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

the maximum round of epoch (initially {0, 1, 2}) currently not used
CONSTANT *Epoches*

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

Return the minimum value from the set S
$Minimum(S) \triangleq$ IF $S = \{\}$ THEN -1
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 $QuorumsAssumption \triangleq \land \forall\, Q \in Quorums : Q \subseteq Server$
$\land \forall\, Q1,\ Q2 \in Quorums : Q1 \cap Q2 \neq \{\}$

$None \triangleq$ CHOOSE $v : v \notin Value$

$NullPoint \triangleq$ 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*(*f.p in paper*).
VARIABLE *currentEpoch*

The last new leader proposal the follower *acknowledged*(*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 repersenting 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 followers who has successfully sent *CEPOCH* to pleader in pleader.
VARIABLE *cepochRecv*

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* to represent the prefix entries of the history
that has applied to state machine, but we can tolerate that *applyIndex* = *commitIndex*.
This does not violate correctness.
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 *cepochSent*

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 $tempMaxLastEpoch$

VARIABLE $tempInitialHistory$

$serverVars \triangleq \langle state, currentEpoch, leaderEpoch, leaderOracle, history, commitIndex\rangle$
$leaderVars \triangleq \langle cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter, sendCounter, initialHistory, co$
$tempVars \triangleq \langle tempMaxEpoch, tempMaxLastEpoch, tempInitialHistory\rangle$

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

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

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

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/Pleader broadcasts a message to all other servers
$Broadcast(i, m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } ii = i \wedge ij \neq i \text{ THEN } Append(msgs[ii][ij], m)$
$\text{ELSE } msgs[ii][ij]]]$

Combination of *Send* and *Discard* $-$ discard head of $msgs[j][i]$ and add m into $msgs[i][j]$
$Reply(i, j, m) \triangleq msgs' = [msgs \text{ EXCEPT } ![j][i] = Tail(msgs[j][i]),$
$![i][j] = Append(msgs[i][j], m)]$

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

Define initial values for all variables
$$
\begin{aligned}
Init \triangleq\ & \wedge state && = [s \in Server \mapsto Follower] \\
& \wedge currentEpoch && = [s \in Server \mapsto 0] \\
& \wedge leaderEpoch && = [s \in Server \mapsto 0] \\
& \wedge leaderOracle && = [s \in Server \mapsto NullPoint] \\
& \wedge history && = [s \in Server \mapsto \langle\rangle] \\
& \wedge msgs && = [i \in Server \mapsto [j \in Server \mapsto \langle\rangle]] \\
& \wedge cepochRecv && = [s \in Server \mapsto \{\}] \\
& \wedge ackeRecv && = [s \in Server \mapsto \{\}] \\
& \wedge ackldRecv && = [s \in Server \mapsto \{\}] \\
& \wedge ackIndex && = [i \in Server \mapsto [j \in Server \mapsto 0]] \\
& \wedge currentCounter && = [s \in Server \mapsto 0] \\
& \wedge sendCounter && = [s \in Server \mapsto 0] \\
& \wedge commitIndex && = [s \in Server \mapsto 0] \\
& \wedge committedIndex && = [s \in Server \mapsto 0] \\
& \wedge initialHistory && = [s \in Server \mapsto \langle\rangle]
\end{aligned}
$$

$\land$ *cepochSent* $= [s \in Server \mapsto \text{FALSE}]$
$\land$ *tempMaxEpoch* $= [s \in Server \mapsto 0]$
$\land$ *tempMaxLastEpoch* $= [s \in Server \mapsto 0]$
$\land$ *tempInitialHistory* $= [s \in Server \mapsto \langle\rangle]$

A server becomes pleader and a quorum servers knows that.

Election$(i,\ Q) \triangleq$
$\land\ i \in Q$
$\land$ *state*$'$ $= [s \in Server \mapsto \text{IF } s = i \text{ THEN } ProspectiveLeader$
$\text{ELSE IF } s \in Q \text{ THEN } Follower$
$\text{ELSE } state[s]]$
$\land$ *cepochRecv*$'$ $= [cepochRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land$ *ackeRecv*$'$ $= [ackeRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land$ *ackldRecv*$'$ $= [ackldRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land$ *ackIndex*$'$ $= [ii \in Server \mapsto [ij \in Server \mapsto$
$\text{IF } ii = i \text{ THEN } 0$
$\text{ELSE } ackIndex[ii][ij]]]$
$\land$ *committedIndex*$'$ $= [committedIndex \text{ EXCEPT } ![i] = 0]$
$\land$ *initialHistory*$'$ $= [initialHistory \text{ EXCEPT } ![i] = \langle\rangle]$
$\land$ *tempMaxEpoch*$'$ $= [tempMaxEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
$\land$ *tempMaxLastEpoch*$'$ $= [tempMaxLastEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
$\land$ *tempInitialHistory*$'$ $= [tempInitialHistory \text{ EXCEPT } ![i] = history[i]]$
$\land$ *leaderOracle*$'$ $= [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } i$
$\text{ELSE } leaderOracle[s]]$
$\land$ *leaderEpoch*$'$ $= [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } currentEpoch[s]$
$\text{ELSE } leaderEpoch[s]]$
$\land$ *cepochSent*$'$ $= [s \in Server \mapsto \text{IF } s \in Q \text{ THEN FALSE}$
$\text{ELSE } cepochSent[s]]$
$\land$ *msgs*$'$ $= [ii \in Server \mapsto [ij \in Server \mapsto$
$\text{IF } ii \in Q \land ij \in Q \text{ THEN } \langle\rangle$
$\text{ELSE } msgs[ii][ij]]]$
$\land \text{UNCHANGED } \langle currentEpoch,\ history,\ commitIndex,\ currentCounter,\ sendCounter\rangle$

A server halts and restarts.

Restart$(i) \triangleq$
$\land\ state' = [state \text{ EXCEPT } ![i] = Follower]$
$\land\ leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = NullPoint]$
$\land\ cepochSent' = [cepochSent \text{ EXCEPT } ![i] = \text{FALSE}]$
$\land \text{UNCHANGED } \langle currentEpoch,\ leaderEpoch,\ history,\ commitIndex,\ leaderVars,\ tempVars,\ msgs\rangle$

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

FollowerDiscovery$1(i) \triangleq$
$\land\ state[i] = Follower$
$\land\ leaderOracle[i] \neq NullPoint$

$\land \neg cepochSent[i]$
$\land$ LET $leader \triangleq leaderOracle[i]$
IN $Send(i, leader, [mtype \mapsto CEPOCH, mepoch \mapsto currentEpoch[i]])$
$\land cepochSent' = [cepochSent$ EXCEPT $![i] =$ TRUE$]$
$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars\rangle$

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.

$LeaderHandleCEPOCH(i, j) \triangleq$
$\land state[i] = ProspectiveLeader$
$\land msgs[j][i] \neq \langle\rangle$
$\land msgs[j][i][1].mtype = CEPOCH$
$\land \lor$ redundant message - just discard
$\land j \in cepochRecv[i]$
$\land$ UNCHANGED $\langle tempMaxEpoch, cepochRecv\rangle$
$\lor$ new message - modify $tempMaxEpoch$ and $cepochRecv$
$\land j \notin cepochRecv[i]$
$\land$ LET $newEpoch \triangleq Maximum(\{tempMaxEpoch[i], msgs[j][i][1].mepoch\})$
IN $tempMaxEpoch' = [tempMaxEpoch$ EXCEPT $![i] = newEpoch]$
$\land cepochRecv' = [cepochRecv$ EXCEPT $![i] = cepochRecv[i] \cup \{j\}]$
$\land Discard(j, i)$
$\land$ UNCHANGED $\langle serverVars, ackeRecv, ackldRecv, ackIndex, currentCounter, sendCounter,$
$initialHistory, committedIndex, cepochSent, tempMaxLastEpoch, tempInitialHistory\rangle$

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 expoch. So here I just change $leaderEpoch$, and use it in handling ACK-E.

$LeaderDiscovery1(i) \triangleq$
$\land state[i] = ProspectiveLeader$
$\land cepochRecv[i] \in Quorums$
$\land leaderEpoch' = [leaderEpoch$ EXCEPT $![i] = tempMaxEpoch[i] + 1]$
$\land cepochRecv' = [cepochRecv$ EXCEPT $![i] = \{\}]$
$\land Broadcast(i, [mtype \mapsto NEWEPOCH, mepoch \mapsto leaderEpoch'[i]])$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderOracle, history, ackeRecv, ackldRecv, ackIndex,$
$currentCounter, sendCounter, initialHistory, commitIndex, committedIndex, cepochS$

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.

$FollowerDiscovery2(i, j) \triangleq$
$\land state[i] = Follower$
$\land msgs[j][i] \neq \langle\rangle$
$\land msgs[j][i][1].mtype = NEWEPOCH$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new $NEWEPOCH -$ accept and reply
$\land currentEpoch[i] \leq msg.mepoch$ Here use $\leq$, because one follower may send $CEPOCH$ more then o

$\land\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\land\ leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j]$
$\land\ Reply(i, j, [mtype \mapsto ACKE,$
$mepoch \mapsto msg.mepoch,$
$mlastEpoch \mapsto leaderEpoch[i],$
$mhf \mapsto history[i]])$
$\lor$ stale *NEWEPOCH − diacard*
$\land\ currentEpoch[i] > msg.mepoch$
$\land\ Discard(j, i)$
$\land\ \text{UNCHANGED } \langle currentEpoch, leaderOracle \rangle$
$\land\ \text{UNCHANGED } \langle state, leaderEpoch, history, leaderVars, commitIndex, cepochSent, tempVars \rangle$

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.

$LeaderHandleACKE(i, j) \triangleq$
$\land\ state[i] = ProspectiveLeader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACKE$
$\land\ \text{LET } msg \triangleq msgs[j][i][1]$
$infoOk \triangleq \lor\ msg.mlastEpoch > tempMaxLastEpoch[i]$
$\lor\ \land\ msg.mlastEpoch = tempMaxLastEpoch[i]$
$\land\ \lor\ LastZxid(msg.mhf)[1] > LastZxid(tempInitialHistory[i])[1]$
$\lor\ \land\ LastZxid(msg.mhf)[1] = LastZxid(tempInitialHistory[i])[1]$
$\land\ LastZxid(msg.mhf)[2] \geq LastZxid(tempInitialHistory[i])[2]$
IN $\lor\ \land\ leaderEpoch[i] = msg.mepoch$
$\land\ \lor\ \land\ infoOk$
$\land\ tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = msg.mlastEpoch]$
$\land\ tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = msg.mhf]$
$\lor\ \land\ \neg infoOk$
$\land\ \text{UNCHANGED } \langle tempMaxLastEpoch, tempInitialHistory \rangle$
$\land\ ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackeRecv[i] \text{ THEN } ackeRecv[i] \cup \{j\}$
$\text{ELSE } ackeRecv[i]]$
$\lor\ \land\ leaderEpoch[i] \neq msg.mepoch$
$\land\ \text{UNCHANGED } \langle tempMaxLastEpoch, tempInitialHistory, ackeRecv \rangle$
$\land\ Discard(j, i)$
$\land\ \text{UNCHANGED } \langle serverVars, cepochRecv, ackldRecv, ackIndex, currentCounter,$
$sendCounter, initialHistory, committedIndex, cepochSent, tempMaxEpoch \rangle$

$LeaderDiscovery2Sync1(i) \triangleq$
$\land\ state[i] = ProspectiveLeader$
$\land\ ackeRecv[i] \in Quorums$
$\land\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = leaderEpoch[i]]$
$\land\ history' = [history \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\land\ initialHistory' = [initialHistory \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\land\ commitIndex' = [commitIndex \text{ EXCEPT } ![i] = 0]$

$\land$ $ackeRecv'$ $= [ackeRecv$ EXCEPT $![i] = \{\}]$
$\land$ $ackIndex'$ $= [ackIndex$ EXCEPT $![i] = Len(tempInitialHistory[i])]$
until now, $phase1(Discovery)$ ends
$\land$ $Broadcast(i, [mtype \mapsto NEWLEADER,$
$mepoch \mapsto currentEpoch[i],$
$minitialHistory \mapsto history'[i]])$
$\land$ UNCHANGED $\langle state, leaderEpoch, leaderOracle, cepochRecv, ackldRecv,$
$currentCounter, sendCounter, committedIndex, cepochSent, tempVars\rangle$

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

$FollowerSync1(i, j) \triangleq$
$\land$ $state[i] = Follower$
$\land$ $msgs[j][i] \neq \langle\rangle$
$\land$ $msgs[j][i][1].mtype = NEWLEADER$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new $NEWLEADER-$ accept and reply
$\land$ $currentEpoch[i] \leq msg.mepoch$
$\land$ $currentEpoch' = [currentEpoch$ EXCEPT $![i] = msg.mepoch]$
$\land$ $leaderEpoch' = [leaderEpoch$ EXCEPT $![i] = msg.mepoch]$
$\land$ $leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\land$ $history' = [history$ EXCEPT $![i] = msg.minitialHistory]$
$\land$ $commitIndex' = [commitIndex$ EXCEPT $![i] = 0]$
$\land$ $Reply(i, j, [mtype \mapsto ACKLD,$
$mepoch \mapsto msg.mepoch,$
$mhistory \mapsto msg.minitialHistory])$
$\lor$ stale $NEWLEADER-$ discard
$\land$ $currentEpoch[i] > msg.mepoch$
$\land$ $Discard(j, i)$
$\land$ UNCHANGED $\langle currentEpoch, leaderEpoch, leaderOracle, history, commitIndex\rangle$
$\land$ UNCHANGED $\langle state, leaderVars, tempVars, cepochSent\rangle$

In phase $l22$, pleader receives ACK-LD from a quorum of followers, and sends $COMMIT$-LD to followers.
$LeaderHandleACKLD(i, j) \triangleq$
$\land$ $state[i] = ProspectiveLeader$
$\land$ $msgs[j][i] \neq \langle\rangle$
$\land$ $msgs[j][i][1].mtype = ACKLD$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new ACK-LD - accept
$\land$ $currentEpoch[i] = msg.mepoch$
$\land$ $ackIndex' = [ackIndex$ EXCEPT $![i][j] = Len(initialHistory[i])]$
$\land$ $ackldRecv' = [ackldRecv$ EXCEPT $![i] =$ IF $j \notin ackldRecv[i]$ THEN $ackldRecv[i] \cup \{j\}$
ELSE $ackldRecv[i]]$
$\lor$ stale ACK-LD - impossible

$\wedge\ currentEpoch[i] \neq msg.mepoch$
$\wedge$ UNCHANGED $\langle ackldRecv,\ ackIndex\rangle$
$\wedge\ Discard(j,\ i)$
$\wedge$ UNCHANGED $\langle serverVars,\ cepochRecv,\ ackeRecv,\ currentCounter,$
$sendCounter,\ initialHistory,\ committedIndex,\ tempVars,\ cepochSent\rangle$

$LeaderSync2(i) \triangleq$
$\wedge\ state[i] = ProspectiveLeader$
$\wedge\ ackldRecv[i] \in Quorums$
$\wedge\ commitIndex' = [commitIndex$ EXCEPT $![i] = Len(history[i])]$
$\wedge\ committedIndex' = [committedIndex$ EXCEPT $![i] = Len(history[i])]$
$\wedge\ state' = [state$ EXCEPT $![i] = Leader]$
$\wedge\ currentCounter' = [currentCounter$ EXCEPT $![i] = 0]$
$\wedge\ sendCounter' = [sendCounter$ EXCEPT $![i] = 0]$
$\wedge\ ackldRecv' = [ackldRecv$ EXCEPT $![i] = \{\}]$
$\wedge\ Broadcast(i, [mtype \mapsto COMMITLD,$
$mepoch \mapsto currentEpoch[i]])$
$\wedge\ $ UNCHANGED $\langle currentEpoch,\ leaderEpoch,\ leaderOracle,\ history,\ cepochRecv,$
$ackeRecv,\ ackIndex,\ initialHistory,\ tempVars,\ cepochSent\rangle$

In phase $f22$, follower receives $COMMIT$-LD and submits all unprocessed transaction.
$FollowerSync2(i,\ j) \triangleq$
$\wedge\ state[i] = Follower$
$\wedge\ msgs[j][i] \neq \langle\rangle$
$\wedge\ msgs[j][i][1].mtype = COMMITLD$
$\wedge$ LET $msg \triangleq msgs[j][i][1]$
IN $\vee$ new $COMMIT$-LD - commit all transactions in initial history
$\wedge\ currentEpoch[i] = msg.mepoch$
$\wedge\ commitIndex' = [commitIndex$ EXCEPT $![i] = Len(history[i])]$
$\wedge\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\vee$ stale $COMMIT$-LD - discard
$\wedge\ currentEpoch[i] \neq msg.mepoch$
$\wedge$ UNCHANGED $\langle commitIndex,\ leaderOracle\rangle$
$\wedge\ Discard(j,\ i)$
$\wedge$ UNCHANGED $\langle state,\ currentEpoch,\ leaderOracle,\ history,\ leaderVars,\ tempVars,\ cepochSent\rangle$

In phase $l31$, leader receives client request and broadcasts $PROPOSE$.
$ClientRequest(i,\ v) \triangleq$
$\wedge\ state[i] = Leader$
$\wedge\ currentCounter' = [currentCounter$ EXCEPT $![i] = currentCounter[i] + 1]$
$\wedge$ LET $newTransaction \triangleq [epoch \mapsto currentEpoch[i],$
$counter \mapsto currentCounter'[i],$
$value \mapsto v]$
IN $\wedge\ history' = [history$ EXCEPT $![i] = Append(history[i],\ newTransaction)]$
$\wedge\ ackIndex' = [ackIndex$ EXCEPT $![i] = Len(history'[i])]$

$\land$ UNCHANGED $\langle msgs, state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, cepochRecv, ackeRecv, ackldRecv, sendCounter, initialHistory, committedIndex, tempVars, cepoch$

$LeaderBroadcast1(i) \triangleq$
$\land\ state[i] = Leader$
$\land\ sendCounter[i] < currentCounter[i]$
$\land$ LET $toBeSentCounter \triangleq sendCounter[i] + 1$
$toBeSentIndex \triangleq Len(initialHistory[i]) + toBeSentCounter$
$toBeSentEntry \triangleq history[i][toBeSentIndex]$
IN $\land\ Broadcast(i, [mtype \mapsto PROPOSE,$
$mepoch \mapsto currentEpoch[i],$
$mproposal \mapsto toBeSentEntry])$
$\land\ sendCounter' = [sendCounter$ EXCEPT $![i] = toBeSentCounter]$
$\land$ UNCHANGED $\langle serverVars, cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter, initialHistory, committedIndex, tempVars, cepochSent\rangle$

In phase $f31$, follower accepts proposal and append it to history.
$FollowerBroadcast1(i, j) \triangleq$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = PROPOSE$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ It should be that $msg.mproposal.counter = 1 \lor msg.mrpoposal.counter = history[Len(history)].counter + 1$
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ history' = [history$ EXCEPT $![i] = Append(history[i], msg.mproposal)]$
$\land\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\land\ Reply(i, j, [mtype \mapsto ACK,$
$mepoch \mapsto currentEpoch[i],$
$mindex \mapsto Len(history'[i])])$
$\lor$ If happens, $\neq$ must be $>$, namely a stale leader sends it.
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle history, leaderOracle\rangle$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderEpoch, commitIndex, leaderVars, tempVars, cepochSent\rangle$

In phase $l32$, leader receives ack from a quorum of followers to a certain proposal, and commits the proposal.
$LeaderHandleACK(i, j) \triangleq$
$\land\ state[i] = Leader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACK$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ There should be $ackIndex[i][j] + 1 \triangleq msg.mindex$
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ ackIndex' = [ackIndex$ EXCEPT $![i][j] = Maximum(\{ackIndex[i][j], msg.mindex\})]$
$\lor$ If happens, $\neq$ must be $>$, namely a stale follower sends it.

```
                    ∧ currentEpoch[i] ≠ msg.mepoch
                    ∧ UNCHANGED ackIndex
          ∧ Discard(j, i)
          ∧ UNCHANGED ⟨serverVars, cepochRecv, ackeRecv, ackldRecv, currentCounter,
                       sendCounter, initialHistory, committedIndex, tempVars, cepochSent⟩

LeaderAdvanceCommit(i) ≜
          ∧ state[i] = Leader
          ∧ commitIndex[i] < Len(history[i])
          ∧ LET Agree(index)     ≜ {i} ∪ {k ∈ Server : ackIndex[i][k] ≥ index}
                agreeIndexes     ≜ {index ∈ (commitIndex[i] + 1) .. Len(history[i]) : Agree(index) ∈ Quorum
                newCommitIndex   ≜ IF agreeIndexes ≠ {} THEN Maximum(agreeIndexes)
                                                        ELSE commitIndex[i]
            IN  commitIndex' = [commitIndex EXCEPT ![i] = newCommitIndex]
          ∧ UNCHANGED ⟨state, currentEpoch, leaderEpoch, leaderOracle, history,
                       msgs, leaderVars, tempVars, cepochSent⟩

LeaderBroadcast2(i) ≜
          ∧ state[i] = Leader
          ∧ committedIndex[i] < commitIndex[i]
          ∧ LET newCommittedIndex ≜ committedIndex[i] + 1
            IN  ∧ Broadcast(i, [mtype    ↦ COMMIT,
                                mepoch   ↦ currentEpoch[i],
                                mindex   ↦ newCommittedIndex,
                                mcounter ↦ history[newCommittedIndex].counter])
                ∧ committedIndex' = [committedIndex EXCEPT ![i] = committedIndex[i] + 1]
          ∧ UNCHANGED ⟨serverVars, cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter,
                       sendCounter, initialHistory, tempVars, cepochSent⟩

 In phase f32, follower receives COMMIT and commits transaction.
FollowerBroadcast2(i, j) ≜
          ∧ state[i] = Follower
          ∧ msgs[j][i] ≠ ⟨⟩
          ∧ msgs[j][i][1].mtype = COMMIT
          ∧ LET msg ≜ msgs[j][i][1]
            IN  ∨  new COMMIT − commit transaction in history
                   ∧ currentEpoch[i] = msg.mepoch
                   ∧ commitIndex'  = [commitIndex  EXCEPT ![i] = Maximum({commitIndex[i], msg.mindex}
                   ∧ leaderOracle' = [leaderOracle EXCEPT ![i]  = j]
                ∨  stale COMMIT − discard
                   ∧ currentEpoch[i] ≠ msg.mepoch
                   ∧ UNCHANGED ⟨commitIndex, leaderOracle⟩
          ∧ Discard(j, i)
          ∧ UNCHANGED ⟨state, currentEpoch, leaderEpoch, history,
                       leaderVars, tempVars, cepochSent⟩
```

There may be two ways to make sure all followers as up-to-date as the leader.
*way*1: choose *Send* not *Broadcast* when leader is going to send *PROPOSE* and *COMMIT*.
*way*2: 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 *way*2, which I need not to rewrite *PROPOSE* and *COMMIT*, but need to modify the code when follower receives *NEWLEADER* and *COMMIT*.

In phase *l*33, upon receiving *CEPOCH*, leader l proposes back *NEWEPOCH* and *NEWLEADER*.

$$
\begin{array}{l}
LeaderHandleCEPOCHinPhase3(i,\ j) \triangleq \\
\quad \land\ state[i] = Leader \\
\quad \land\ msgs[j][i] \neq \langle\rangle \\
\quad \land\ msgs[j][i][1].mtype = CEPOCH \\
\quad \land\ \text{LET}\ msg \triangleq msgs[j][i][1] \\
\quad\ \ \text{IN}\quad \lor\ \land\ currentEpoch[i] \geq msg.mepoch \\
\qquad\qquad\ \ \land\ Reply2(i,\ j,\ [mtype\ \ \mapsto NEWEPOCH, \\
\qquad\qquad\qquad\qquad\qquad\ \ mepoch \mapsto currentEpoch[i]], \\
\qquad\qquad\qquad\qquad\qquad\ [mtype\ \ \mapsto NEWLEADER, \\
\qquad\qquad\qquad\qquad\qquad\ \ mepoch\ \ \mapsto currentEpoch[i], \\
\qquad\qquad\qquad\qquad\qquad\ \ minitialHistory \mapsto history[i]]) \\
\qquad\qquad \lor\ \land\ currentEpoch[i] < msg.mepoch \\
\qquad\qquad\ \ \land\ \text{UNCHANGED}\ msgs \\
\quad \land\ \text{UNCHANGED}\ \langle serverVars,\ leaderVars,\ tempVars,\ cepochSent\rangle
\end{array}
$$

In phase *l*34, upon receiving ack from f of the *NEWLEADER*, it sends a commit message to f. Leader l also makes $Q := Q \cup \{f\}$.

$$
\begin{array}{l}
LeaderHandleACKLDinPhase3(i,\ j) \triangleq \\
\quad \land\ state[i] = Leader \\
\quad \land\ msgs[j][i] \neq \langle\rangle \\
\quad \land\ msgs[j][i][1].mtype = ACKLD \\
\quad \land\ \text{LET}\ msg \triangleq msgs[j][i][1] \\
\qquad\ \ aimCommitIndex \triangleq Minimum(\{commitIndex[i],\ Len(msg.mhistory)\}) \\
\quad\ \ \text{IN}\quad \lor\ \land\ currentEpoch[i] = msg.mepoch \\
\qquad\qquad\ \ \land\ ackIndex' = [ackIndex\ \text{EXCEPT}\ ![i][j] = Len(msg.mhistory)] \\
\qquad\qquad\ \ \land\ Reply(i,\ j,\ [mtype\ \ \ \mapsto COMMIT, \\
\qquad\qquad\qquad\qquad\qquad\ \ mepoch\ \ \mapsto currentEpoch[i], \\
\qquad\qquad\qquad\qquad\qquad\ \ mindex\ \ \mapsto aimCommitIndex, \\
\qquad\qquad\qquad\qquad\qquad\ \ mcounter \mapsto history[aimCommitIndex].counter]) \\
\qquad\qquad \lor\ \land\ currentEpoch[i] \neq msg.mepoch \\
\qquad\qquad\ \ \land\ Discard(j,\ i) \\
\qquad\qquad\ \ \land\ \text{UNCHANGED}\ \langle ackIndex\rangle \\
\quad \land\ \text{UNCHANGED}\ \langle serverVars,\ cepochRecv,\ ackeRecv,\ ackldRecv,\ currentCounter,\ sendCounter, \\
\qquad\qquad\qquad\qquad initialHistory,\ committedIndex,\ tempVars,\ cepochSent\rangle
\end{array}
$$

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.

$$
\begin{array}{l}
BecomeFollower(i) \triangleq \\
\quad \land \exists j \in Server \setminus \{i\} : \land msgs[j][i] \neq \langle\rangle \\
\qquad\qquad \land \text{LET } msg \triangleq msgs[j][i][1] \\
\qquad\qquad\quad \text{IN} \quad \land currentEpoch[i] < msg.mepoch \\
\qquad\qquad\qquad\qquad \land \lor msg.mtype = NEWEPOCH \\
\qquad\qquad\qquad\qquad\quad \lor msg.mtype = NEWLEADER \\
\qquad\qquad\qquad\qquad\quad \lor msg.mtype = COMMITLD \\
\qquad\qquad\qquad\qquad\quad \lor msg.mtype = PROPOSE \\
\qquad\qquad\qquad\qquad\quad \lor msg.mtype = COMMIT \\
\qquad\qquad\qquad\qquad \land state' = [state \text{ EXCEPT } ![i] = Follower] \\
\qquad\qquad\qquad\qquad \land currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch] \\
\qquad\qquad\qquad\qquad \land leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j] \\
\qquad\qquad\qquad\qquad \text{Here we should not use } Discard. \\
\quad \land \text{UNCHANGED } \langle leaderEpoch, history, commitIndex, msgs, leaderVars, tempVars, cepochSent\rangle
\end{array}
$$

$$
\begin{array}{l}
DiscardStaleMessage(i) \triangleq \\
\quad \land \exists j \in Server \setminus \{i\} : \land msgs[j][i] \neq \langle\rangle \\
\qquad\qquad \land \text{LET } msg \triangleq msgs[j][i][1] \\
\qquad\qquad\quad \text{IN} \quad \lor \land state[i] = Follower \\
\qquad\qquad\qquad\qquad\quad \land \lor msg.mepoch < currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad \lor msg.mtype = CEPOCH \\
\qquad\qquad\qquad\qquad\qquad \lor msg.mtype = ACKE \\
\qquad\qquad\qquad\qquad\qquad \lor msg.mtype = ACKLD \\
\qquad\qquad\qquad\qquad\qquad \lor msg.mtype = ACK \\
\qquad\qquad\qquad\qquad \lor \land state[i] = Leader \\
\qquad\qquad\qquad\qquad\quad \land msg.mtype \neq CEPOCH \\
\qquad\qquad\qquad\qquad\quad \land \lor msg.mepoch < currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad \lor msg.mtype = ACKE \text{ response of } NEWEPOCH \\
\qquad\qquad\qquad\qquad \lor \land state[i] = ProspectiveLeader \\
\qquad\qquad\qquad\qquad\quad \land msg.mtype \neq CEPOCH \\
\qquad\qquad\qquad\qquad\quad \land \lor msg.mepoch < currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad \lor msg.mtype = ACK \\
\qquad\qquad\qquad\qquad \land Discard(j, i) \\
\quad \land \text{UNCHANGED } \langle serverVars, leaderVars, tempVars, cepochSent\rangle
\end{array}
$$

Defines how the variables may transition.

$$
\begin{aligned}
Next \triangleq\ & \lor \exists\, i \in Server : && Restart(i) \\
& \lor \exists\, i \in Server,\ Q \in Quorums : && Election(i,\ Q) \\
& \lor \exists\, i \in Server : && FollowerDiscovery1(i) \\
& \lor \exists\, i, j \in Server : && LeaderHandleCEPOCH(i,\ j) \\
& \lor \exists\, i \in Server : && LeaderDiscovery1(i) \\
& \lor \exists\, i, j \in Server : && FollowerDiscovery2(i,\ j) \\
& \lor \exists\, i, j \in Server : && LeaderHandleACKE(i,\ j) \\
& \lor \exists\, i \in Server : && LeaderDiscovery2Sync1(i) \\
& \lor \exists\, i, j \in Server : && FollowerSync1(i,\ j) \\
& \lor \exists\, i, j \in Server : && LeaderHandleACKLD(i,\ j) \\
& \lor \exists\, i \in Server : && LeaderSync2(i) \\
& \lor \exists\, i, j \in Server : && FollowerSync2(i,\ j) \\
& \lor \exists\, i \in Server,\ v \in Value : && ClientRequest(i,\ v) \\
& \lor \exists\, i \in Server : && LeaderBroadcast1(i) \\
& \lor \exists\, i, j \in Server : && FollowerBroadcast1(i,\ j) \\
& \lor \exists\, i, j \in Server : && LeaderHandleACK(i,\ j) \\
& \lor \exists\, i \in Server : && LeaderAdvanceCommit(i) \\
& \lor \exists\, i \in Server : && LeaderBroadcast2(i) \\
& \lor \exists\, i, j \in Server : && FollowerBroadcast2(i,\ j) \\
& \lor \exists\, i, j \in Server : && LeaderHandleCEPOCHinPhase3(i,\ j) \\
& \lor \exists\, i, j \in Server : && LeaderHandleACKLDinPhase3(i,\ j) \\
& \lor \exists\, i \in Server : && DiscardStaleMessage(i) \\
& \lor \exists\, i \in Server : && BecomeFollower(i)
\end{aligned}
$$

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

Defines some variants, safety propoties, and liveness propoties of zab consensus algorithm.

Safety properties

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

$$
\begin{aligned}
Consistency \triangleq\ & \exists\, i, j \in Server : \\
& \land state[i] = Leader \\
& \land state[j] = Leader \\
& \land currentEpoch[i] = currentEpoch[j] \\
& \Rightarrow i = j
\end{aligned}
$$

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

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

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

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

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.

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.

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.

Integrity $\triangleq$ $\forall\, l, f \in Server, msg \in msgs$:
$\land\ state[l] = Leader \land state[f] = Follower$
$\land\ msg.type = COMMIT \land msg \in histroy[f]$
$\Rightarrow\ msg \in history[l]$

$LivenessProperty1 \triangleq \forall\, i,\ j \in Server,\ msg \in msgs$: $(state[i] = Leader) \land (msg.type = COMMIT) \rightsquigarrow (msg \in history[j]) \land (state[j] = Follower)$

\ * Modification History
\ * Last modified *Thu Apr* 15 21:25:04 *CST* 2021 by Dell
\ * Created Sat *Dec* 05 13:32:08 *CST* 2020 by Dell