



The leader's epoch or the last new epoch proposal the follower acknowledged  
(namely epoch of the last *NEWEPOCH* accepted,  $f.p$  in paper).

VARIABLE *currentEpoch*

The last new leader proposal the follower acknowledged  
(namely epoch of the last *NEWLEADER* accepted,  $f.a$  in paper).

VARIABLE *leaderEpoch*

The identifier of the leader for followers.

VARIABLE *leaderOracle*

The history of servers as the sequence of transactions.

VARIABLE *history*

The messages representing requests and responses sent from one server to another.  
 $msgs[i][j]$  means the input buffer of server  $j$  from server  $i$ .

VARIABLE *msgs*

The set of servers which the leader think follow itself ( $Q$  in paper).

VARIABLE *cluster*

The set of followers who has successfully sent *CEPOCH* to pleader in pleader.

VARIABLE *ceepochRecv*

The set of followers who has successfully sent *ACK-E* to pleader in pleader.

VARIABLE *ackeRecv*

The set of followers who has successfully sent *ACK-LD* to pleader in pleader.

VARIABLE *ackldRecv*

$ackIndex[i][j]$  means leader  $i$  has received how many *ACK* messages from follower  $j$ .  
So  $ackIndex[i][i]$  is not used.

VARIABLE *ackIndex*

$currentCounter[i]$  means the count of transactions client requests leader.

VARIABLE *currentCounter*

$sendCounter[i]$  means the count of transactions leader has broadcast.

VARIABLE *sendCounter*

$initialHistory[i]$  means the initial history of leader  $i$  in epoch  $currentEpoch[i]$ .

VARIABLE *initialHistory*

$commitIndex[i]$  means leader/follower  $i$  should commit how many proposals and sent *COMMIT* messages.

It should be more formal to add variable *applyIndex*/*deliverIndex* to represent the prefix entries of the history that has applied to state machine, but we can tolerate that  $applyIndex(deliverIndex \text{ here}) = commitIndex$ .

This does not violate correctness. (*commitIndex* increases monotonically before restarting)

VARIABLE *commitIndex*

$commitIndex[i]$  means leader  $i$  has committed how many proposals and sent *COMMIT* messages.  
 VARIABLE *committedIndex*

Helper matrix for follower to stop sending *CEPOCH* to pleader in followers.  
 Because *CEPOCH* is the sole message which follower actively sends to pleader.  
 VARIABLE *ceepochSent*

the maximum epoch in *CEPOCH* pleader received from followers.  
 VARIABLE *tempMaxEpoch*

the maximum *leaderEpoch* and most up-to-date history in *ACKE* pleader received from followers.  
 VARIABLE *tempMaxLastEpoch*

Because pleader updates state and broadcasts *NEWLEADER* when it receives *ACKE* from a quorum of followers,  
 and *initialHistory* is determined. But *tempInitialHistory* may change when receiving other *ACKEs* after entering into *phase2*.  
 So it is necessary to split *initialHistory* with *tempInitialHistory*.  
 VARIABLE *tempInitialHistory*

the set of all broadcast messages whose type is proposal that any leader has sent, only used in verifying properties.  
 So the variable will only be changed in transition *LeaderBroadcast1*.  
 VARIABLE *proposalMsgsLog*

Helper set for server who restarts to collect which servers has responded to it.  
 VARIABLE *recoveryRespRecv*

the maximum epoch and corresponding *leaderOracle* in *RECOVERYRESPONSE* from followers.  
 VARIABLE *recoveryMaxEpoch*

VARIABLE *recoveryMEOracle*

VARIABLE *recoverySent*

Persistent state of a server: history, *currentEpoch*, *leaderEpoch*  
 $serverVars \triangleq \langle state, currentEpoch, leaderEpoch, leaderOracle, history, commitIndex \rangle$   
 $leaderVars \triangleq \langle cluster, cepochRecv, ackRecv, ackldRecv, ackIndex, currentCounter, sendCounter, initialHistory \rangle$   
 $tempVars \triangleq \langle tempMaxEpoch, tempMaxLastEpoch, tempInitialHistory \rangle$   
 $recoveryVars \triangleq \langle recoveryRespRecv, recoveryMaxEpoch, recoveryMEOracle, recoverySent \rangle$

$vars \triangleq \langle serverVars, msgs, leaderVars, tempVars, recoveryVars, cepochSent, proposalMsgsLog \rangle$

---

$LastZxid(his) \triangleq \text{IF } Len(his) > 0 \text{ THEN } \langle his[Len(his)].epoch, his[Len(his)].counter \rangle$   
 ELSE  $\langle -1, -1 \rangle$

Add a message to *msgs* – add a message *m* to *msgs*[*i*][*j*]  
 $Send(i, j, m) \triangleq msgs' = [msgs \text{ EXCEPT } ![i][j] = Append(msgs[i][j], m)]$

$Send2(i, j, m1, m2) \triangleq msgs' = [msgs \text{ EXCEPT } ![i][j] = Append(Append(msgs[i][j], m1), m2)]$

Remove a message from *msgs* – discard head of *msgs*[*i*][*j*]

$Discard(i, j) \triangleq msgs' = \text{IF } msgs[i][j] \neq \langle \rangle \text{ THEN } [msgs \text{ EXCEPT } ![i][j] = Tail(msgs[i][j])] \\ \text{ELSE } msgs$

Leader/Plader broadcasts a message to all other servers in  $Q$

$Broadcast(i, m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } \wedge ii = i \\ \wedge ij \neq i \\ \wedge ij \in cluster[i] \text{ THEN } Append(msgs[ii][ij], m) \\ \text{ELSE } msgs[ii][ij]]]$

$BroadcastToAll(i, m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } \wedge ii = i \wedge ij \neq i \text{ THEN } Append(msgs[ii][ij], m) \\ \text{ELSE } msgs[ii][ij]]]$

Combination of *Send* and *Discard* – discard head of  $msgs[j][i]$  and add  $m$  into  $msgs[i][j]$

$Reply(i, j, m) \triangleq msgs' = [msgs \text{ EXCEPT } ![j][i] = Tail(msgs[j][i]), \\ ![i][j] = Append(msgs[i][j], m)]$

$Reply2(i, j, m1, m2) \triangleq msgs' = [msgs \text{ EXCEPT } ![j][i] = Tail(msgs[j][i]), \\ ![i][j] = Append(Append(msgs[i][j], m1), m2)]$

$clean(i, j) \triangleq msgs' = [msgs \text{ EXCEPT } ![i][j] = \langle \rangle, ![j][i] = \langle \rangle]$

---

Define initial values for all variables

$Init \triangleq \wedge state = [s \in Server \mapsto Follower] \\ \wedge currentEpoch = [s \in Server \mapsto 0] \\ \wedge leaderEpoch = [s \in Server \mapsto 0] \\ \wedge leaderOracle = [s \in Server \mapsto NullPoint] \\ \wedge history = [s \in Server \mapsto \langle \rangle] \\ \wedge msgs = [i \in Server \mapsto [j \in Server \mapsto \langle \rangle]] \\ \wedge cluster = [i \in Server \mapsto \{\}] \\ \wedge cepochRecv = [s \in Server \mapsto \{\}] \\ \wedge ackRecv = [s \in Server \mapsto \{\}] \\ \wedge ackldRecv = [s \in Server \mapsto \{\}] \\ \wedge ackIndex = [i \in Server \mapsto [j \in Server \mapsto 0]] \\ \wedge currentCounter = [s \in Server \mapsto 0] \\ \wedge sendCounter = [s \in Server \mapsto 0] \\ \wedge commitIndex = [s \in Server \mapsto 0] \\ \wedge committedIndex = [s \in Server \mapsto 0] \\ \wedge initialHistory = [s \in Server \mapsto \langle \rangle] \\ \wedge cepochSent = [s \in Server \mapsto FALSE] \\ \wedge tempMaxEpoch = [s \in Server \mapsto 0] \\ \wedge tempMaxLastEpoch = [s \in Server \mapsto 0] \\ \wedge tempInitialHistory = [s \in Server \mapsto \langle \rangle] \\ \wedge recoveryRespRecv = [s \in Server \mapsto \{\}] \\ \wedge recoveryMaxEpoch = [s \in Server \mapsto 0] \\ \wedge recoveryMEOracle = [s \in Server \mapsto NullPoint] \\ \wedge recoverySent = [s \in Server \mapsto FALSE]$

$$\wedge \text{proposalMsgsLog} = \{\}$$

A server becomes pleader and a quorum servers knows that.

$$\text{Election}(i, Q) \triangleq$$

test restrictions

$$\wedge \forall s \in \text{Server} : \text{currentEpoch}[s] \leq 1 \wedge \text{Len}(\text{history}[s]) \leq 2$$

$$\wedge i \in Q$$

$$\wedge \text{state}' = [s \in \text{Server} \mapsto \text{IF } s = i \text{ THEN } \text{ProspectiveLeader} \\ \text{ELSE IF } s \in Q \text{ THEN } \text{Follower} \\ \text{ELSE } \text{state}[s]]$$

$$\wedge \text{cluster}' = [\text{cluster} \text{ EXCEPT } ![i] = Q] \text{ cluster is first initialized in election, not phase1.}$$

$$\wedge \text{cepochRecv}' = [\text{cepochRecv} \text{ EXCEPT } ![i] = \{i\}]$$

$$\wedge \text{ackRecv}' = [\text{ackRecv} \text{ EXCEPT } ![i] = \{i\}]$$

$$\wedge \text{ackldRecv}' = [\text{ackldRecv} \text{ EXCEPT } ![i] = \{i\}]$$

$$\wedge \text{ackIndex}' = [ii \in \text{Server} \mapsto [ij \in \text{Server} \mapsto \\ \text{IF } ii = i \text{ THEN } 0$$

$$\text{ELSE } \text{ackIndex}[ii][ij]]]$$

$$\wedge \text{committedIndex}' = [\text{committedIndex} \text{ EXCEPT } ![i] = 0]$$

$$\wedge \text{initialHistory}' = [\text{initialHistory} \text{ EXCEPT } ![i] = \langle \rangle]$$

$$\wedge \text{tempMaxEpoch}' = [\text{tempMaxEpoch} \text{ EXCEPT } ![i] = \text{currentEpoch}[i]]$$

$$\wedge \text{tempMaxLastEpoch}' = [\text{tempMaxLastEpoch} \text{ EXCEPT } ![i] = \text{currentEpoch}[i]]$$

$$\wedge \text{tempInitialHistory}' = [\text{tempInitialHistory} \text{ EXCEPT } ![i] = \text{history}[i]]$$

$$\wedge \text{leaderOracle}' = [s \in \text{Server} \mapsto \text{IF } s \in Q \text{ THEN } i \\ \text{ELSE } \text{leaderOracle}[s]]$$

$$\wedge \text{leaderEpoch}' = [s \in \text{Server} \mapsto \text{IF } s \in Q \text{ THEN } \text{currentEpoch}[s] \\ \text{ELSE } \text{leaderEpoch}[s]]$$

$$\wedge \text{cepochSent}' = [s \in \text{Server} \mapsto \text{IF } s \in Q \text{ THEN } \text{FALSE} \\ \text{ELSE } \text{cepochSent}[s]]$$

$$\wedge \text{msgs}' = [ii \in \text{Server} \mapsto [ij \in \text{Server} \mapsto \\ \text{IF } ii \in Q \vee ij \in Q \text{ THEN } \langle \rangle \\ \text{ELSE } \text{msgs}[ii][ij]]]$$

$$\wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{proposalMsgsLog} \rangle$$

The action should be triggered once at the beginning.

Because we abstract the part of leader election, we can use global variables in this action.

$$\text{InitialElection}(i, Q) \triangleq$$

test restrictions

$$\wedge \text{currentEpoch}[i] \leq 2$$

$$\wedge \text{Len}(\text{history}[i]) \leq 2$$

$$\wedge \forall s \in \text{Server} : \text{state}[s] = \text{Follower} \wedge \text{leaderOracle}[s] = \text{NullPoint}$$

$$\wedge \text{Election}(i, Q)$$

$$\wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{recoveryVars}, \text{proposals} \rangle$$

The leader finds timeout with another follower.

$$\text{LeaderTimeout}(i, j) \triangleq$$

$$\begin{aligned}
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] \neq \text{Follower} \\
& \wedge j \neq i \\
& \wedge j \in \text{cluster}[i] \\
& \wedge \text{LET } \text{newCluster} \triangleq \text{cluster}[i] \setminus \{j\} \\
& \quad \text{IN } \wedge \vee \wedge \text{newCluster} \in \text{Quorums} \\
& \quad \quad \wedge \text{cluster}' = [\text{cluster} \text{ EXCEPT } ![i] = \text{newCluster}] \\
& \quad \quad \wedge \text{clean}(i, j) \\
& \quad \quad \wedge \text{UNCHANGED } \langle \text{state}, \text{ceepochRecv}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{committedIndex}, \text{initialHistory}, \\
& \quad \quad \quad \text{tempMaxEpoch}, \text{tempMaxLastEpoch}, \text{tempInitialHistory}, \text{leaderOracle}, \text{leaderOracleIndex} \rangle \\
& \quad \vee \wedge \text{newCluster} \notin \text{Quorums} \\
& \quad \wedge \text{LET } Q \triangleq \text{CHOOSE } q \in \text{Quorums}: i \in q \\
& \quad \quad v \triangleq \text{CHOOSE } s \in Q: \text{TRUE} \\
& \quad \quad \text{IN } \text{Election}(v, Q) \\
& \quad \quad \exists Q \in \text{Quorums} : \wedge i \in Q \\
& \quad \quad \quad \wedge \exists v \in Q : \text{Election}(v, Q) \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{recoveryVars}, \text{proposed} \rangle
\end{aligned}$$

A follower finds timeout with the leader.

$$\begin{aligned}
\text{FollowerTimeout}(i) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Follower} \\
& \wedge \text{leaderOracle}[i] \neq \text{NullPoint} \\
& \wedge \exists Q \in \text{Quorums} : \wedge i \in Q \\
& \quad \wedge \exists v \in Q : \text{Election}(v, Q) \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{recoveryVars}, \text{proposed} \rangle
\end{aligned}$$

---

A server halts and restarts.

Like Recovery protocol in View-stamped Replication, we let a server join in cluster by broadcast recovery and wait until receiving responses from a quorum of servers.

$$\begin{aligned}
\text{Restart}(i) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}' = [\text{state} \text{ EXCEPT } ![i] = \text{Follower}] \\
& \wedge \text{leaderOracle}' = [\text{leaderOracle} \text{ EXCEPT } ![i] = \text{NullPoint}] \\
& \wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = 0] \\
& \wedge \text{ceepochSent}' = [\text{ceepochSent} \text{ EXCEPT } ![i] = \text{FALSE}] \\
& \wedge \text{msgs}' = [ii \in \text{Server} \mapsto [ij \in \text{Server} \mapsto \text{IF } ij = i \text{ THEN } \langle \rangle \\
& \quad \quad \quad \text{ELSE } \text{msgs}[ii][ij]]]
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{recoverySent}' = [\text{recoverySent} \text{ EXCEPT } ![i] = \text{FALSE}] \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{leaderEpoch}, \text{history}, \text{leaderVars}, \text{tempVars}, \\
& \quad \text{recoveryRespRecv}, \text{recoveryMaxEpoch}, \text{recoveryMEOracle}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

$$\text{RecoveryAfterRestart}(i) \triangleq$$

$$\begin{aligned}
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Follower} \\
& \wedge \text{leaderOracle}[i] = \text{NullPoint} \\
& \wedge \neg \text{recoverySent}[i] \\
& \wedge \text{recoveryRespRecv}' = [\text{recoveryRespRecv} \text{ EXCEPT } ![i] = \{\}] \\
& \wedge \text{recoveryMaxEpoch}' = [\text{recoveryMaxEpoch} \text{ EXCEPT } ![i] = \text{currentEpoch}[i]] \\
& \wedge \text{recoveryMEOracle}' = [\text{recoveryMEOracle} \text{ EXCEPT } ![i] = \text{NullPoint}] \\
& \wedge \text{recoverySent}' = [\text{recoverySent} \text{ EXCEPT } ![i] = \text{TRUE}] \\
& \wedge \text{BroadcastToAll}(i, [\text{mtype} \mapsto \text{RECOVERYREQUEST}]) \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{epochSent}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

$$\text{HandleRecoveryRequest}(i, j) \triangleq$$

$$\begin{aligned}
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{RECOVERYREQUEST} \\
& \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{RECOVERYRESPONSE}, \\
& \quad \text{moracle} \mapsto \text{leaderOracle}[i], \\
& \quad \text{mepoch} \mapsto \text{currentEpoch}[i]]) \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{epochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

$$\text{HandleRecoveryResponse}(i, j) \triangleq$$

$$\begin{aligned}
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{RECOVERYRESPONSE} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \quad \text{infoOk} \triangleq \wedge \text{msg.mepoch} \geq \text{recoveryMaxEpoch}[i] \\
& \quad \quad \wedge \text{msg.moracle} \neq \text{NullPoint} \\
& \text{IN } \vee \wedge \text{infoOk} \\
& \quad \wedge \text{recoveryMaxEpoch}' = [\text{recoveryMaxEpoch} \text{ EXCEPT } ![i] = \text{msg.mepoch}] \\
& \quad \wedge \text{recoveryMEOracle}' = [\text{recoveryMEOracle} \text{ EXCEPT } ![i] = \text{msg.moracle}] \\
& \vee \wedge \neg \text{infoOk} \\
& \quad \wedge \text{UNCHANGED } \langle \text{recoveryMaxEpoch}, \text{recoveryMEOracle} \rangle \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{recoveryRespRecv}' = [\text{recoveryRespRecv} \text{ EXCEPT } ![i] = \text{IF } j \in \text{recoveryRespRecv}[i] \text{ THEN } \text{recoveryRe} \\
& \quad \quad \quad \text{ELSE } \text{recoveryRe}
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{cepochSent}, \text{recoverySent}, \text{proposalMsgsLog} \rangle \\
\text{FindCluster}(i) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Follower} \\
& \wedge \text{leaderOracle}[i] = \text{NullPoint} \\
& \wedge \text{recoveryRespRecv}[i] \in \text{Quorums} \\
& \wedge \text{LET } \text{infoOk} \triangleq \wedge \text{recoveryMEOracle}[i] \neq i \\
& \quad \wedge \text{recoveryMEOracle}[i] \neq \text{NullPoint} \\
& \quad \wedge \text{currentEpoch}[i] \leq \text{recoveryMaxEpoch}[i] \\
& \text{IN } \vee \wedge \neg \text{infoOk} \\
& \quad \wedge \text{recoverySent}' = [\text{recoverySent} \text{ EXCEPT } ![i] = \text{FALSE}] \\
& \quad \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{leaderOracle}, \text{msgs} \rangle \\
& \vee \wedge \text{infoOk} \\
& \quad \wedge \text{currentEpoch}' = [\text{currentEpoch} \text{ EXCEPT } ![i] = \text{recoveryMaxEpoch}[i]] \\
& \quad \wedge \text{leaderOracle}' = [\text{leaderOracle} \text{ EXCEPT } ![i] = \text{recoveryMEOracle}[i]] \\
& \quad \wedge \text{Send}(i, \text{recoveryMEOracle}[i], [\text{mtype} \mapsto \text{CEPOCH}, \\
& \quad \quad \quad \text{mepoch} \mapsto \text{recoveryMaxEpoch}[i]]) \\
& \quad \wedge \text{UNCHANGED } \text{recoverySent} \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{leaderEpoch}, \text{history}, \text{commitIndex}, \text{leaderVars}, \text{tempVars}, \\
& \quad \text{recoveryRespRecv}, \text{recoveryMaxEpoch}, \text{recoveryMEOracle}, \text{cepochSent}, \text{proposalMsgsLog} \rangle
\end{aligned}$$


---

In phase *f11*, follower sends *f.p* to pleader via *CEPOCH*.

$$\begin{aligned}
\text{FollowerDiscovery1}(i) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Follower} \\
& \wedge \text{leaderOracle}[i] \neq \text{NullPoint} \\
& \wedge \neg \text{cepochSent}[i] \\
& \wedge \text{LET } \text{leader} \triangleq \text{leaderOracle}[i] \\
& \text{IN } \text{Send}(i, \text{leader}, [\text{mtype} \mapsto \text{CEPOCH}, \\
& \quad \quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i]]) \\
& \wedge \text{cepochSent}' = [\text{cepochSent} \text{ EXCEPT } ![i] = \text{TRUE}] \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

In phase *l11*, pleader receives *CEPOCH* from a quorum, and choose a new epoch *e'* as its own *l.p* and sends *NEWEOPOCH* to followers.

$$\begin{aligned}
\text{LeaderHandleCEPOCH}(i, j) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{tempMaxEpoch}[i] \leq 1 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{ProspectiveLeader}
\end{aligned}$$



$\wedge \text{msgs}[j][i] \neq \langle \rangle$   
 $\wedge \text{msgs}[j][i][1].\text{mtype} = \text{CEPOCH}$   
 $\wedge \vee$  new message - modify  $\text{tempMaxEpoch}$  and  $\text{cepochRecv}$   
 $\wedge \text{NullPoint} \notin \text{cepochRecv}[i]$   
 $\wedge \text{LET } \text{newEpoch} \triangleq \text{Maximum}(\{\text{tempMaxEpoch}[i], \text{msgs}[j][i][1].\text{mepoch}\})$   
 $\quad \text{IN } \text{tempMaxEpoch}' = [\text{tempMaxEpoch} \text{ EXCEPT } ![i] = \text{newEpoch}]$   
 $\wedge \text{cepochRecv}' = [\text{cepochRecv} \text{ EXCEPT } ![i] = \text{IF } j \in \text{cepochRecv}[i] \text{ THEN } \text{cepochRecv}[i]$   
 $\quad \text{ELSE } \text{cepochRecv}[i] \cup \{j\}]$   
 $\wedge \text{Discard}(j, i)$   
 $\vee$  new follower who joins in cluster / follower whose history and  $\text{commitIndex}$  do not match  
 $\wedge \text{NullPoint} \in \text{cepochRecv}[i]$   
 $\wedge \vee \wedge \text{NullPoint} \notin \text{ackRecv}[i]$   
 $\quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{NEWPOCH},$   
 $\quad \quad \text{mepoch} \mapsto \text{leaderEpoch}[i]])$   
 $\vee \wedge \text{NullPoint} \in \text{ackRecv}[i]$   
 $\quad \wedge \text{Reply2}(i, j, [\text{mtype} \mapsto \text{NEWPOCH},$   
 $\quad \quad \text{mepoch} \mapsto \text{leaderEpoch}[i],$   
 $\quad \quad \text{[mtype} \mapsto \text{NEWLEADER},$   
 $\quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i],$   
 $\quad \quad \text{minitialHistory} \mapsto \text{initialHistory}[i]])$   
 $\wedge \text{UNCHANGED } \langle \text{cepochRecv}, \text{tempMaxEpoch} \rangle$   
 $\wedge \text{cluster}' = [\text{cluster} \text{ EXCEPT } ![i] = \text{IF } j \in \text{cluster}[i] \text{ THEN } \text{cluster}[i] \text{ ELSE } \text{cluster}[i] \cup \{j\}]$   
 $\wedge \text{UNCHANGED } \langle \text{serverVars}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \text{sendCounter}, \text{initialHist}$   
 $\quad \text{committedIndex}, \text{cepochSent}, \text{tempMaxLastEpoch}, \text{tempInitialHistory}, \text{recoveryVars}, p$

Here I decide to change leader's epoch in  $l12 \& l21$ , otherwise there may exist an old leader and a new leader who share the same epoch. So here I just change  $\text{leaderEpoch}$ , and use it in handling  $\text{ACK-E}$ .

$\text{LeaderDiscovery1}(i) \triangleq$   
test restrictions  
 $\wedge \text{tempMaxEpoch}[i] \leq 1$   
 $\wedge \text{Len}(\text{history}[i]) \leq 2$   
 $\wedge \text{state}[i] = \text{ProspectiveLeader}$   
 $\wedge \text{cepochRecv}[i] \in \text{Quorums}$   
 $\wedge \text{leaderEpoch}' = [\text{leaderEpoch} \text{ EXCEPT } ![i] = \text{tempMaxEpoch}[i] + 1]$   
 $\wedge \text{cepochRecv}' = [\text{cepochRecv} \text{ EXCEPT } ![i] = \{\text{NullPoint}\}]$   
 $\wedge \text{Broadcast}(i, [\text{mtype} \mapsto \text{NEWPOCH},$   
 $\quad \text{mepoch} \mapsto \text{leaderEpoch}'[i]])$   
 $\wedge \text{UNCHANGED } \langle \text{state}, \text{currentEpoch}, \text{leaderOracle}, \text{history}, \text{cluster}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \text{sendCounter}, \text{initialHistory}, \text{commitIndex}, \text{committedIndex}, \text{cepochSent}, \text{tempVars}, \text{recoveryVars}, p$

In phase  $f12$ , follower receives  $\text{NEWPOCH}$ . If  $e' > f.p$  then sends back  $\text{ACKE}$ , and  $\text{ACKE}$  contains  $f.a$  and  $hf$  to help pleader choose a newer history.

$\text{FollowerDiscovery2}(i, j) \triangleq$   
test restrictions  
 $\wedge \text{currentEpoch}[i] \leq 2$

$$\begin{aligned}
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Follower} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{NEWPOCH} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \text{IN } \vee \text{new NEWPOCH} - \text{accept and reply} \\
& \quad \wedge \text{currentEpoch}[i] < \text{msg.mepoch} \\
& \quad \wedge \vee \wedge \text{leaderOracle}[i] = j \\
& \quad \quad \wedge \text{currentEpoch}' = [\text{currentEpoch} \text{ EXCEPT } ![i] = \text{msg.mepoch}] \\
& \quad \quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{ACKE}, \\
& \quad \quad \quad \text{mepoch} \mapsto \text{msg.mepoch}, \\
& \quad \quad \quad \text{mlastEpoch} \mapsto \text{leaderEpoch}[i], \\
& \quad \quad \quad \text{mhf} \mapsto \text{history}[i]]) \\
& \quad \vee \wedge \text{leaderOracle}[i] \neq j \\
& \quad \quad \wedge \text{Discard}(j, i) \\
& \quad \quad \wedge \text{UNCHANGED currentEpoch} \\
& \vee \wedge \text{currentEpoch}[i] = \text{msg.mepoch} \\
& \quad \wedge \vee \wedge \text{leaderOracle}[i] = j \\
& \quad \quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{ACKE}, \\
& \quad \quad \quad \text{mepoch} \mapsto \text{msg.mepoch}, \\
& \quad \quad \quad \text{mlastEpoch} \mapsto \text{leaderEpoch}[i], \\
& \quad \quad \quad \text{mhf} \mapsto \text{history}[i]]) \\
& \quad \quad \wedge \text{UNCHANGED currentEpoch} \\
& \vee \text{It may happen when a leader do not update new epoch to all followers in } Q, \text{ and a new election begins} \\
& \quad \wedge \text{leaderOracle}[i] \neq j \\
& \quad \wedge \text{Discard}(j, i) \\
& \quad \wedge \text{UNCHANGED currentEpoch} \\
& \vee \text{stale NEWPOCH} - \text{discard} \\
& \quad \wedge \text{currentEpoch}[i] > \text{msg.mepoch} \\
& \quad \wedge \text{Discard}(j, i) \\
& \quad \wedge \text{UNCHANGED currentEpoch} \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \text{leaderVars}, \\
& \quad \text{commitIndex}, \text{cepochSent}, \text{tempVars}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

In phase *l12*, pleader receives *ACKE* from a quorum,

and select the history of one most up-to-date follower to be the initial history.

*LeaderHandleACKE*(*i*, *j*)  $\triangleq$

test restrictions

$$\begin{aligned}
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACKE} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \quad \text{infoOk} \triangleq \vee \text{msg.mlastEpoch} > \text{tempMaxLastEpoch}[i]
\end{aligned}$$

$$\begin{aligned}
& \vee \wedge \text{msg.mlastEpoch} = \text{tempMaxLastEpoch}[i] \\
& \wedge \vee \text{LastZxid}(\text{msg.mhf})[1] > \text{LastZxid}(\text{tempInitialHistory}[i])[1] \\
& \vee \wedge \text{LastZxid}(\text{msg.mhf})[1] = \text{LastZxid}(\text{tempInitialHistory}[i])[1] \\
& \wedge \text{LastZxid}(\text{msg.mhf})[2] \geq \text{LastZxid}(\text{tempInitialHistory}[i])[2] \\
\text{IN } & \vee \wedge \text{leaderEpoch}[i] = \text{msg.mepoch} \\
& \wedge \vee \wedge \text{infoOk} \\
& \wedge \text{tempMaxLastEpoch}' = [\text{tempMaxLastEpoch} \text{ EXCEPT } ![i] = \text{msg.mlastEpoch}] \\
& \wedge \text{tempInitialHistory}' = [\text{tempInitialHistory} \text{ EXCEPT } ![i] = \text{msg.mhf}] \\
& \vee \wedge \neg \text{infoOk} \\
& \wedge \text{UNCHANGED } \langle \text{tempMaxLastEpoch}, \text{tempInitialHistory} \rangle \\
& \text{Followers not in } Q \text{ will not receive } \text{NEW EPOCH}, \text{ so leader will receive } \text{ACKE} \text{ only when the source is in } Q \\
& \wedge \text{ackRecv}' = [\text{ackRecv} \text{ EXCEPT } ![i] = \text{IF } j \notin \text{ackRecv}[i] \text{ THEN } \text{ackRecv}[i] \cup \{j\} \\
& \hspace{15em} \text{ELSE } \text{ackRecv}[i]] \\
& \vee \wedge \text{leaderEpoch}[i] \neq \text{msg.mepoch} \\
& \wedge \text{UNCHANGED } \langle \text{tempMaxLastEpoch}, \text{tempInitialHistory}, \text{ackRecv} \rangle \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{cluster}, \text{cepochnRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \\
& \hspace{10em} \text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{cepochnSent}, \text{tempMaxEpoch}, \text{recoveryVars} \rangle \\
\text{LeaderDiscovery2Sync1}(i) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \wedge \text{ackRecv}[i] \in \text{Quorums} \\
& \wedge \text{currentEpoch}' = [\text{currentEpoch} \text{ EXCEPT } ![i] = \text{leaderEpoch}[i]] \\
& \wedge \text{history}' = [\text{history} \text{ EXCEPT } ![i] = \text{tempInitialHistory}[i]] \\
& \wedge \text{initialHistory}' = [\text{initialHistory} \text{ EXCEPT } ![i] = \text{tempInitialHistory}[i]] \\
& \wedge \text{ackRecv}' = [\text{ackRecv} \text{ EXCEPT } ![i] = \{\text{NullPoint}\}] \\
& \wedge \text{ackIndex}' = [\text{ackIndex} \text{ EXCEPT } ![i][i] = \text{Len}(\text{tempInitialHistory}[i])] \\
& \text{until now, phase1(Discovery) ends} \\
& \wedge \text{Broadcast}(i, [\text{mtype} \mapsto \text{NEWLEADER}, \\
& \hspace{10em} \text{mepoch} \mapsto \text{currentEpoch}'[i], \\
& \hspace{10em} \text{minitialHistory} \mapsto \text{history}'[i]]) \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{leaderEpoch}, \text{leaderOracle}, \text{commitIndex}, \text{cluster}, \text{cepochnRecv}, \text{ackldRecv}, \\
& \hspace{10em} \text{currentCounter}, \text{sendCounter}, \text{committedIndex}, \text{cepochnSent}, \text{tempVars}, \text{recoveryVars} \rangle \\
\end{aligned}$$

*Note1:* Delete the change of *commitIndex* in *LeaderDiscovery2Sync1* and *FollowerSync1*, then we can promise that *commitIndex* of every server increases monotonically, except that some server halts and restarts.

*Note2:* Set *cepochnRecv*, *ackRecv*, *ackldRecv* to *{NullPoint}* in corresponding three actions to make sure that the prospective leader will not broadcast *NEW EPOCH* / *NEWLEADER* / *COMMITLD* twice.

---

In phase *f21*, follower receives *NEWLEADER*. The follower updates its epoch and history, and sends back *ACK-LD* to pleader.

*FollowerSync1*(*i*, *j*)  $\triangleq$

test restrictions  
 $\wedge \text{currentEpoch}[i] \leq 2$   
 $\wedge \text{Len}(\text{history}[i]) \leq 2$   
 $\wedge \text{state}[i] = \text{Follower}$   
 $\wedge \text{msgs}[j][i] \neq \langle \rangle$   
 $\wedge \text{msgs}[j][i][1].\text{mtype} = \text{NEWLEADER}$   
 $\wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1]$   
 $\text{replyOk} \triangleq \wedge \text{currentEpoch}[i] \leq \text{msg.mepoch}$   
 $\wedge \text{leaderOracle}[i] = j$   
IN  $\vee$  new **NEWLEADER** – accept and reply  
 $\wedge \text{replyOk}$   
 $\wedge \text{currentEpoch}' = [\text{currentEpoch} \text{ EXCEPT } ![i] = \text{msg.mepoch}]$   
 $\wedge \text{leaderEpoch}' = [\text{leaderEpoch} \text{ EXCEPT } ![i] = \text{msg.mepoch}]$   
 $\wedge \text{history}' = [\text{history} \text{ EXCEPT } ![i] = \text{msg.minitialHistory}]$   
 $\wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{ACKLD},$   
 $\text{mepoch} \mapsto \text{msg.mepoch},$   
 $\text{mhistory} \mapsto \text{msg.minitialHistory}])$   
 $\vee$  stale **NEWLEADER** – discard  
 $\wedge \neg \text{replyOk}$   
 $\wedge \text{Discard}(j, i)$   
 $\wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{leaderEpoch}, \text{history} \rangle$   
 $\wedge \text{UNCHANGED } \langle \text{state}, \text{commitIndex}, \text{leaderOracle}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, p$

In phase *l22*, pleader receives **ACK-LD** from a quorum of followers, and sends **COMMIT-LD** to followers.

$\text{LeaderHandleACKLD}(i, j) \triangleq$

test restrictions  
 $\wedge \text{currentEpoch}[i] \leq 2$   
 $\wedge \text{Len}(\text{history}[i]) \leq 2$   
 $\wedge \text{state}[i] = \text{ProspectiveLeader}$   
 $\wedge \text{msgs}[j][i] \neq \langle \rangle$   
 $\wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACKLD}$   
 $\wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1]$   
IN  $\vee$  new **ACK-LD** - accept  
 $\wedge \text{currentEpoch}[i] = \text{msg.mepoch}$   
 $\wedge \text{ackIndex}' = [\text{ackIndex} \text{ EXCEPT } ![i][j] = \text{Len}(\text{initialHistory}[i])]$   
 $\wedge \text{ackldRecv}' = [\text{ackldRecv} \text{ EXCEPT } ![i] = \text{IF } j \notin \text{ackldRecv}[i] \text{ THEN } \text{ackldRecv}[i] \cup \{j\}$   
 $\text{ELSE } \text{ackldRecv}[i]]$   
 $\vee$  stale **ACK-LD** - discard  
 $\wedge \text{currentEpoch}[i] \neq \text{msg.mepoch}$   
 $\wedge \text{UNCHANGED } \langle \text{ackldRecv}, \text{ackIndex} \rangle$   
 $\wedge \text{Discard}(j, i)$   
 $\wedge \text{UNCHANGED } \langle \text{serverVars}, \text{cluster}, \text{ceepochRecv}, \text{ackRecv}, \text{currentCounter},$   
 $\text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, p$

$\text{LeaderSync2}(i) \triangleq$

test restrictions  
 $\wedge \text{currentEpoch}[i] \leq 2$   
 $\wedge \text{Len}(\text{history}[i]) \leq 2$   
 $\wedge \text{state}[i] = \text{ProspectiveLeader}$   
 $\wedge \text{ackldRecv}[i] \in \text{Quorums}$   
 $\wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])]$   
 $\wedge \text{committedIndex}' = [\text{committedIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])]$   
 $\wedge \text{state}' = [\text{state} \text{ EXCEPT } ![i] = \text{Leader}]$   
 $\wedge \text{currentCounter}' = [\text{currentCounter} \text{ EXCEPT } ![i] = 0]$   
 $\wedge \text{sendCounter}' = [\text{sendCounter} \text{ EXCEPT } ![i] = 0]$   
 $\wedge \text{ackldRecv}' = [\text{ackldRecv} \text{ EXCEPT } ![i] = \{\text{NullPoint}\}]$   
 $\wedge \text{Broadcast}(i, [\text{mtype} \mapsto \text{COMMITLD},$   
 $\quad \text{mepoch} \mapsto \text{currentEpoch}[i],$   
 $\quad \text{mlength} \mapsto \text{Len}(\text{history}[i])])$   
 $\wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \text{cluster}, \text{cepochRecv},$   
 $\quad \text{ackRecv}, \text{ackIndex}, \text{initialHistory}, \text{tempVars}, \text{cepochSent}, \text{recoveryVars}, \text{proposalMsgs} \rangle$

In phase  $f22$ , follower receives  $\text{COMMIT-LD}$  and delivers all unprocessed transaction.

$\text{FollowerSync2}(i, j) \triangleq$   
test restrictions  
 $\wedge \text{currentEpoch}[i] \leq 2$   
 $\wedge \text{Len}(\text{history}[i]) \leq 2$   
 $\wedge \text{state}[i] = \text{Follower}$   
 $\wedge \text{msgs}[j][i] \neq \langle \rangle$   
 $\wedge \text{msgs}[j][i][1].\text{mtype} = \text{COMMITLD}$   
 $\wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1]$   
 $\quad \text{replyOk} \triangleq \wedge \text{currentEpoch}[i] = \text{msg.mepoch}$   
 $\quad \wedge \text{leaderOracle}[i] = j$   
IN  $\vee$  new  $\text{COMMIT-LD}$  - commit all transactions in initial history  
Regardless of  $\text{Restart}$ , it must be true because one will receive  $\text{NEWLEADER}$  before receiving  $\text{COMMIT-LD}$   
 $\wedge \text{replyOk}$   
 $\wedge \vee \wedge \text{Len}(\text{history}[i]) = \text{msg.mlength}$   
 $\quad \wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])]$   
 $\quad \wedge \text{Discard}(j, i)$   
 $\vee \wedge \text{Len}(\text{history}[i]) \neq \text{msg.mlength}$   
 $\quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{CEPOCH},$   
 $\quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i]])$   
 $\quad \wedge \text{UNCHANGED } \text{commitIndex}$   
 $\vee$   $>$  : stale  $\text{COMMIT-LD}$  - discard  
 $<$  : In our implementation, ' $<$ ' does not exist due to the guarantee of  $\text{Restart}$   
 $\wedge \neg \text{replyOk}$   
 $\wedge \text{Discard}(j, i)$   
 $\wedge \text{UNCHANGED } \text{commitIndex}$   
 $\wedge \text{UNCHANGED } \langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{history},$   
 $\quad \text{leaderVars}, \text{tempVars}, \text{cepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle$

---

In phase l31, leader receives client request and broadcasts *PROPOSE*.

$ClientRequest(i, v) \triangleq$

test restrictions

$\wedge currentEpoch[i] \leq 2$

$\wedge Len(history[i]) \leq 1$

$\wedge state[i] = Leader$

$\wedge currentCounter' = [currentCounter \text{ EXCEPT } ![i] = currentCounter[i] + 1]$

$\wedge LET newTransaction \triangleq [epoch \mapsto currentEpoch[i],$   
 $counter \mapsto currentCounter'[i],$   
 $value \mapsto v]$

IN  $\wedge history' = [history \text{ EXCEPT } ![i] = Append(history[i], newTransaction)]$

$\wedge ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(history'[i])] \text{ necessary, to push } commitIndex$

$\wedge UNCHANGED \langle msgs, state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, cluster, cepochRecv,$   
 $ackRecv, ackldRecv, sendCounter, initialHistory, committedIndex, tempVars, cepochSent \rangle$

$LeaderBroadcast1(i) \triangleq$

test restrictions

$\wedge currentEpoch[i] \leq 2$

$\wedge Len(history[i]) \leq 2$

$\wedge state[i] = Leader$

$\wedge sendCounter[i] < currentCounter[i]$

$\wedge LET toBeSentCounter \triangleq sendCounter[i] + 1$   
 $toBeSentIndex \triangleq Len(initialHistory[i]) + toBeSentCounter$   
 $toBeSentEntry \triangleq history[i][toBeSentIndex]$

IN  $\wedge Broadcast(i, [mtype \mapsto PROPOSE,$   
 $mepoch \mapsto currentEpoch[i],$   
 $mproposal \mapsto toBeSentEntry])$

$\wedge sendCounter' = [sendCounter \text{ EXCEPT } ![i] = toBeSentCounter]$

$\wedge LET m \triangleq [msource \mapsto i, mtype \mapsto PROPOSE, mepoch \mapsto currentEpoch[i], mproposal \mapsto toBeSentEntry]$

IN  $proposalMsgsLog' = proposalMsgsLog \cup \{m\}$

$\wedge UNCHANGED \langle serverVars, cepochRecv, cluster, ackRecv, ackldRecv, ackIndex,$   
 $currentCounter, initialHistory, committedIndex, tempVars, recoveryVars, cepochSent \rangle$

In phase f31, follower accepts proposal and append it to history.

$FollowerBroadcast1(i, j) \triangleq$

test restrictions

$\wedge currentEpoch[i] \leq 2$

$\wedge Len(history[i]) \leq 2$

$\wedge state[i] = Follower$

$\wedge msgs[j][i] \neq \langle \rangle$

$\wedge msgs[j][i][1].mtype = PROPOSE$

$\wedge LET msg \triangleq msgs[j][i][1]$   
 $replyOk \triangleq \wedge currentEpoch[i] = msg.mepoch$   
 $\wedge leaderOracle[i] = j$

IN  $\vee$  It should be that  $\vee \text{msg.mproposal.counter} = 1$   
 $\vee \text{msg.mrproposal.counter} = \text{history}[\text{Len}(\text{history})].\text{counter} + 1$   
 $\wedge \text{replyOk}$   
 $\wedge \text{history}' = [\text{history} \text{ EXCEPT } ![i] = \text{Append}(\text{history}[i], \text{msg.mproposal})]$   
 $\wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{ACK},$   
 $\text{mepoch} \mapsto \text{currentEpoch}[i],$   
 $\text{mindex} \mapsto \text{Len}(\text{history}'[i]))]$   
 $\vee$  If happens,  $\neq$  must be  $>$ , namely a stale leader sends it.  
 $\wedge \neg \text{replyOk}$   
 $\wedge \text{Discard}(j, i)$   
 $\wedge \text{UNCHANGED } \text{history}$   
 $\wedge \text{UNCHANGED } \langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{commitIndex},$   
 $\text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle$

In phase *l32*, leader receives ack from a quorum of followers to a certain proposal,  
and commits the proposal.

$\text{LeaderHandleACK}(i, j) \triangleq$

test restrictions  
 $\wedge \text{currentEpoch}[i] \leq 2$   
 $\wedge \text{Len}(\text{history}[i]) \leq 2$   
 $\wedge \text{state}[i] = \text{Leader}$   
 $\wedge \text{msgs}[j][i] \neq \langle \rangle$   
 $\wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACK}$   
 $\wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1]$   
IN  $\vee$  It should be that  $\text{ackIndex}[i][j] + 1 \triangleq \text{msg.mindex}$   
 $\wedge \text{currentEpoch}[i] = \text{msg.mepoch}$   
 $\wedge \text{ackIndex}' = [\text{ackIndex} \text{ EXCEPT } ![i][j] = \text{Maximum}(\{\text{ackIndex}[i][j], \text{msg.mindex}\})]$   
 $\vee$  If happens,  $\neq$  must be  $>$ , namely a stale follower sends it.  
 $\wedge \text{currentEpoch}[i] \neq \text{msg.mepoch}$   
 $\wedge \text{UNCHANGED } \text{ackIndex}$   
 $\wedge \text{Discard}(j, i)$   
 $\wedge \text{UNCHANGED } \langle \text{serverVars}, \text{cluster}, \text{ceepochRecv}, \text{ackRecv}, \text{ackldRecv}, \text{currentCounter},$   
 $\text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, p \rangle$

$\text{LeaderAdvanceCommit}(i) \triangleq$

test restrictions  
 $\wedge \text{currentEpoch}[i] \leq 2$   
 $\wedge \text{Len}(\text{history}[i]) \leq 2$   
 $\wedge \text{state}[i] = \text{Leader}$   
 $\wedge \text{commitIndex}[i] < \text{Len}(\text{history}[i])$   
 $\wedge \text{LET } \text{Agree}(\text{index}) \triangleq \{i\} \cup \{k \in (\text{Server} \setminus \{i\}) : \text{ackIndex}[i][k] \geq \text{index}\}$   
 $\text{agreeIndexes} \triangleq \{\text{index} \in (\text{commitIndex}[i] + 1) \dots \text{Len}(\text{history}[i]) : \text{Agree}(\text{index}) \in \text{Quorum}\}$   
 $\text{newCommitIndex} \triangleq \text{IF } \text{agreeIndexes} \neq \{\} \text{ THEN } \text{Maximum}(\text{agreeIndexes})$   
 $\text{ELSE } \text{commitIndex}[i]$   
IN  $\text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{newCommitIndex}]$

$$\begin{aligned}
& \wedge \text{UNCHANGED} \langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \\
& \quad \text{msgs}, \text{leaderVars}, \text{tempVars}, \text{epochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle \\
\text{LeaderBroadcast2}(i) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Leader} \\
& \wedge \text{committedIndex}[i] < \text{commitIndex}[i] \\
& \wedge \text{LET } \text{newCommittedIndex} \triangleq \text{committedIndex}[i] + 1 \\
& \quad \text{IN } \wedge \text{Broadcast}(i, [\text{mtype} \mapsto \text{COMMIT}, \\
& \quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i], \\
& \quad \quad \text{mindex} \mapsto \text{newCommittedIndex}, \\
& \quad \quad \text{mcounter} \mapsto \text{history}[i][\text{newCommittedIndex}].\text{counter}]) \\
& \quad \wedge \text{committedIndex}' = [\text{committedIndex} \text{ EXCEPT } ![i] = \text{committedIndex}[i] + 1] \\
& \wedge \text{UNCHANGED} \langle \text{serverVars}, \text{cluster}, \text{epochRecv}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \\
& \quad \text{sendCounter}, \text{initialHistory}, \text{tempVars}, \text{epochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle \\
\text{In phase } f32, \text{ follower receives } \text{COMMIT} \text{ and commits transaction.} \\
\text{FollowerBroadcast2}(i, j) & \triangleq \\
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Follower} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{COMMIT} \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{COMMIT} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \quad \text{replyOk} \triangleq \wedge \text{currentEpoch}[i] = \text{msg.mepoch} \\
& \quad \quad \wedge \text{leaderOracle}[i] = j \\
& \text{IN } \vee \wedge \text{replyOk} \\
& \quad \wedge \text{LET } \text{infoOk} \triangleq \wedge \text{Len}(\text{history}[i]) \geq \text{msg.mindex} \\
& \quad \quad \wedge \vee \wedge \text{msg.mindex} > 0 \\
& \quad \quad \quad \wedge \text{history}[i][\text{msg.mindex}].\text{epoch} = \text{msg.mepoch} \\
& \quad \quad \quad \wedge \text{history}[i][\text{msg.mindex}].\text{counter} = \text{msg.mcounter} \\
& \quad \quad \vee \text{msg.mindex} = 0 \\
& \text{IN } \vee \text{new COMMIT} - \text{commit transaction in history} \\
& \quad \wedge \text{infoOk} \\
& \quad \wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{Maximum}(\{\text{commitIndex}[i], \text{msg.mindex}\})] \\
& \quad \wedge \text{Discard}(j, i) \\
& \vee \text{It may happen when the server is a new follower who joined in the cluster,} \\
& \quad \text{and it misses the corresponding PROPOSE.} \\
& \quad \wedge \neg \text{infoOk} \\
& \quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{CEPOCH}, \\
& \quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i]])
\end{aligned}$$



$$\begin{aligned}
& \wedge \text{UNCHANGED } \text{commitIndex} \\
& \vee \text{stale } \text{COMMIT} - \text{discard} \\
& \wedge \neg \text{replyOk} \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \text{commitIndex} \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{history}, \text{leaderOracle}, \\
& \quad \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

There may be two ways to make sure all followers as up-to-date as the leader.

*way1*: choose *Send* not *Broadcast* when leader is going to send *PROPOSE* and *COMMIT*.

*way2*: When one follower receives *PROPOSE* or *COMMIT* which misses some entries between its history and the newest entry, the follower send *CEPOCH* to catch pace.

Here I choose *way2*, which I need not to rewrite *PROPOSE* and *COMMIT*, but need to modify the code when follower receives *COMMIT-LD* and *COMMIT*.

In phase *l33*, upon receiving *CEPOCH*, leader *l* proposes back *NEWEPOCH* and *NEWLEADER*.

*LeaderHandleCEPOCHinPhase3*(*i*, *j*)  $\triangleq$

$$\begin{aligned}
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Leader} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{CEPOCH} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \quad \text{IN } \vee \wedge \text{currentEpoch}[i] \geq \text{msg.mepoch} \\
& \quad \quad \wedge \text{Reply2}(i, j, [\text{mtype} \mapsto \text{NEWEPOCH}, \\
& \quad \quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i], \\
& \quad \quad \quad [\text{mtype} \mapsto \text{NEWLEADER}, \\
& \quad \quad \quad \text{mepoch} \mapsto \text{currentEpoch}[i], \\
& \quad \quad \quad \text{minitialHistory} \mapsto \text{history}[i]]) \\
& \vee \wedge \text{currentEpoch}[i] < \text{msg.mepoch} \\
& \quad \wedge \text{UNCHANGED } \text{msgs} \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

In phase *l34*, upon receiving ack from *f* of the *NEWLEADER*, it sends a commit message to *f*.

Leader *l* also makes  $Q := Q \cup \{f\}$ .

*LeaderHandleACKLDinPhase3*(*i*, *j*)  $\triangleq$

$$\begin{aligned}
& \text{test restrictions} \\
& \wedge \text{currentEpoch}[i] \leq 2 \\
& \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \wedge \text{state}[i] = \text{Leader} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACKLD} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \quad \text{aimCommitIndex} \triangleq \text{Minimum}(\{\text{commitIndex}[i], \text{Len}(\text{msg.mhistory})\})
\end{aligned}$$

$$\begin{aligned}
& aimCommitCounter \triangleq \text{IF } aimCommitIndex = 0 \text{ THEN } 0 \text{ ELSE } history[i][aimCommitIndex].co \\
\text{IN } & \vee \wedge currentEpoch[i] = msg.mepoch \\
& \wedge ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = Len(msg.mhistory)] \\
& \wedge Reply(i, j, [mtype \mapsto COMMIT, \\
& \quad mepoch \mapsto currentEpoch[i], \\
& \quad mindex \mapsto aimCommitIndex, \\
& \quad mcounter \mapsto aimCommitCounter]) \\
& \vee \wedge currentEpoch[i] \neq msg.mepoch \\
& \wedge Discard(j, i) \\
& \wedge \text{UNCHANGED } ackIndex \\
& \wedge cluster' = [cluster \text{ EXCEPT } ![i] = \text{IF } j \in cluster[i] \text{ THEN } cluster[i] \\
& \quad \quad \quad \text{ELSE } cluster[i] \cup \{j\}] \\
& \wedge \text{UNCHANGED } \langle serverVars, cepochRecv, ackRecv, ackldRecv, currentCounter, sendCounter, \\
& \quad initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, proposalMsgsLo,
\end{aligned}$$

To ensure any follower can find the correct leader, the follower should modify *leaderOracle* anytime when it receive messages from leader, because a server may restart and join the cluster *Q* halfway and receive the first message which is not *NEWEPOCH*. But we can delete this restriction when we ensure *Broadcast* function acts on the followers in the cluster not any servers in the whole system, then one server must has correct *leaderOracle* before it receives messages.

Let me suppose two conditions when one follower sends *CEPOCH* to leader:

0. Usually, the server becomes follower in election and sends *CEPOCH* before receiving *NEWEPOCH*.
1. The follower wants to join the cluster halfway and get the newest history.
2. The follower has received *COMMIT*, but there exists the gap between its own history and *mindex*, which means there are some transactions before *mindex* miss. Here we choose to send *CEPOCH* again, to receive the newest history from leader.

*BecomeFollower*(*i*)  $\triangleq$

$$\begin{aligned}
& \text{test restrictions} \\
& \wedge currentEpoch[i] \leq 2 \\
& \wedge Len(history[i]) \leq 2 \\
& \wedge state[i] \neq Follower \\
& \wedge \exists j \in Server \setminus \{i\} : \wedge msgs[j][i] \neq \langle \rangle \\
& \quad \wedge msgs[j][i][1].mtype \neq RECOVERYREQUEST \\
& \quad \wedge msgs[j][i][1].mtype \neq RECOVERYRESPONSE \\
& \quad \wedge \text{LET } msg \triangleq msgs[j][i][1] \\
& \quad \text{IN } \wedge NullPoint \in cepochRecv[i] \\
& \quad \wedge Maximum(\{currentEpoch[i], leaderEpoch[i]\}) < msg.mepoch \\
& \quad \wedge \vee msg.mtype = NEWEPOCH \\
& \quad \quad \vee msg.mtype = NEWLEADER \\
& \quad \quad \vee msg.mtype = COMMITLD \\
& \quad \quad \vee msg.mtype = PROPOSE \\
& \quad \quad \vee msg.mtype = COMMIT \\
& \quad \wedge state' = [state \text{ EXCEPT } ![i] = Follower] \\
& \quad \wedge currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch]
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{leaderOracle}' = [\text{leaderOracle} \text{ EXCEPT } ![i] = j] \\
& \wedge \text{Reply}(i, j, [mtype \mapsto \text{CEPOCH}, \\
& \quad \quad \quad mepoch \mapsto \text{currentEpoch}[i]]) \\
& \quad \quad \quad \text{Here we should not use } \text{Discard}. \\
& \wedge \text{UNCHANGED } \langle \text{leaderEpoch}, \text{history}, \text{commitIndex}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \\
& \quad \quad \quad \text{DiscardStaleMessage}(i) \triangleq \\
& \quad \quad \quad \text{test restrictions} \\
& \quad \quad \quad \wedge \text{currentEpoch}[i] \leq 2 \\
& \quad \quad \quad \wedge \text{Len}(\text{history}[i]) \leq 2 \\
& \quad \quad \quad \wedge \exists j \in \text{Server} \setminus \{i\} : \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \quad \quad \quad \quad \wedge \text{msgs}[j][i][1].mtype \neq \text{RECOVERYREQUEST} \\
& \quad \quad \quad \quad \wedge \text{msgs}[j][i][1].mtype \neq \text{RECOVERYRESPONSE} \\
& \quad \quad \quad \quad \wedge \text{LET } msg \triangleq \text{msgs}[j][i][1] \\
& \quad \quad \quad \quad \text{IN } \vee \wedge \text{state}[i] = \text{Follower} \\
& \quad \quad \quad \quad \quad \wedge \vee \text{msg.mepoch} < \text{currentEpoch}[i] \setminus * \text{ Discussed before.} \\
& \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{CEPOCH} \\
& \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{ACKE} \\
& \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{ACKLD} \\
& \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{ACK} \\
& \quad \quad \quad \quad \vee \wedge \text{state}[i] \neq \text{Follower} \\
& \quad \quad \quad \quad \quad \wedge \text{msg.mtype} \neq \text{CEPOCH} \\
& \quad \quad \quad \quad \quad \wedge \vee \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \quad \quad \quad \quad \quad \quad \wedge \vee \text{msg.mtype} = \text{ACK} \\
& \quad \quad \quad \quad \quad \quad \vee \wedge \text{msg.mepoch} \leq \text{Maximum}(\{\text{currentEpoch}[i], \text{leaderEpoch}[i]\}) \\
& \quad \quad \quad \quad \quad \quad \quad \wedge \vee \text{msg.mtype} = \text{NEWPOCH} \\
& \quad \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{NEWLEADER} \\
& \quad \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{COMMITLD} \\
& \quad \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{PROPOSE} \\
& \quad \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{COMMIT} \\
& \quad \quad \quad \quad \vee \wedge \text{state}[i] = \text{Leader} \\
& \quad \quad \quad \quad \quad \wedge \vee \text{msg.mtype} = \text{ACKE} \\
& \quad \quad \quad \quad \quad \vee \wedge \text{msg.mepoch} \leq \text{currentEpoch}[i] \\
& \quad \quad \quad \quad \quad \quad \wedge \vee \text{msg.mtype} = \text{NEWPOCH} \\
& \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{NEWLEADER} \\
& \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{COMMITLD} \\
& \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{PROPOSE} \\
& \quad \quad \quad \quad \quad \quad \vee \text{msg.mtype} = \text{COMMIT} \\
& \quad \quad \quad \quad \wedge \text{Discard}(j, i) \\
& \quad \quad \quad \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle \\
& \quad \quad \quad \text{Defines how the variables may transition.} \\
& \text{Next} \triangleq
\end{aligned}$$

$\vee \exists i \in \text{Server}, Q \in \text{Quorums} : \text{InitialElection}(i, Q)$   
 $\vee \exists i \in \text{Server} : \text{Restart}(i)$   
 $\vee \exists i \in \text{Server} : \text{RecoveryAfterRestart}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{HandleRecoveryRequest}(i, j)$   
 $\vee \exists i, j \in \text{Server} : \text{HandleRecoveryResponse}(i, j)$   
 $\vee \exists i, j \in \text{Server} : \text{FindCluster}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{LeaderTimeout}(i, j)$   
 $\vee \exists i \in \text{Server} : \text{FollowerTimeout}(i)$   
 $\vee \exists i \in \text{Server} : \text{FollowerDiscovery1}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{LeaderHandleCEPOCH}(i, j)$   
 $\vee \exists i \in \text{Server} : \text{LeaderDiscovery1}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{FollowerDiscovery2}(i, j)$   
 $\vee \exists i, j \in \text{Server} : \text{LeaderHandleACKE}(i, j)$   
 $\vee \exists i \in \text{Server} : \text{LeaderDiscovery2Sync1}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{FollowerSync1}(i, j)$   
 $\vee \exists i, j \in \text{Server} : \text{LeaderHandleACKLD}(i, j)$   
 $\vee \exists i \in \text{Server} : \text{LeaderSync2}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{FollowerSync2}(i, j)$   
 $\vee \exists i \in \text{Server}, v \in \text{Value} : \text{ClientRequest}(i, v)$   
 $\vee \exists i \in \text{Server} : \text{LeaderBroadcast1}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{FollowerBroadcast1}(i, j)$   
 $\vee \exists i, j \in \text{Server} : \text{LeaderHandleACK}(i, j)$   
 $\vee \exists i \in \text{Server} : \text{LeaderAdvanceCommit}(i)$   
 $\vee \exists i \in \text{Server} : \text{LeaderBroadcast2}(i)$   
 $\vee \exists i, j \in \text{Server} : \text{FollowerBroadcast2}(i, j)$   
 $\vee \exists i, j \in \text{Server} : \text{LeaderHandleCEPOCHinPhase3}(i, j)$   
 $\vee \exists i, j \in \text{Server} : \text{LeaderHandleACKLDinPhase3}(i, j)$   
 $\vee \exists i \in \text{Server} : \text{DiscardStaleMessage}(i)$   
 $\vee \exists i \in \text{Server} : \text{BecomeFollower}(i)$

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

---

Define some variants, safety propoties, and liveness propoties of *Zab* consensus algorithm.

Safety properties

There is most one leader/prospective leader in a certain epoch.

$\text{Leadership} \triangleq \forall i, j \in \text{Server} :$

$\wedge \vee \text{state}[i] = \text{Leader}$

$\vee \wedge \text{state}[i] = \text{ProspectiveLeader}$

$\wedge \text{NullPoint} \in \text{ackeRecv}[i]$  prospective leader determines its epoch after broadcasting *NEWLE*

$\wedge \vee \text{state}[j] = \text{Leader}$

$\vee \wedge \text{state}[j] = \text{ProspectiveLeader}$

$\wedge \text{NullPoint} \in \text{ackeRecv}[j]$

$\wedge \text{currentEpoch}[i] = \text{currentEpoch}[j]$

$$\Rightarrow i = j$$

Here, delivering means deliver some transaction from history to replica. We can assume  $deliverIndex = commitIndex$ .

So we can assume the set of delivered transactions is the prefix of history with index from 1 to  $commitIndex$ .

We can express a transaction by two-tuple  $\langle epoch, counter \rangle$  according to its uniqueness.

$$\begin{aligned} equal(entry1, entry2) \triangleq & \quad \wedge entry1.epoch = entry2.epoch \\ & \quad \wedge entry1.counter = entry2.counter \end{aligned}$$

$$\begin{aligned} precede(entry1, entry2) \triangleq & \quad \vee entry1.epoch < entry2.epoch \\ & \quad \vee \wedge entry1.epoch = entry2.epoch \\ & \quad \quad \wedge entry1.counter < entry2.counter \end{aligned}$$

*PrefixConsistency*: The prefix that have been delivered in history in any process is the same.

$$\begin{aligned} PrefixConsistency \triangleq & \quad \forall i, j \in Server : \\ & \quad LET \ smaller \triangleq Minimum(\{commitIndex[i], commitIndex[j]\}) \\ & \quad IN \quad \vee smaller = 0 \\ & \quad \quad \vee \wedge smaller > 0 \\ & \quad \quad \quad \wedge \forall index \in 1 \dots smaller : equal(history[i][index], history[j][index]) \end{aligned}$$

*Integrity*: If some follower delivers one transaction, then some primary has broadcast it.

$$\begin{aligned} Integrity \triangleq & \quad \forall i \in Server : \\ & \quad state[i] = Follower \wedge commitIndex[i] > 0 \\ & \quad \Rightarrow \forall index \in 1 \dots commitIndex[i] : \exists msg \in proposalMsgsLog : \\ & \quad \quad equal(msg.mproposal, history[i][index]) \end{aligned}$$

*Agreement*: If some follower  $f$  delivers transaction a and some follower  $f'$  delivers transaction b, then  $f'$  delivers a or  $f$  delivers b.

$$\begin{aligned} Agreement \triangleq & \quad \forall i, j \in Server : \\ & \quad \wedge state[i] = Follower \wedge commitIndex[i] > 0 \\ & \quad \wedge state[j] = Follower \wedge commitIndex[j] > 0 \\ & \quad \Rightarrow \\ & \quad \forall index1 \in 1 \dots commitIndex[i], index2 \in 1 \dots commitIndex[j] : \\ & \quad \quad \vee \exists indexj \in 1 \dots commitIndex[j] : \\ & \quad \quad \quad equal(history[j][indexj], history[i][index1]) \\ & \quad \quad \vee \exists indexi \in 1 \dots commitIndex[i] : \\ & \quad \quad \quad equal(history[i][indexi], history[j][index2]) \end{aligned}$$

*Total order*: If some follower delivers a before b, then any process that delivers b must also deliver a and deliver a before b.

$$\begin{aligned} TotalOrder \triangleq & \quad \forall i, j \in Server : commitIndex[i] \geq 2 \wedge commitIndex[j] \geq 2 \\ & \quad \Rightarrow \forall indexi1 \in 1 \dots (commitIndex[i] - 1) : \forall indexi2 \in (indexi1 + 1) \dots commitIndex[i] : \\ & \quad \quad LET \ logOk \triangleq \exists index \in 1 \dots commitIndex[j] : equal(history[i][indexi2], history[j][index]) \\ & \quad \quad IN \quad \vee \neg logOk \\ & \quad \quad \vee \wedge logOk \\ & \quad \quad \quad \wedge \exists indexj2 \in 1 \dots commitIndex[j] : \\ & \quad \quad \quad \quad \wedge equal(history[i][indexi2], history[j][indexj2]) \\ & \quad \quad \quad \quad \wedge \exists indexj1 \in 1 \dots (indexj2 - 1) : equal(history[i][indexi1], history[j][indexj1]) \end{aligned}$$

Local primary order: If a primary broadcasts a before it broadcasts b, then a follower that delivers b must also deliver a before b.

$$\begin{aligned}
LocalPrimaryOrder \triangleq & \text{LET } mset(i, e) \triangleq \{msg \in proposalMsgsLog : msg.msource = i \wedge msg.mproposal.epoch = e\} \\
& mentries(i, e) \triangleq \{msg.mproposal : msg \in mset(i, e)\} \\
\text{IN } & \forall i \in Server : \forall e \in 1 \dots currentEpoch[i] : \\
& \wedge Cardinality(mentries(i, e)) \geq 2 \\
& \wedge \exists tsc1 \in mentries(i, e) : \exists tsc2 \in mentries(i, e) : \\
& \quad \wedge \neg equal(tsc2, tsc1) \\
& \quad \wedge \text{LET } tscPre \triangleq \text{IF } precede(tsc1, tsc2) \text{ THEN } tsc1 \text{ ELSE } tsc2 \\
& \quad \quad tscNext \triangleq \text{IF } precede(tsc1, tsc2) \text{ THEN } tsc2 \text{ ELSE } tsc1 \\
& \text{IN } \forall j \in Server : \wedge commitIndex[j] \geq 2 \\
& \quad \wedge \exists index \in 1 \dots commitIndex[j] : equal(history[j][index], tscPre) \\
\Rightarrow & \\
& \exists index2 \in 1 \dots commitIndex[j] : \\
& \quad \wedge equal(history[j][index2], tscNext) \\
& \quad \wedge index2 > 1 \\
& \quad \wedge \exists index1 \in 1 \dots (index2 - 1) : equal(history[j][index1], tscPre)
\end{aligned}$$

Global primary order: A follower f delivers both a with epoch  $e$  and b with epoch  $e'$ , and  $e < e'$ , then f must deliver a before b.

$$\begin{aligned}
GlobalPrimaryOrder \triangleq & \forall i \in Server : commitIndex[i] \geq 2 \\
\Rightarrow & \forall idx1, idx2 \in 1 \dots commitIndex[i] : \vee history[i][idx1].epoch \geq history[i][idx2].epoch \\
& \vee \wedge history[i][idx1].epoch < history[i][idx2].epoch \\
& \wedge idx1 < idx2
\end{aligned}$$

Primary integrity: If primary  $p$  broadcasts a and some follower  $f$  delivers b such that b has epoch smaller than epoch of  $p$ , then  $p$  must deliver b before it broadcasts a.

$$\begin{aligned}
PrimaryIntegrity \triangleq & \forall i, j \in Server : \wedge state[i] = Leader \\
& \wedge state[j] = Follower \wedge commitIndex[j] \geq 1 \\
\Rightarrow & \forall index \in 1 \dots commitIndex[j] : \vee history[j][index].epoch \geq currentEpoch[i] \\
& \vee \wedge history[j][index].epoch < currentEpoch[i] \\
& \wedge \exists idx \in 1 \dots commitIndex[i] : equal(history[i][idx], history[j][index])
\end{aligned}$$

Liveness property

Suppose that :

- A quorum  $Q$  of followers are up.
- The followers in  $Q$  elect the same process  $l$  and  $l$  is up.
- Messages between a follower in  $Q$  and  $l$  are received in a timely fashion.

If  $l$  proposes a transaction  $a$ , then  $a$  is eventually committed.

---

\ \* Modification History  
\ \* Last modified *Fri Apr 30 15:47:49 CST 2021* by Dell  
\ \* Created Sat *Dec 05 13:32:08 CST 2020* by Dell