
MODULE *Zab*

This is the formal specification for the *Zab* consensus algorithm,
which means *Zookeeper Atomic Broadcast*.

This work is driven by *Flavio P. Junqueira*, “*Zab*: High-performance broadcast for primary-backup systems”

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

The set of server identifiers
CONSTANT *Server*

The set of requests that can go into history
CONSTANT *Value*

Server states
It is unnecessary to add state *ELECTION*, we can own it by setting *leaderOracle* to Null.
CONSTANTS *Follower, Leader, ProspectiveLeader*

Message types
CONSTANTS *CEPOCH, NEWEPOCH, ACKE, NEWLEADER, ACKLD, COMMITLD, PROPOSE, ACK, C*

Additional Message types used for recovery when restarting
CONSTANTS *RECOVERYREQUEST, RECOVERYRESPONSE*

the maximum round of epoch, currently not used
CONSTANT *Epoches*

Return the maximum value from the set *S*
 $Maximum(S) \triangleq \text{IF } S = \{\} \text{ THEN } -1$
 $\text{ELSE CHOOSE } n \in S : \forall m \in S : n \geq m$

Return the minimum value from the set *S*
 $Minimum(S) \triangleq \text{IF } S = \{\} \text{ THEN } -1$
 $\text{ELSE CHOOSE } n \in S : \forall m \in S : n \leq m$

$Quorums \triangleq \{Q \in \text{SUBSET } Server : Cardinality(Q) * 2 > Cardinality(Server)\}$
 ASSUME $QuorumsAssumption \triangleq \wedge \forall Q \in Quorums : Q \subseteq Server$
 $\wedge \forall Q1, Q2 \in Quorums : Q1 \cap Q2 \neq \{\}$

$None \triangleq \text{CHOOSE } v : v \notin Value$

$NullPoint \triangleq \text{CHOOSE } p : p \notin Server$

The server's *state(Follower, Leader, ProspectiveLeader)*.
VARIABLE *state*

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*

Hepler 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 $\langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \text{commitIndex} \rangle$

leaderVars \triangleq $\langle \text{cluster}, \text{ceepochRecv}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \text{sendCounter}, \text{initialHistory} \rangle$

tempVars \triangleq $\langle \text{tempMaxEpoch}, \text{tempMaxLastEpoch}, \text{tempInitialHistory} \rangle$

recoveryVars \triangleq $\langle \text{recoveryRespRecv}, \text{recoveryMaxEpoch}, \text{recoveryMEOracle}, \text{recoverySent} \rangle$

vars \triangleq $\langle \text{serverVars}, \text{msgs}, \text{leaderVars}, \text{tempVars}, \text{recoveryVars}, \text{ceepochSent}, \text{proposalMsgsLog} \rangle$

LastZxid(his) \triangleq IF *Len(his)* > 0 THEN $\langle \text{his}[\text{Len}(\text{his})].\text{epoch}, \text{his}[\text{Len}(\text{his})].\text{counter} \rangle$
 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* EXCEPT ![*i*][*j*] = *Append*(*msgs*[*i*][*j*], *m*)]

Send2(*i*, *j*, *m1*, *m2*) \triangleq *msgs'* = [*msgs* 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'* = IF *msgs*[*i*][*j*] $\neq \langle \rangle$ THEN [*msgs* EXCEPT ![*i*][*j*] = *Tail*(*msgs*[*i*][*j*])]
 ELSE *msgs*

Leader/Pleader broadcasts a message to all other servers in *Q*

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

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

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

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

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

Define initial values for all variables

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

A server becomes pleader and a quorum servers knows that.

$$\begin{aligned}
Election(i, Q) &\triangleq \\
&\wedge i \in Q \\
&\wedge state' \\
&= [s \in Server \mapsto \text{IF } s = i \text{ THEN } ProspectiveLeader \\
&\quad \text{ELSE IF } s \in Q \text{ THEN } Follower \\
&\quad \text{ELSE } state[s]] \\
&\wedge cluster' \\
&= [cluster \text{ EXCEPT } ![i] = Q] \quad \text{cluster is first initialized in election, not } phase1. \\
&\wedge cepochRecv' \\
&= [cepochRecv \text{ EXCEPT } ![i] = \{i\}] \\
&\wedge ackeRecv' \\
&= [ackeRecv \text{ EXCEPT } ![i] = \{i\}] \\
&\wedge ackldRecv' \\
&= [ackldRecv \text{ EXCEPT } ![i] = \{i\}] \\
&\wedge ackIndex' \\
&= [ii \in Server \mapsto [ij \in Server \mapsto \\
&\quad \text{IF } ii = i \text{ THEN } 0 \\
&\quad \text{ELSE } ackIndex[ii][ij]]] \\
&\wedge committedIndex' \\
&= [committedIndex \text{ EXCEPT } ![i] = 0] \\
&\wedge initialHistory' \\
&= [initialHistory \text{ EXCEPT } ![i] = \langle \rangle] \\
&\wedge tempMaxEpoch' \\
&= [tempMaxEpoch \text{ EXCEPT } ![i] = currentEpoch[i]] \\
&\wedge tempMaxLastEpoch' \\
&= [tempMaxLastEpoch \text{ EXCEPT } ![i] = currentEpoch[i]] \\
&\wedge tempInitialHistory' \\
&= [tempInitialHistory \text{ EXCEPT } ![i] = history[i]] \\
&\wedge leaderOracle' \\
&= [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } i \\
&\quad \text{ELSE } leaderOracle[s]] \\
&\wedge leaderEpoch' \\
&= [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } currentEpoch[s] \\
&\quad \text{ELSE } leaderEpoch[s]] \\
&\wedge cepochSent' \\
&= [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } FALSE \\
&\quad \text{ELSE } cepochSent[s]] \\
&\wedge msgs' \\
&= [ii \in Server \mapsto [ij \in Server \mapsto \\
&\quad \text{IF } ii \in Q \vee ij \in Q \text{ THEN } \langle \rangle \\
&\quad \text{ELSE } msgs[ii][ij]]]
\end{aligned}$$

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.

$$\begin{aligned}
InitialElection(i, Q) &\triangleq \\
&\wedge \forall s \in Server : state[i] = Follower \wedge leaderOracle[i] = NullPoint \\
&\wedge Election(i, Q) \\
&\wedge \text{UNCHANGED } \langle currentEpoch, history, commitIndex, currentCounter, sendCounter, recoveryVars, pr \rangle
\end{aligned}$$

The leader finds timeout with another follower.

$$\begin{aligned}
LeaderTimeout(i, j) &\triangleq \\
&\wedge state[i] \neq Follower \\
&\wedge j \neq i \\
&\wedge j \in cluster[i] \\
&\wedge \text{LET } newCluster \triangleq cluster[i] \setminus \{j\} \\
&\text{IN } \wedge \vee \wedge newCluster \in Quorums \\
&\quad \wedge cluster' = [cluster \text{ EXCEPT } ![i] = newCluster] \\
&\quad \wedge clean(i, j) \\
&\quad \wedge \text{UNCHANGED } \langle state, cepochRecv, ackeRecv, ackldRecv, ackIndex, committedIndex, initialHistory, \\
&\quad \quad tempMaxEpoch, tempMaxLastEpoch, tempInitialHistory, leaderOracle, leaderEpoch, cepochSent, msgs \rangle
\end{aligned}$$

$$\begin{aligned}
& \vee \wedge \text{newCluster} \notin \text{Quorums} \\
& \wedge \text{LET } Q \triangleq \text{CHOOSE } q \in \text{Quorums}: i \in q \\
& \quad v \triangleq \text{CHOOSE } s \in Q: \text{TRUE} \\
& \text{IN } \text{Election}(v, Q) \\
& \exists Q \in \text{Quorums} : \wedge i \in Q \\
& \quad \wedge \exists v \in Q : \text{Election}(v, Q) \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{history}, \text{commitIndex}, \text{currentCounter}, \text{sendCounter}, \text{recoveryVars}, \text{proposals} \rangle
\end{aligned}$$

A follower finds timeout with the leader.

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

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.

$$\begin{aligned}
\text{Restart}(i) & \triangleq \\
& \wedge \text{state}' = [\text{state} \text{ EXCEPT } ![i] = \text{Follower}] \\
& \wedge \text{leaderOracle}' = [\text{leaderOracle} \text{ EXCEPT } ![i] = \text{NullPoint}] \\
& \wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = 0] \\
& \wedge \text{cephochSent}' = [\text{cephochSent} \text{ EXCEPT } ![i] = \text{FALSE}] \\
& \wedge \text{msgs}' = [ii \in \text{Server} \mapsto [ij \in \text{Server} \mapsto \text{IF } ij = i \text{ THEN } \langle \rangle \\
& \quad \text{ELSE } \text{msgs}[ii][ij]]] \\
& \wedge \text{recoverySent}' = [\text{recoverySent} \text{ EXCEPT } ![i] = \text{FALSE}] \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{leaderEpoch}, \text{history}, \text{leaderVars}, \text{tempVars}, \\
& \quad \text{recoveryRespRecv}, \text{recoveryMaxEpoch}, \text{recoveryMEOracle}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

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

$$\begin{aligned}
\text{HandleRecoveryRequest}(i, j) & \triangleq \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{RECOVERYREQUEST} \\
& \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{RECOVERYRESPONSE}], \text{msgs}[j][i][2])
\end{aligned}$$

$$\begin{aligned}
& \text{moracle} \mapsto \text{leaderOracle}[i], \\
& \text{mepoch} \mapsto \text{currentEpoch}[i]) \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle \\
\text{HandleRecoveryResponse}(i, j) & \triangleq \\
& \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} \quad \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}] \\
& \quad \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} \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoverySent}, \text{proposalMsgsLog} \rangle \\
\text{FindCluster}(i) & \triangleq \\
& \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} \quad \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 \\
& \quad \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{ceepochSent}, \text{proposalMsgs} \rangle
\end{aligned}$$

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

$$\begin{aligned}
\text{FollowerDiscovery1}(i) & \triangleq \\
& \wedge \text{state}[i] = \text{Follower} \\
& \wedge \text{leaderOracle}[i] \neq \text{NullPoint} \\
& \wedge \neg \text{ceepochSent}[i] \\
& \wedge \text{LET } \text{leader} \triangleq \text{leaderOracle}[i]
\end{aligned}$$

IN $Send(i, leader, [mtype \mapsto CEPOCH,$
 $mepoch \mapsto currentEpoch[i]])$
 $\wedge cepochSent' = [ceepochSent \text{ EXCEPT } ![i] = \text{TRUE}]$
 $\wedge \text{UNCHANGED } \langle serverVars, leaderVars, tempVars, recoveryVars, proposalMsgsLog \rangle$

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

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

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 $leaderEpoch$, and use it in handling *ACK-E*.

$LeaderDiscovery1(i) \triangleq$
 $\wedge state[i] = ProspectiveLeader$
 $\wedge cepochRecv[i] \in Quorums$
 $\wedge leaderEpoch' = [leaderEpoch \text{ EXCEPT } ![i] = tempMaxEpoch[i] + 1]$
 $\wedge cepochRecv' = [ceepochRecv \text{ EXCEPT } ![i] = \{NullPoint\}]$
 $\wedge Broadcast(i, [mtype \mapsto NEWEPOCH,$
 $mepoch \mapsto leaderEpoch'[i]])$
 $\wedge \text{UNCHANGED } \langle state, currentEpoch, leaderOracle, history, cluster, ackeRecv, ackldRecv, ackIndex, cv$
 $initialHistory, commitIndex, committedIndex, cepochSent, tempVars, recoveryVars, p$

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

$FollowerDiscovery2(i, j) \triangleq$

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

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.

$LeaderHandleACKE(i, j) \triangleq$

- $\wedge state[i] = ProspectiveLeader$
- $\wedge msgs[j][i] \neq \langle \rangle$
- $\wedge msgs[j][i][1].mtype = ACKE$
- $\wedge LET\ msg \triangleq msgs[j][i][1]$
- $infoOk \triangleq \vee msg.mlastEpoch > tempMaxLastEpoch[i]$
- $\vee \wedge msg.mlastEpoch = tempMaxLastEpoch[i]$

$$\begin{aligned}
& \wedge \vee LastZxid(msg.mhf)[1] > LastZxid(tempInitialHistory[i])[1] \\
& \vee \wedge LastZxid(msg.mhf)[1] = LastZxid(tempInitialHistory[i])[1] \\
& \wedge LastZxid(msg.mhf)[2] \geq LastZxid(tempInitialHistory[i])[2] \\
IN \quad & \vee \wedge leaderEpoch[i] = msg.mepoch \\
& \wedge \vee \wedge infoOk \\
& \quad \wedge tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = msg.mlastEpoch] \\
& \quad \wedge tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = msg.mhf] \\
& \vee \wedge \neg infoOk \\
& \quad \wedge UNCHANGED \langle tempMaxLastEpoch, tempInitialHistory \rangle \\
& \quad \text{Followers not in } Q \text{ will not receive } NEWEPOCH, \text{ so leader will receive } ACKE \text{ only when the source is in } Q \\
& \quad \wedge ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackeRecv[i] \text{ THEN } ackeRecv[i] \cup \{j\} \\
& \quad \quad \quad \text{ELSE } ackeRecv[i]] \\
& \vee \wedge leaderEpoch[i] \neq msg.mepoch \\
& \quad \wedge UNCHANGED \langle tempMaxLastEpoch, tempInitialHistory, ackeRecv \rangle \\
& \wedge Discard(j, i) \\
& \wedge UNCHANGED \langle serverVars, cluster, cepochRecv, ackldRecv, ackIndex, currentCounter, \\
& \quad \quad \quad sendCounter, initialHistory, committedIndex, cepochSent, tempMaxEpoch, recoveryVars \rangle \\
LeaderDiscovery2Sync1(i) & \triangleq \\
& \wedge state[i] = ProspectiveLeader \\
& \wedge ackeRecv[i] \in Quorums \\
& \wedge currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = leaderEpoch[i]] \\
& \wedge history' = [history \text{ EXCEPT } ![i] = tempInitialHistory[i]] \\
& \wedge initialHistory' = [initialHistory \text{ EXCEPT } ![i] = tempInitialHistory[i]] \\
& \wedge ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{NullPoint\}] \\
& \wedge ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(tempInitialHistory[i])] \\
& \text{until now, phase1(Discovery) ends} \\
& \wedge Broadcast(i, [mtype \mapsto NEWLEADER, \\
& \quad \quad \quad mepoch \mapsto currentEpoch'[i], \\
& \quad \quad \quad minitialHistory \mapsto history'[i]]) \\
& \wedge UNCHANGED \langle state, leaderEpoch, leaderOracle, commitIndex, cluster, cepochRecv, ackldRecv, \\
& \quad \quad \quad currentCounter, sendCounter, committedIndex, cepochSent, tempVars, recoveryVars,
\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 *cepochRecv*, *ackeRecv*, *ackldRecv* to *{NullPoint}* in corresponding three actions to make sure that the prospective leader will not broadcast *NEWEPOCH/NEWLEADER/COMMITLD* twice.

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

$$\begin{aligned}
FollowerSync1(i, j) & \triangleq \\
& \wedge state[i] = Follower \\
& \wedge msgs[j][i] \neq \langle \rangle \\
& \wedge msgs[j][i][1].mtype = NEWLEADER \\
& \wedge LET msg \triangleq msgs[j][i][1]
\end{aligned}$$

$$\begin{aligned}
& \text{replyOk} \triangleq \wedge \text{currentEpoch}[i] \leq \text{msg.mepoch} \\
& \quad \wedge \text{leaderOracle}[i] = j \\
\text{IN } & \vee \text{new NEWLEADER - accept and reply} \\
& \wedge \text{replyOk} \\
& \wedge \text{currentEpoch}' = [\text{currentEpoch} \text{ EXCEPT } ![i] = \text{msg.mepoch}] \\
& \wedge \text{leaderEpoch}' = [\text{leaderEpoch} \text{ EXCEPT } ![i] = \text{msg.mepoch}] \\
& \wedge \text{history}' = [\text{history} \text{ EXCEPT } ![i] = \text{msg.minitialHistory}] \\
& \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{ACKLD}, \\
& \quad \text{mepoch} \mapsto \text{msg.mepoch}, \\
& \quad \text{mhistory} \mapsto \text{msg.minitialHistory}]) \\
& \vee \text{stale NEWLEADER - discard} \\
& \wedge \neg \text{replyOk} \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \langle \text{currentEpoch}, \text{leaderEpoch}, \text{history} \rangle \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{commitIndex}, \text{leaderOracle}, \text{leaderVars}, \text{tempVars}, \text{cepocheSent}, \text{recoveryVars}, p \rangle
\end{aligned}$$

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

$$\begin{aligned}
& \text{LeaderHandleACKLD}(i, j) \triangleq \\
& \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACKLD} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \text{IN } \vee \text{new ACK-LD - accept} \\
& \quad \wedge \text{currentEpoch}[i] = \text{msg.mepoch} \\
& \quad \wedge \text{ackIndex}' = [\text{ackIndex} \text{ EXCEPT } ![i][j] = \text{Len}(\text{initialHistory}[i])] \\
& \quad \wedge \text{ackldRecv}' = [\text{ackldRecv} \text{ EXCEPT } ![i] = \text{IF } j \notin \text{ackldRecv}[i] \text{ THEN } \text{ackldRecv}[i] \cup \{j\} \\
& \quad \quad \quad \text{ELSE } \text{ackldRecv}[i]] \\
& \vee \text{stale ACK-LD - discard} \\
& \quad \wedge \text{currentEpoch}[i] \neq \text{msg.mepoch} \\
& \quad \wedge \text{UNCHANGED } \langle \text{ackldRecv}, \text{ackIndex} \rangle \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{cluster}, \text{cepocheRecv}, \text{ackRecv}, \text{currentCounter}, \\
& \quad \text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{tempVars}, \text{cepocheSent}, \text{recoveryVars}, p \rangle
\end{aligned}$$

$$\begin{aligned}
& \text{LeaderSync2}(i) \triangleq \\
& \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \wedge \text{ackldRecv}[i] \in \text{Quorums} \\
& \wedge \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])] \\
& \wedge \text{committedIndex}' = [\text{committedIndex} \text{ EXCEPT } ![i] = \text{Len}(\text{history}[i])] \\
& \wedge \text{state}' = [\text{state} \text{ EXCEPT } ![i] = \text{Leader}] \\
& \wedge \text{currentCounter}' = [\text{currentCounter} \text{ EXCEPT } ![i] = 0] \\
& \wedge \text{sendCounter}' = [\text{sendCounter} \text{ EXCEPT } ![i] = 0] \\
& \wedge \text{ackldRecv}' = [\text{ackldRecv} \text{ EXCEPT } ![i] = \{\text{NullPoint}\}] \\
& \wedge \text{Broadcast}(i, [\text{mtype} \mapsto \text{COMMITLD}, \\
& \quad \text{mepoch} \mapsto \text{currentEpoch}[i],
\end{aligned}$$

$$\begin{aligned}
& mlength \mapsto Len(history[i]) \\
\wedge \quad & \text{UNCHANGED } \langle currentEpoch, leaderEpoch, leaderOracle, history, cluster, cepochRecv, \\
& \quad ackRecv, ackIndex, initialHistory, tempVars, cepochSent, recoveryVars, proposalM \\
& \text{In phase } f22, \text{ follower receives } COMMIT-LD \text{ and delivers all unprocessed transaction.} \\
FollowerSync2(i, j) \triangleq & \\
& \wedge state[i] = Follower \\
& \wedge msgs[j][i] \neq \langle \rangle \\
& \wedge msgs[j][i][1].mtype = COMMITLD \\
& \wedge \text{LET } msg \triangleq msgs[j][i][1] \\
& \quad replyOk \triangleq \wedge currentEpoch[i] = msg.mepoch \\
& \quad \quad \wedge leaderOracle[i] = j \\
\text{IN } \quad & \vee \quad \text{new } COMMIT-LD - \text{commit all transactions in initial history} \\
& \quad \text{Regardless of } Restart, \text{ it must be true because one will receive } NEWLEADER \text{ before receiving } COMMIT-LD \\
& \wedge replyOk \\
& \wedge \vee \wedge Len(history[i]) = msg.mlength \\
& \quad \wedge commitIndex' = [commitIndex \text{ EXCEPT } ![i] = Len(history[i])] \\
& \quad \wedge Discard(j, i) \\
& \vee \wedge Len(history[i]) \neq msg.mlength \\
& \quad \wedge Reply(i, j, [mtype \mapsto CEPOCH, \\
& \quad \quad mepoch \mapsto currentEpoch[i]]) \\
& \quad \wedge \text{UNCHANGED } commitIndex \\
& \vee \quad > : \text{stale } COMMIT-LD - \text{discard} \\
& \quad < : \text{In our implementation, ' < ' does not exist due to the guarantee of } Restart \\
& \wedge \neg replyOk \\
& \wedge Discard(j, i) \\
& \wedge \text{UNCHANGED } commitIndex \\
& \wedge \text{UNCHANGED } \langle state, currentEpoch, leaderEpoch, leaderOracle, history, \\
& \quad leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog \rangle
\end{aligned}$$

$$\begin{aligned}
& \text{In phase } l31, \text{ leader receives client request and broadcasts } PROPOSE. \\
ClientRequest(i, v) \triangleq & \\
& \wedge state[i] = Leader \\
& \wedge currentCounter' = [currentCounter \text{ EXCEPT } ![i] = currentCounter[i] + 1] \\
& \wedge \text{LET } newTransaction \triangleq [epoch \mapsto currentEpoch[i], \\
& \quad counter \mapsto currentCounter'[i], \\
& \quad value \mapsto v] \\
\text{IN } \quad & \wedge history' = [history \text{ EXCEPT } ![i] = Append(history[i], newTransaction)] \\
& \wedge ackIndex' = [ackIndex \text{ EXCEPT } ![i][i] = Len(history'[i])] \quad \text{necessary, to push } commitIndex \\
& \wedge \text{UNCHANGED } \langle msgs, state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, cluster, cepochRecv, \\
& \quad ackRecv, ackldRecv, sendCounter, initialHistory, committedIndex, tempVars, cepoch \\
LeaderBroadcast1(i) \triangleq & \\
& \wedge state[i] = Leader \\
& \wedge sendCounter[i] < currentCounter[i]
\end{aligned}$$

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

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

$FollowerBroadcast1(i, j) \triangleq$
 $\wedge state[i] = Follower$
 $\wedge msgs[j][i] \neq \langle \rangle$
 $\wedge msgs[j][i][1].mtype = PROPOSE$
 \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 $\vee msg.mproposal.counter = 1$
 $\vee msg.mproposal.counter = history[Len(history)].counter + 1$
 $\wedge replyOk$
 $\wedge history' = [history \text{ EXCEPT } ![i] = Append(history[i], msg.mproposal)]$
 $\wedge Reply(i, j, [mtype \mapsto ACK,$
 $mepoch \mapsto currentEpoch[i],$
 $mindex \mapsto Len(history'[i])])$
 \vee If happens, \neq must be $>$, namely a stale leader sends it.
 $\wedge \neg replyOk$
 $\wedge Discard(j, i)$
 \wedge UNCHANGED $history$
 \wedge UNCHANGED $\langle state, currentEpoch, leaderEpoch, leaderOracle, commitIndex,$
 $leaderVars, tempVars, cepochSent, recoveryVars, proposalMsgsLog \rangle$

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

$LeaderHandleACK(i, j) \triangleq$
 $\wedge state[i] = Leader$
 $\wedge msgs[j][i] \neq \langle \rangle$
 $\wedge msgs[j][i][1].mtype = ACK$
 \wedge LET $msg \triangleq msgs[j][i][1]$
 IN \vee It should be that $ackIndex[i][j] + 1 \triangleq msg.mindex$
 $\wedge currentEpoch[i] = msg.mepoch$
 $\wedge ackIndex' = [ackIndex \text{ EXCEPT } ![i][j] = Maximum(\{ackIndex[i][j], msg.mindex\})]$
 \vee If happens, \neq must be $>$, namely a stale follower sends it.

$$\begin{aligned}
& \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}, \\
& \quad \text{sendCounter}, \text{initialHistory}, \text{committedIndex}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, p \rangle \\
\text{LeaderAdvanceCommit}(i) & \triangleq \\
& \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}\} \\
& \quad \text{agreeIndexes} \triangleq \{\text{index} \in (\text{commitIndex}[i] + 1) \dots \text{Len}(\text{history}[i]) : \text{Agree}(\text{index}) \in \text{Quorum}\} \\
& \quad \text{newCommitIndex} \triangleq \text{IF } \text{agreeIndexes} \neq \{\} \text{ THEN } \text{Maximum}(\text{agreeIndexes}) \\
& \quad \quad \quad \text{ELSE } \text{commitIndex}[i] \\
& \text{IN } \text{commitIndex}' = [\text{commitIndex} \text{ EXCEPT } ![i] = \text{newCommitIndex}] \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{leaderOracle}, \text{history}, \\
& \quad \text{msgs}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle \\
\text{LeaderBroadcast2}(i) & \triangleq \\
& \wedge \text{state}[i] = \text{Leader} \\
& \wedge \text{committedIndex}[i] < \text{commitIndex}[i] \\
& \wedge \text{LET } \text{newCommittedIndex} \triangleq \text{committedIndex}[i] + 1 \\
& \text{IN } \wedge \text{Broadcast}(i, [\text{mtype} \mapsto \text{COMMIT}, \\
& \quad \text{mepoch} \mapsto \text{currentEpoch}[i], \\
& \quad \text{mindex} \mapsto \text{newCommittedIndex}, \\
& \quad \text{mcounter} \mapsto \text{history}[i][\text{newCommittedIndex}].\text{counter}]) \\
& \wedge \text{committedIndex}' = [\text{committedIndex} \text{ EXCEPT } ![i] = \text{committedIndex}[i] + 1] \\
& \wedge \text{UNCHANGED } \langle \text{serverVars}, \text{cluster}, \text{ceepochRecv}, \text{ackRecv}, \text{ackldRecv}, \text{ackIndex}, \text{currentCounter}, \\
& \quad \text{sendCounter}, \text{initialHistory}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle \\
\text{In phase } f32, \text{ follower receives } \text{COMMIT} \text{ and commits transaction.} \\
\text{FollowerBroadcast2}(i, j) & \triangleq \\
& \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{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}\})]
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{Discard}(j, i) \\
\vee & \text{ It may happen when the server is a new follower who joined in the cluster,} \\
& \text{and it misses the corresponding } \text{PROPOSE}. \\
& \wedge \neg \text{infoOk} \\
& \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{CEPOCH}, \\
& \quad \text{mepoch} \mapsto \text{currentEpoch}[i]]) \\
& \wedge \text{UNCHANGED } \text{commitIndex} \\
\vee & \text{ stale } \text{COMMIT} - \text{discard} \\
& \wedge \neg \text{replyOk} \\
& \wedge \text{Discard}(j, i) \\
& \wedge \text{UNCHANGED } \text{commitIndex} \\
& \wedge \text{UNCHANGED } \langle \text{state}, \text{currentEpoch}, \text{leaderEpoch}, \text{history}, \text{leaderOracle}, \\
& \quad \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

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.

In phase *l33*, upon receiving *CEPOCH*, leader *l* proposes back *NEWEPOCH* and *NEWLEADER*.
 $\text{LeaderHandleCEPOCHinPhase3}(i, j) \triangleq$

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

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

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

$\text{LeaderHandleACKLDinPhase3}(i, j) \triangleq$

$$\begin{aligned}
& \wedge \text{state}[i] = \text{Leader} \\
& \wedge \text{msgs}[j][i] \neq \langle \rangle \\
& \wedge \text{msgs}[j][i][1].\text{mtype} = \text{ACKLD} \\
& \wedge \text{LET } \text{msg} \triangleq \text{msgs}[j][i][1] \\
& \quad \text{aimCommitIndex} \triangleq \text{Minimum}(\{\text{commitIndex}[i], \text{Len}(\text{msg.mhistory})\}) \\
& \quad \text{aimCommitCounter} \triangleq \text{IF } \text{aimCommitIndex} = 0 \text{ THEN } 0 \text{ ELSE } \text{history}[i][\text{aimCommitIndex}].\text{co} \\
& \quad \text{IN } \vee \wedge \text{currentEpoch}[i] = \text{msg.mepoch} \\
& \quad \quad \wedge \text{ackIndex}' = [\text{ackIndex} \text{ EXCEPT } ![i][j] = \text{Len}(\text{msg.mhistory})] \\
& \quad \quad \wedge \text{Reply}(i, j, [\text{mtype} \mapsto \text{COMMIT},
\end{aligned}$$

$$\begin{aligned}
& mepoch \mapsto currentEpoch[i], \\
& mindex \mapsto aimCommitIndex, \\
& mcounter \mapsto aimCommitCounter]) \\
& \vee \wedge currentEpoch[i] \neq msg.mepoch \\
& \wedge Discard(j, i) \\
& \wedge UNCHANGED ackIndex \\
& \wedge cluster' = [cluster \text{ EXCEPT } ![i] = \text{IF } j \in cluster[i] \text{ THEN } cluster[i] \\
& \hspace{15em} \text{ELSE } cluster[i] \cup \{j\}] \\
& \wedge UNCHANGED \langle serverVars, cepochRecv, ackRecv, ackldRecv, currentCounter, sendCounter, \\
& \hspace{10em} initialHistory, committedIndex, tempVars, cepochSent, recoveryVars, proposalMsgsLo,
\end{aligned}$$

Let me suppose three 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.

$$\begin{aligned}
BecomeFollower(i) & \triangleq \\
& \wedge state[i] \neq Follower \\
& \wedge \exists j \in Server \setminus \{i\} : \wedge msgs[j][i] \neq \langle \rangle \\
& \hspace{1em} \wedge msgs[j][i][1].mtype \neq RECOVERYREQUEST \\
& \hspace{1em} \wedge msgs[j][i][1].mtype \neq RECOVERYRESPONSE \\
& \wedge LET msg \triangleq msgs[j][i][1] \\
& \hspace{1em} IN \wedge NullPoint \in cepochRecv[i] \\
& \hspace{1em} \wedge Maximum(\{currentEpoch[i], leaderEpoch[i]\}) < msg.mepoch \\
& \hspace{1em} \wedge \vee msg.mtype = NEWEPOCH \\
& \hspace{2em} \vee msg.mtype = NEWLEADER \\
& \hspace{2em} \vee msg.mtype = COMMITLD \\
& \hspace{2em} \vee msg.mtype = PROPOSE \\
& \hspace{2em} \vee msg.mtype = COMMIT \\
& \wedge state' = [state \text{ EXCEPT } ![i] = Follower] \\
& \wedge currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch] \\
& \wedge leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j] \\
& \wedge Reply(i, j, [mtype \mapsto CEPOCH, \\
& \hspace{10em} mepoch \mapsto currentEpoch[i]]) \\
& \hspace{10em} \text{Here we should not use } Discard. \\
& \wedge UNCHANGED \langle leaderEpoch, history, commitIndex, leaderVars, tempVars, cepochSent, recoveryVars,
\end{aligned}$$

$$\begin{aligned}
DiscardStaleMessage(i) & \triangleq \\
& \wedge \exists j \in Server \setminus \{i\} : \wedge msgs[j][i] \neq \langle \rangle \\
& \hspace{1em} \wedge msgs[j][i][1].mtype \neq RECOVERYREQUEST \\
& \hspace{1em} \wedge msgs[j][i][1].mtype \neq RECOVERYRESPONSE \\
& \wedge LET msg \triangleq msgs[j][i][1] \\
& \hspace{1em} IN \vee \wedge state[i] = Follower
\end{aligned}$$

$$\begin{aligned}
& \wedge \quad \vee \text{ msg.mepoch} < \text{currentEpoch}[i] \setminus * \text{ Discussed before.} \\
& \quad \vee \text{ msg.mtype} = \text{CEPOCH} \\
& \quad \vee \text{ msg.mtype} = \text{ACKE} \\
& \quad \vee \text{ msg.mtype} = \text{ACKLD} \\
& \quad \vee \text{ msg.mtype} = \text{ACK} \\
& \vee \wedge \text{ state}[i] \neq \text{Follower} \\
& \quad \wedge \text{ msg.mtype} \neq \text{CEPOCH} \\
& \quad \wedge \vee \wedge \text{ state}[i] = \text{ProspectiveLeader} \\
& \quad \quad \wedge \vee \text{ msg.mtype} = \text{ACK} \\
& \quad \quad \vee \wedge \text{ msg.mepoch} \leq \text{Maximum}(\{\text{currentEpoch}[i], \text{leaderEpoch}[i]\}) \\
& \quad \quad \quad \wedge \vee \text{ msg.mtype} = \text{NEWPOCH} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{NEWLEADER} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{COMMITLD} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{PROPOSE} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{COMMIT} \\
& \vee \wedge \text{ state}[i] = \text{Leader} \\
& \quad \wedge \vee \text{ msg.mtype} = \text{ACKE} \\
& \quad \quad \vee \wedge \text{ msg.mepoch} \leq \text{currentEpoch}[i] \\
& \quad \quad \quad \wedge \vee \text{ msg.mtype} = \text{NEWPOCH} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{NEWLEADER} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{COMMITLD} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{PROPOSE} \\
& \quad \quad \quad \vee \text{ msg.mtype} = \text{COMMIT} \\
& \quad \wedge \text{ Discard}(j, i) \\
& \wedge \text{ UNCHANGED } \langle \text{serverVars}, \text{leaderVars}, \text{tempVars}, \text{ceepochSent}, \text{recoveryVars}, \text{proposalMsgsLog} \rangle
\end{aligned}$$

Defines how the variables may transition.

$\text{Next} \triangleq$

$$\begin{aligned}
& \vee \exists i \in \text{Server}, Q \in \text{Quorums} : \text{InitialElection}(i, Q) \\
& \vee \exists i \in \text{Server} : \text{Restart}(i) \\
& \vee \exists i \in \text{Server} : \text{RecoveryAfterRestart}(i) \\
& \vee \exists i, j \in \text{Server} : \text{HandleRecoveryRequest}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{HandleRecoveryResponse}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{FindCluster}(i) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderTimeout}(i, j) \\
& \vee \exists i \in \text{Server} : \text{FollowerTimeout}(i) \\
& \vee \exists i \in \text{Server} : \text{FollowerDiscovery1}(i) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleCEPOCH}(i, j) \\
& \vee \exists i \in \text{Server} : \text{LeaderDiscovery1}(i) \\
& \vee \exists i, j \in \text{Server} : \text{FollowerDiscovery2}(i, j) \\
& \vee \exists i, j \in \text{Server} : \text{LeaderHandleACKE}(i, j) \\
& \vee \exists i \in \text{Server} : \text{LeaderDiscovery2Sync1}(i) \\
& \vee \exists i, j \in \text{Server} : \text{FollowerSync1}(i, j)
\end{aligned}$$

$$\begin{aligned}
& \forall \exists i, j \in \text{Server} : \text{LeaderHandleACKLD}(i, j) \\
& \forall \exists i \in \text{Server} : \text{LeaderSync2}(i) \\
& \forall \exists i, j \in \text{Server} : \text{FollowerSync2}(i, j) \\
& \forall \exists i \in \text{Server}, v \in \text{Value} : \text{ClientRequest}(i, v) \\
& \forall \exists i \in \text{Server} : \text{LeaderBroadcast1}(i) \\
& \forall \exists i, j \in \text{Server} : \text{FollowerBroadcast1}(i, j) \\
& \forall \exists i, j \in \text{Server} : \text{LeaderHandleACK}(i, j) \\
& \forall \exists i \in \text{Server} : \text{LeaderAdvanceCommit}(i) \\
& \forall \exists i \in \text{Server} : \text{LeaderBroadcast2}(i) \\
& \forall \exists i, j \in \text{Server} : \text{FollowerBroadcast2}(i, j) \\
& \forall \exists i, j \in \text{Server} : \text{LeaderHandleCEPOCHinPhase3}(i, j) \\
& \forall \exists i, j \in \text{Server} : \text{LeaderHandleACKLDinPhase3}(i, j) \\
& \forall \exists i \in \text{Server} : \text{DiscardStaleMessage}(i) \\
& \forall \exists i \in \text{Server} : \text{BecomeFollower}(i)
\end{aligned}$$

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

Safety properties of Zab consensus algorithm

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

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

$$\begin{aligned}
& \wedge \vee \text{state}[i] = \text{Leader} \\
& \quad \vee \wedge \text{state}[i] = \text{ProspectiveLeader} \\
& \quad \quad \wedge \text{NullPoint} \in \text{ackRecv}[i] \quad \text{prospective leader determines its epoch after broadcasting NEWLE} \\
& \wedge \vee \text{state}[j] = \text{Leader} \\
& \quad \vee \wedge \text{state}[j] = \text{ProspectiveLeader} \\
& \quad \quad \wedge \text{NullPoint} \in \text{ackRecv}[j] \\
& \wedge \text{currentEpoch}[i] = \text{currentEpoch}[j] \\
& \Rightarrow i = j
\end{aligned}$$

Here, delivering means deliver some transaction from history to replica. We assume $\text{deliverIndex} = \text{commitIndex}$.

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

And we can express a transaction by two-tuple $\langle \text{epoch}, \text{counter} \rangle$ according to its uniqueness.

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

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

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

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

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

$Integrity \triangleq \forall i \in Server :$

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

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

$Agreement \triangleq \forall i, j \in Server :$

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

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

$TotalOrder \triangleq \forall i, j \in Server : commitIndex[i] \geq 2 \wedge commitIndex[j] \geq 2$

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

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

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

$$\begin{aligned}
& \wedge \text{equal}(\text{history}[j][\text{index2}], \text{tscNext}) \\
& \wedge \text{index2} > 1 \\
& \wedge \exists \text{index1} \in 1 \dots (\text{index2} - 1) : \text{equal}(\text{history}[j][\text{index1}], \text{tscPre})
\end{aligned}$$

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 .

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

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 .

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

- \ * Modification History
- \ * Last modified *Thu Apr 29 23:10:30 CST 2021* by Dell
- \ * Created Sat *Dec 05 13:32:08 CST 2020* by Dell