

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

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

*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*  $\mapsto s \in (1 \dots)$ ],

$timestamp \mapsto t \in (1 \dots)$

*ClientReply*

[ $src \mapsto r \in Replicas,$   
 $dest \mapsto c \in Clients,$   
 $type \mapsto MClientReply,$   
 $viewID \mapsto i \in ViewIDs,$   
 $value \mapsto v \in Values,$   
 $seqNum \mapsto s \in (1 \dots),$   
 $index \mapsto i \in (1 \dots),$   
 $succeeded \mapsto TRUE \vee FALSE]$

*RepairRequest*

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

*RepairReply*

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

*AbortRequest*

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

*AbortReply*

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

*ViewChangeRequest*

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

```

ViewChangeReply
[src      ↦ r ∈ Replicas,
 dest     ↦ r ∈ Replicas,
 type     ↦ MViewChangeReply,
 viewID   ↦ i ∈ ViewIDs,
 lastViewID ↦ i ∈ (1 ..),
 log      ↦ [c ∈ Clients ↦ [ i ∈ 1 .. (1 ..) ↦ [
                      value ↦ v ∈ Values, timestamp ↦ (1 .. )]]]]

StartViewRequest
[src      ↦ r ∈ Replicas,
 dest     ↦ r ∈ Replicas,
 type     ↦ MStartViewRequest,
 viewID   ↦ i ∈ ViewIDs,
 timestamp ↦ t ∈ (1 ..), log      ↦ [c ∈ Clients ↦ [
                      i ∈ 1 .. (1 ..) ↦ [ value ↦ v ∈ Values, timestamp ↦ (1 .. )]]]]

```

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

Selects an element of set 'S'  
 $Pick(S) \triangleq$  CHOOSE  $s \in S : \text{TRUE}$

RECURSIVE  $SetReduce(-, -, -)$   
 $SetReduce(Op(-, -), S, value) \triangleq$   
 IF  $S = \{\}$  THEN

$value$   
 ELSE  
 $LET\ s \triangleq Pick(S)$   
 $IN\ SetReduce(Op, S \setminus \{s\}, Op(s, value))$   
 Computes the greatest vluue 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'  
 $Sum(S) \triangleq LET\ _op(a, b) \triangleq a + b$   
 $IN\ SetReduce(_op, S, 0)$   
 A boolean indicating whether the given set is a quorum  
 $IsQuorum(s) \triangleq Cardinality(s) * 2 \geq Cardinality(Replicas)$   
 The set of all quorums  
 $Quorums \triangleq \{r \in SUBSET\ Replicas : IsQuorum(r)\}$   
 The primary for view 'v'  
 $Primary(v) \triangleq replicas[(v \% Len(replicas)) + (IF\ v \geq Len(replicas)\ THEN\ 1\ ELSE\ 0)]$   
 A boolean indicating whether replica 'r' is the primary for the current view  
 $IsPrimary(r) \triangleq Primary(rViewID[r]) = r$

---

This section models the network.

Send a set of messages  
 $Sends(ms) \triangleq$   
 $\wedge\ messages' = messages \cup ms$   
 $\wedge\ messageCount' = messageCount + Cardinality(ms)$   
 Send a message  
 $Send(m) \triangleq Sends(\{m\})$   
 Reply to a message with a set of responses  
 $Replies(req, reps) \triangleq$   
 $\wedge\ messages' = (messages \cup reps) \setminus \{req\}$   
 $\wedge\ messageCount' = messageCount + Cardinality(reps)$   
 Reply to a message  
 $Reply(req, resp) \triangleq Replies(req, \{resp\})$   
 Discard a message  
 $Discard(m) \triangleq$   
 $\wedge\ messages' = messages \setminus \{m\}$   
 $\wedge\ messageCount' = messageCount + 1$

---

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 it 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.

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

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.

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

$$\begin{aligned}
& \wedge cReps' = [cReps \text{ EXCEPT } ![c] = \{\}] \\
& \wedge \text{UNCHANGED } \langle cCommits \rangle \\
& \vee \wedge m.viewID < cViewID[c] \\
& \wedge \text{UNCHANGED } \langle cViewID, cSeqNum, cReps, cCommits \rangle \\
& \wedge Discard(m) \\
& \wedge \text{UNCHANGED } \langle globalVars, replicaVars, cTime \rangle
\end{aligned}$$

---

This section models the replica protocol.

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

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

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

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

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



$$\begin{aligned}
& \wedge \text{UNCHANGED } \langle rStatus, rAbortPoint, rAbortReps \rangle \\
& \wedge \text{UNCHANGED } \langle rLog, rSeqNum, rTimestamp \rangle \\
\vee & \wedge m.viewID < rViewID[r] \\
& \wedge \text{Reply}(m, [src \mapsto r, \\
& \quad dest \mapsto c, \\
& \quad type \mapsto MClientReply, \\
& \quad viewID \mapsto rViewID[r], \\
& \quad seqNum \mapsto m.seqNum, \\
& \quad succeeded \mapsto \text{FALSE}]) \\
& \wedge \text{UNCHANGED } \langle rStatus, rLog, rSeqNum, rTimestamp, rAbortPoint, rAbortReps \rangle \\
& \wedge \text{UNCHANGED } \langle globalVars, clientVars, rViewID, rLastViewID, rViewChanges \rangle
\end{aligned}$$

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.

$$\begin{aligned}
\text{HandleRepairRequest}(r, s, m) & \triangleq \\
& \wedge m.viewID = rViewID[r] \\
& \wedge \text{IsPrimary}(r) \\
& \wedge rStatus[r] = SNormal \\
& \wedge \text{LET } offset \triangleq \text{Len}(rLog[r][m.client]) - (rSeqNum[r][m.client] - m.seqNum) \\
& \text{IN} \\
& \vee \wedge offset \leq \text{Len}(rLog[r][m.client]) \\
& \wedge \text{Reply}(m, [src \mapsto r, \\
& \quad dest \mapsto s, \\
& \quad type \mapsto MRepairReply, \\
& \quad viewID \mapsto rViewID[r], \\
& \quad client \mapsto m.client, \\
& \quad seqNum \mapsto m.seqNum, \\
& \quad value \mapsto rLog[r][m.client][offset].value, \\
& \quad timestamp \mapsto rLog[r][m.client][offset].timestamp]) \\
& \wedge \text{UNCHANGED } \langle rStatus, rAbortPoint, rAbortReps \rangle \\
& \vee \wedge offset = \text{Len}(rLog[r][m.client]) + 1 \\
& \wedge \text{Abort}(r, m.client, m) \\
& \wedge \text{UNCHANGED } \langle globalVars, clientVars, rLog, rSeqNum, rTimestamp, rViewID, rLastViewID, rViewChanges \rangle
\end{aligned}$$

Replica 'r' handles replica 's' repair response 'm'

Repair responses are handled like client requests.

$$\begin{aligned}
\text{HandleRepairReply}(r, s, m) & \triangleq \\
& \text{HandleClientRequest}(r, m.client, [m \text{ EXCEPT } !.src = m.client])
\end{aligned}$$

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 appended to the log, it is.

$$\begin{aligned}
\text{HandleAbortRequest}(r, s, m) & \triangleq \\
& \wedge m.viewID = rViewID[r] \\
& \wedge rStatus[r] \in \{SNormal, SAborting\}
\end{aligned}$$

$\wedge$  LET  
 $\text{offset} \triangleq \text{Len}(\text{rLog}[r][m.\text{client}]) - (\text{rSeqNum}[r][m.\text{client}] - m.\text{seqNum})$   
 $\text{entry} \triangleq [\text{type} \mapsto \text{TNoOp}, \text{value} \mapsto \text{Nil}, \text{timestamp} \mapsto 0]$   
 IN  
 $\wedge \vee \wedge \text{offset} \leq \text{Len}(\text{rLog}[r][m.\text{client}])$   
 $\wedge \text{rLog}' = [\text{rLog} \text{ EXCEPT } ![r] = [$   
 $\quad \text{rLog}[r] \text{ EXCEPT } ![m.\text{client}] = [$   
 $\quad \text{rLog}[r][m.\text{client}] \text{ EXCEPT } ![\text{offset}] = [$   
 $\quad \text{rLog}[r][m.\text{client}][\text{offset}] \text{ EXCEPT } !.\text{type} = \text{TNoOp}]]]$   
 $\wedge \text{UNCHANGED } \langle \text{rTimestamp}, \text{rSeqNum} \rangle$   
 $\vee \wedge \text{offset} = \text{Len}(\text{rLog}[r][m.\text{client}]) + 1$   
 $\wedge \text{rLog}' = [\text{rLog} \text{ EXCEPT } ![r] = [$   
 $\quad \text{rLog}[r] \text{ EXCEPT } ![m.\text{client}] =$   
 $\quad \text{Append}(\text{rLog}[r][m.\text{client}], \text{entry})]$   
 $\wedge \text{rTimestamp}' = [\text{rTimestamp} \text{ EXCEPT } ![r] = \text{Max}(\{\text{rTimestamp}[r], m.\text{timestamp}\})]$   
 $\wedge \text{rSeqNum}' = [\text{rSeqNum} \text{ EXCEPT } ![r] = [$   
 $\quad \text{rSeqNum}[r] \text{ EXCEPT } ![m.\text{client}] = m.\text{seqNum}]$   
 $\wedge \text{Replies}(m, \{[\text{src} \mapsto r,$   
 $\quad \text{dest} \mapsto \text{Primary}(\text{rViewID}[r]),$   
 $\quad \text{type} \mapsto \text{MAbortReply},$   
 $\quad \text{viewID} \mapsto \text{rViewID}[r],$   
 $\quad \text{client} \mapsto m.\text{client},$   
 $\quad \text{seqNum} \mapsto m.\text{seqNum}],$   
 $\quad [\text{src} \mapsto r,$   
 $\quad \text{dest} \mapsto m.\text{client},$   
 $\quad \text{type} \mapsto \text{MClientReply},$   
 $\quad \text{viewID} \mapsto \text{rViewID}[r],$   
 $\quad \text{seqNum} \mapsto m.\text{seqNum},$   
 $\quad \text{succeeded} \mapsto \text{FALSE}]\})$   
 $\wedge \text{UNCHANGED } \langle \text{globalVars}, \text{clientVars}, \text{rStatus}, \text{rAbortPoint},$   
 $\quad \text{rAbortReps}, \text{rViewID}, \text{rLastViewID}, \text{rViewChanges} \rangle$

Replica 'r' handles replica 's' repair response 'm'  
 $\text{HandleAbortReply}(r, s, m) \triangleq$   
 $\wedge \text{rStatus}[r] = \text{SAborting}$   
 $\wedge m.\text{viewID} = \text{rViewID}[r]$   
 $\wedge \text{IsPrimary}(r)$   
 $\wedge m.\text{seqNum} = \text{rAbortPoint}[r].\text{seqNum}$   
 $\wedge \text{rAbortReps}' = [\text{rAbortReps} \text{ EXCEPT } ![r] = \text{rAbortReps}[r] \cup \{m\}]$   
 $\wedge \text{LET } \text{reps} \triangleq \{ \text{res}.\text{src} : \text{res} \in \{ \text{resp} \in \text{rAbortReps}'[r] :$   
 $\quad \wedge \text{resp}.\text{viewID} = \text{rViewID}[r]$   
 $\quad \wedge \text{resp}.\text{client} = \text{rAbortPoint}[r].\text{client}$   
 $\quad \wedge \text{resp}.\text{seqNum} = \text{rAbortPoint}[r].\text{seqNum} \} \}$   
 $\quad \text{isQuorum} \triangleq r \in \text{reps} \wedge \text{reps} \in \text{Quorums}$   
 IN

$$\begin{aligned}
& \vee \wedge isQuorum \\
& \wedge rStatus' = [rStatus \text{ EXCEPT } ![r] = SNormal] \\
& \vee \wedge \neg isQuorum \\
& \wedge UNCHANGED \langle rStatus \rangle \\
& \wedge UNCHANGED \langle globalVars, messageVars, clientVars, rLog, rSeqNum, rTimestamp, \\
& \quad rAbortPoint, rViewID, rViewChanges, rLastViewID \rangle
\end{aligned}$$

Replica 'r' requests a view change

The view change is requested by sending a *ViewChangeRequest* to each replica.

$$\begin{aligned}
ChangeView(r) & \triangleq \\
& \wedge Sends(\{ \begin{array}{ll} src & \mapsto r, \\ dest & \mapsto d, \\ type & \mapsto MViewChangeRequest, \\ viewID & \mapsto rViewID[r] + 1 : d \in Replicas \end{array} \}) \\
& \wedge UNCHANGED \langle globalVars, clientVars, replicaVars \rangle
\end{aligned}$$

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.

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

Replica 'r' handles replica 's' view change response 'm'

*ViewChangeReplies* 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.

$$\begin{aligned}
HandleViewChangeReply(r, s, m) & \triangleq \\
& \wedge IsPrimary(r) \\
& \wedge rViewID[r] = m.viewID
\end{aligned}$$



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


---


$$\begin{aligned}
InitMessageVars &\triangleq \\
&\wedge messages = \{\} \\
&\wedge messageCount = 0
\end{aligned}$$

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

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

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


---

This section specifies the invariants for the protocol.

Merges the set of logs 'L' into a single ordered *log*

RECURSIVE *MergeLogs*(*L*)

*MergeLogs*(*L*)  $\triangleq$

IF  $\neg \exists l \in L : Len(l) > 0$  THEN

$\langle \rangle$

ELSE

LET *nextLog*  $\triangleq$  CHOOSE *l1*  $\in L$  :

$$\begin{aligned}
& \wedge \text{Len}(l1) > 0 \\
& \wedge \forall l2 \in L : \\
& \quad \vee \text{Len}(l2) = 0 \\
& \quad \vee l1[1].\text{timestamp} \leq l2[1].\text{timestamp} \\
& \text{nextEntry} \triangleq \text{nextLog}[1] \\
& \text{newLogs} \triangleq \{ \text{IF } \text{Len}(l) > 0 \wedge l[1].\text{timestamp} = \text{nextEntry}.\text{timestamp} \text{ THEN} \\
& \quad [i \in 1 \dots \text{Len}(l) - 1 \mapsto l[i + 1]] \\
& \quad \text{ELSE } l : l \in L \} \\
& \text{IN} \\
& \langle \text{nextEntry} \rangle \circ \text{MergeLogs}(\text{newLogs})
\end{aligned}$$

The commit order invariant asserts that no two commits with the same index

$$\begin{aligned}
\text{CommitOrderInv} & \triangleq \\
& \forall c1, c2 \in \text{Clients} : \\
& \quad \forall e1 \in c\text{Commits}[c1] : \\
& \quad \neg \exists e2 \in c\text{Commits}[c2] : \\
& \quad \quad \wedge e1.\text{index} = e2.\text{index} \\
& \quad \quad \wedge e1.\text{value} \neq e2.\text{value}
\end{aligned}$$

The type invariant asserts that the leader's *log* will never contain a different value at the same index as a client commit.

$$\begin{aligned}
\text{TypeOK} & \triangleq \\
& \forall c \in \text{Clients} : \\
& \quad \forall e \in c\text{Commits}[c] : \\
& \quad \neg \exists r \in \text{Replicas} : \\
& \quad \quad \wedge \text{Primary}(r\text{ViewID}[r]) = r \\
& \quad \quad \wedge r\text{Status}[r] = \text{SNormal} \\
& \quad \quad \wedge \text{LET } \text{logs} \triangleq \text{MergeLogs}(\{r\text{Log}[r][i] : i \in \text{DOMAIN } r\text{Log}[r]\}) \\
& \quad \quad \text{IN} \\
& \quad \quad \quad \wedge \text{Len}(\text{logs}) \geq e.\text{index} \\
& \quad \quad \quad \wedge \text{logs}[e.\text{index}].\text{value} \neq e.\text{value}
\end{aligned}$$


---


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

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

$$\begin{aligned}
\text{NextHandleClientRequest} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MClientRequest}
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{HandleClientRequest}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleClientReply} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MClientReply} \\
& \quad \wedge \text{HandleClientReply}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleRepairRequest} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MRepairRequest} \\
& \quad \wedge \text{HandleRepairRequest}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleRepairReply} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MRepairReply} \\
& \quad \wedge \text{HandleRepairReply}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleAbortRequest} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MAbortRequest} \\
& \quad \wedge \text{HandleAbortRequest}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleAbortReply} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MAbortReply} \\
& \quad \wedge \text{HandleAbortReply}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleViewChangeRequest} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MViewChangeRequest} \\
& \quad \wedge \text{HandleViewChangeRequest}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleViewChangeReply} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MViewChangeReply} \\
& \quad \wedge \text{HandleViewChangeReply}(m.\text{dest}, m.\text{src}, m) \\
\text{NextHandleStartViewRequest} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge m.\text{type} = \text{MStartViewRequest} \\
& \quad \wedge \text{HandleStartViewRequest}(m.\text{dest}, m.\text{src}, m) \\
\text{NextDropMessage} & \triangleq \\
& \exists m \in \text{messages} : \\
& \quad \wedge \text{Discard}(m) \\
& \quad \wedge \text{UNCHANGED } \langle \text{globalVars}, \text{clientVars}, \text{replicaVars} \rangle \\
\text{Next} & \triangleq
\end{aligned}$$

$\vee \text{NextClientRequest}$   
 $\vee \text{NextChangeView}$   
 $\vee \text{NextHandleClientRequest}$   
 $\vee \text{NextHandleClientReply}$   
 $\vee \text{NextHandleRepairRequest}$   
 $\vee \text{NextHandleRepairReply}$   
 $\vee \text{NextHandleAbortRequest}$   
 $\vee \text{NextHandleAbortReply}$   
 $\vee \text{NextHandleViewChangeRequest}$   
 $\vee \text{NextHandleViewChangeReply}$   
 $\vee \text{NextHandleStartViewRequest}$   
 $\vee \text{NextDropMessage}$

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

---

\ \* Modification History  
\ \* Last modified *Thu Sep 24 14:24:29 PDT 2020* by *jordanhalterman*  
\ \* Created *Fri Sep 18 22:45:21 PDT 2020* by *jordanhalterman*