mapsto m.log[m.index].timestamp,$
                    $succeeded \mapsto$ TRUE$])$
    $\land$ UNCHANGED $\langle clientVars,\ rViewID,\ rLogViewID,\ rViewChangeReps \rangle$

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

Replica 'r' handles replica 's' view change request 'm'
$HandleViewChangeRequest(r,\ s,\ m)\ \triangleq$
    $\land\ \lor\ \land\ rViewID[r] < m.viewID$
            $\land\ rViewID' = [rViewID$ EXCEPT $![r] = m.viewID]$
            $\land\ rStatus' = [rStatus$ EXCEPT $![r] = SViewChange]$
            $\land\ rViewChangeReps' = [rViewChangeReps$ EXCEPT $![r] = \{\}]$
            $\land\ Reply(m,\ [src \mapsto r,$
                            $dest \mapsto Primary(m.viewID),$
                            $type \mapsto MViewChangeReply,$
                            $viewID \mapsto m.viewID,$
                            $logViewID \mapsto rLogViewID[r],$
                  $log \quad\quad \mapsto rLog[r]])$
      $\lor \land rViewID[r] \geq m.viewID$
         $\land Ack(m)$
         $\land$ UNCHANGED $\langle rViewID, rStatus, rViewChangeReps \rangle$
   $\land$ UNCHANGED $\langle clientVars, rLog, rLogViewID, rSyncIndex \rangle$

Replica 'r' handles replica 's' view change reply 'm'
$HandleViewChangeReply(r, s, m) \triangleq$
   The view change protocol is run by the primary for the view.
   $\land Primary(m.viewID) = r$
   $\land rViewID[r] = m.viewID$
   $\land rStatus[r] \quad = SViewChange$
   $\land rViewChangeReps' = [rViewChangeReps$ EXCEPT $![r] = rViewChangeReps[r] \cup \{m\}]$
   $\land$ LET $viewChanges \triangleq \{v \in rViewChangeReps'[r] : v.viewID = rViewID[r]\}$
     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.
         $\lor \land IsLocalQuorum(r, \{v.src : v \in viewChanges\})$
            $\land$ LET $latestViewID \triangleq Max(\{v.logViewID : v \in viewChanges\})$
                    $latestChange \triangleq$ CHOOSE $v \in viewChanges :$
                                               $\land \quad v.logViewID = latestViewID$
                                               $\land \quad v.src \in Quorum(latestViewID)$
               IN $\quad AckAndSends(m, \{[src \quad \mapsto r,$
                                          $dest \quad \mapsto d,$
                                          $type \quad \mapsto MStartViewRequest,$
                                          $viewID \mapsto rViewID[r],$
                                          $log \quad \mapsto latestChange.log] : d \in Replicas\})$
         If view change replies have not yet been received from a quorum, record
         the view change reply and discard the message.
         $\lor \land \neg IsLocalQuorum(r, \{v.src : v \in viewChanges\})$
            $\land Ack(m)$
   $\land$ UNCHANGED $\langle clientVars, rStatus, rViewID, rLog, rLogViewID, rSyncIndex \rangle$

Replica 'r' handles replica 's' start view request 'm'
$HandleStartViewRequest(r, s, m) \triangleq$
   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.
   $\land \lor rViewID[r] < m.viewID$
      $\lor \land rViewID[r] = m.viewID$
         $\land rStatus[r] \quad = SViewChange$
   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.
```

```
$\land \lor \land r \in Quorum(m.viewID)$
        $\land rLog' = [rLog \text{ EXCEPT } ![r] = m.log]$
        $\land rLogViewID' = [rLogViewID \text{ EXCEPT } ![r] = m.viewID]$
        $\land rSyncIndex' = [rSyncIndex \text{ EXCEPT } ![r] = Len(m.log)]$
    $\lor \land r \notin Quorum(m.viewID)$
        $\land \text{UNCHANGED } \langle rLog, rLogViewID, rSyncIndex \rangle$
Update the replica's view ID and status and clean up view change state.
$\land rViewID' = [rViewID \text{ EXCEPT } ![r] = m.viewID]$
$\land rStatus' = [rStatus \text{ EXCEPT } ![r] = SInSync]$
$\land \text{LET } viewChanges \triangleq \{v \in rViewChangeReps[r] : v.viewID = rViewID[r]\}$
  $\text{IN } rViewChangeReps' = [rViewChangeReps \text{ EXCEPT } ![r] = rViewChangeReps[r] \setminus viewChanges]$
$\land Ack(m)$
$\land \text{UNCHANGED } \langle clientVars \rangle$
```

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

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

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

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

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.

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

```
$NextClientRequest \triangleq$
    $\exists\, c \in Clients :$
      $\exists\, v \in Values :$
        $ClientRequest(c,\, v)$

$NextChangeView \triangleq$
    $\exists\, r \in Replicas :$
      $ChangeView(r)$

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

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

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

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

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

$NextHandleRepairReply \triangleq$
    $\exists\, m \in messages :$
      $\land\, m.type = MRepairReply$
```

```
        $\land$ $HandleRepairReply(m.dest, m.src, m)$

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

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

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

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

$Next \triangleq$
      $\lor$ $NextClientRequest$
      $\lor$ $NextChangeView$
      $\lor$ $NextHandleClientRequest$
      $\lor$ $NextHandleClientReply$
      $\lor$ $NextHandleReconcileRequest$
      $\lor$ $NextHandleReconcileReply$
      $\lor$ $NextHandleRepairRequest$
      $\lor$ $NextHandleRepairReply$
      $\lor$ $NextHandleViewChangeRequest$
      $\lor$ $NextHandleViewChangeReply$
      $\lor$ $NextHandleStartViewRequest$
      $\lor$ $NextDropMessage$

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

\ * Modification History
\ * Last modified *Thu Oct* 01 11:01:54 *PDT* 2020 by *jordanhalterman*
\ * Created *Fri Sep* 18 22:45:21 *PDT* 2020 by *jordanhalterman*