MODULE *TunableMongoDB_Repl*

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

constants and variables

CONSTANTS *Client*, *Server*, the set of clients and servers
Key, *Value*, the set of keys and values
Nil, model value, place holder
PtStop max physical time

VARIABLES *Primary*, Primary node
Secondary, secondary nodes
Oplog, *oplog*[*s*]: *oplog* at *server*[*s*]
Store, *store*[*s*]: data stored at *server*[*s*]
Ct, *Ct*[*s*]: cluster time at node *s*
Ot, *Ot*[*s*]: the last applied operation time at server *s*
ServerMsg, *ServerMsg*[*s*]: the channel of heartbeat *msgs* at server *s*
Pt, *Pt*[*s*]: physical time at server *s*
Cp, *Cp*[*s*]: majority commit point at server *s*
State, *State*[*s*]: the latest *Ot* of all servers that server *s* knows
CalState, *CalState*: sorted *State*[*Primary*]
CurrentTerm, *CurrentTerm*[*s*]: current election term at server *s*
→ updated in *update_position*, heartbeat and replicate
ReadyToServe, equal to 0 before any primary is elected
SyncSource *SyncSource*[*s*]: sync source of server node *s*

ASSUME $Cardinality(Client) \geq 1$ at least one clinet
ASSUME $Cardinality(Server) \geq 2$ at least one primary and one secondary
ASSUME $Cardinality(Key) \geq 1$ at least one object
ASSUME $Cardinality(Value) \geq 2$ at least two values to update

Helpers

$$HLCLt(x,\ y) \triangleq \text{IF } x.p < y.p \text{ THEN TRUE ELSE IF } x.p = y.p \text{ THEN IF } x.l < y.l \text{ THEN TRUE ELSE FALSE ELSE FALSE}$$

$HLCMin(x,\ y) \triangleq$ IF $HLCLt(x,\ y)$ THEN x ELSE y
$HLCMax(x,\ y) \triangleq$ IF $HLCLt(x,\ y)$ THEN y ELSE x
$HLCType \triangleq [p : Nat,\ l : Nat]$
$Min(x,\ y) \triangleq$ IF $x < y$ THEN x ELSE y
$Max(x,\ y) \triangleq$ IF $x > y$ THEN x ELSE y

$vars \triangleq \langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,\ ServerMsg,$
$Pt,\ Cp,\ CalState,\ State,$
$CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

RECURSIVE $CreateState(\_,\ \_)$ init state
$CreateState(len,\ seq) \triangleq$
IF $len = 0$ THEN seq
ELSE $CreateState(len - 1,\ Append(seq,\ [p \mapsto 0,\ l \mapsto 0]))$

$LogTerm(i,\ index) \triangleq$ IF $index = 0$ THEN 0 ELSE $Oplog[i][index].term$
$LastTerm(i) \triangleq CurrentTerm[i]$

Is node i ahead of node j
$NotBehind(i,\ j) \triangleq \lor\ LastTerm(i) > LastTerm(j)$
$\lor\ \land\ LastTerm(i) = LastTerm(j)$
$\land\ Len(Oplog[i]) \geq Len(Oplog[j])$

$IsMajority(servers) \triangleq Cardinality(servers) * 2 > Cardinality(Server)$

Return the maximum value from a set, or undefined if the set is empty.
$MaxVal(s) \triangleq$ CHOOSE $x \in s : \forall\, y \in s : x \geq y$

commit point
RECURSIVE $AddState(\_,\ \_,\ \_)$
$AddState(new,\ state,\ index) \triangleq$
IF $index = 1 \land HLCLt(new,\ state[1])$
THEN $\langle new\rangle \circ state$ less than the first
ELSE IF $index = Len(state) + 1$
THEN $state \circ \langle new\rangle$
ELSE IF $HLCLt(new,\ state[index])$
THEN $SubSeq(state,\ 1,\ index\ \ - 1) \circ \langle new\rangle \circ SubSeq(state,\ index,\ Len(state))$
ELSE $AddState(new,\ state,\ index + 1)$

RECURSIVE $RemoveState(\_,\ \_,\ \_)$
$RemoveState(old,\ state,\ index) \triangleq$
IF $state[index] = old$
THEN $SubSeq(state,\ 1,\ index - 1) \circ SubSeq(state,\ index + 1,\ Len(state))$
ELSE $RemoveState(old,\ state,\ index + 1)$

$AdvanceState(new,\ old,\ state) \triangleq AddState(new,\ RemoveState(old,\ state,\ 1),\ 1)$

clock

$MaxPt \triangleq$ LET $x \triangleq$ CHOOSE $s \in Server : \forall\, s1 \in Server \setminus \{s\} :$
$Pt[s] \geq Pt[s1]$IN $\quad Pt[x]$

$Tick(s) \triangleq Ct' =$ IF $Ct[s].p \geq Pt[s]$
THEN $[Ct$ EXCEPT $![s] = [p \mapsto @.p,\ l \mapsto @.l + 1]]$

$$\text{ELSE}\ \ [Ct\ \text{EXCEPT}\ ![s] = [p \mapsto Pt[s],\ l \mapsto 0]]$$

heartbeat

Only *Primary* node sends heartbeat once advance pt

$$\begin{array}{l}
BroadcastHeartbeat(s) \triangleq \\
\quad \text{LET}\ msg \triangleq [type \mapsto \text{“heartbeat”},\ s \mapsto s,\ aot \mapsto Ot[s], \\
\qquad\qquad ct \mapsto Ct[s],\ cp \quad \mapsto Cp[s],\ term \mapsto CurrentTerm[s]] \\
\quad \text{IN}\ \ ServerMsg' = [x \in Server \mapsto \text{IF}\ x = s\ \text{THEN}\ ServerMsg[x] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE}\ \ Append(ServerMsg[x],\ msg)]
\end{array}$$

Can node i sync from node j?

$$\begin{array}{l}
CanSyncFrom(i,\ j) \triangleq \\
\quad \land\ Len(Oplog[i]) < Len(Oplog[j]) \\
\quad \land\ LastTerm(i) = LogTerm(j,\ Len(Oplog[i]))
\end{array}$$

Oplog entries needed to replicate from j to i

$$\begin{array}{l}
ReplicateOplog(i,\ j) \triangleq \\
\quad \text{LET}\ len_i \triangleq Len(Oplog[i]) \\
\qquad len_j \triangleq Len(Oplog[j]) \\
\quad \text{IN}\ \ \text{IF}\ i \in Secondary \land len_i < len_j \\
\qquad\qquad\qquad\qquad \text{THEN}\ SubSeq(Oplog[j],\ len_i + 1,\ len_j) \\
\qquad\qquad\qquad\qquad \text{ELSE}\ \langle\rangle
\end{array}$$

Can node i rollback its *log* based on j's *log*

$$\begin{array}{ll}
CanRollback(i,\ j) \triangleq & \land\ Len(Oplog[i]) > 0 \\
& \land\ Len(Oplog[j]) > 0 \\
& \land\ CurrentTerm[i] < CurrentTerm[j] \\
& \land \\
& \quad \lor\ Len(Oplog[i]) > Len(Oplog[j]) \\
& \quad \lor\ \land\ Len(Oplog[i]) \leq Len(Oplog[j]) \\
& \qquad \land\ CurrentTerm[i] \neq LogTerm(j,\ Len(Oplog[i]))
\end{array}$$

Returns the highest common index between two divergent logs.

If there is no common index between the logs, returns 0.

$$\begin{array}{l}
RollbackCommonPoint(i,\ j) \triangleq \\
\quad \text{LET}\ commonIndices \triangleq \{k \in \text{DOMAIN}\ Oplog[i] : \\
\qquad\qquad\qquad\qquad\qquad \land\ k \leq Len(Oplog[j]) \\
\qquad\qquad\qquad\qquad\qquad \land\ Oplog[i][k] = Oplog[j][k]\}\text{IN} \\
\qquad \text{IF}\ commonIndices = \{\}\ \text{THEN}\ 0\ \text{ELSE}\ \ MaxVal(commonIndices)
\end{array}$$

Init Part

$$\begin{array}{l}
InitPrimary \triangleq Primary = \{\text{CHOOSE}\ s \in Server : \text{TRUE}\} \\
InitSecondary \triangleq Secondary = Server \setminus Primary \\
InitOplog \triangleq Oplog = [s \in Server \mapsto \langle\rangle] \\
InitStore \triangleq Store = [n \in Server \cup Client \mapsto [k \in Key \mapsto Nil]] \\
InitCt \triangleq Ct = [n \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]
\end{array}$$

$InitOt \triangleq Ot = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitServerMsg \triangleq ServerMsg = [s \in Server \mapsto \langle\rangle]$
$InitPt \triangleq Pt = [s \in Server \mapsto 1]$
$InitCp \triangleq Cp = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitCalState \triangleq CalState = [s \in Server \mapsto CreateState(Cardinality(Server), \langle\rangle)]$

create initial *state*(*for calculate*)

$InitState \triangleq State = [s \in Server \mapsto [s0 \in Server \mapsto [p \mapsto 0, l \mapsto 0]]]$
$InitCurrentTerm \triangleq CurrentTerm = [s \in Server \mapsto 0]$
$InitReadyToServe \triangleq ReadyToServe = 0$
$InitSyncSource \triangleq SyncSource = [s \in Server \mapsto Nil]$

$Init \triangleq$
$\land InitPrimary \land InitSecondary \land InitOplog \land InitStore \land InitCt$
$\land InitOt \land InitPt \land InitCp \land InitCalState$
$\land InitServerMsg$
$\land InitState \land InitCurrentTerm \land InitReadyToServe$
$\land InitSyncSource$

Next *State* Actions
Replication Protocol: possible actions

$TurnOnReadyToServe \triangleq$
$\land ReadyToServe = 0$
$\land \exists\, s \in Primary :$
$\land CurrentTerm' = [CurrentTerm \text{ EXCEPT } ![s] = CurrentTerm[s] + 1]$
$\land ReadyToServe' = ReadyToServe + 1$
$\land \text{UNCHANGED } \langle Primary, Secondary, Oplog, Store, Ct, Ot, ServerMsg, Pt, Cp, State, CalState, SyncSource\rangle$

$Stepdown \triangleq$
$\land ReadyToServe > 0$
$\land \exists\, s \in Primary :$
$\land Primary' = Primary \setminus \{s\}$
$\land Secondary' = Secondary \cup \{s\}$
$\land \text{UNCHANGED } \langle Oplog, Store, Ct, Ot, ServerMsg, Pt, Cp, State, CalState, CurrentTerm, ReadyToServe, SyncSource\rangle$

There are majority nodes agree to elect node i to become primary
$ElectPrimary \triangleq$
$\land ReadyToServe > 0$
$\land \exists\, i \in Server : \exists\, majorNodes \in \text{SUBSET } (Server) :$
$\land \forall\, j \in majorNodes : \land NotBehind(i, j)$

$\land\ CurrentTerm[i] \geq CurrentTerm[j]$
$\land\ IsMajority(majorNodes)$
voted nodes for i cannot be primary anymore
$\land\ Primary' = \text{LET}\ possiblePrimary \triangleq Primary \setminus majorNodes$
$\text{IN}\quad possiblePrimary \cup \{i\}$
add voted nodes into secondaries
$\land\ Secondary' = \text{LET}\ possibleSecondary \triangleq Secondary \cup majorNodes$
$\text{IN}\quad possibleSecondary \setminus \{i\}$
$\land\ CurrentTerm' = [index \in Server \mapsto \text{IF}\ index \in (majorNodes \cup \{i\})$
$\text{THEN}\ CurrentTerm[i] + 1$
$\text{ELSE}\ CurrentTerm[index]]$
A primary node do not have any sync source
$\land\ SyncSource' = [SyncSource\ \text{EXCEPT}\ ![i] = Nil]$
$\land\ \text{UNCHANGED}\ \langle Oplog,\ Store,\ Ct,\ Ot,\ ServerMsg,\ Pt,\ Cp,\ State,\ CalState,$
$ReadyToServe\rangle$

$AdvanceCp \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, s \in Primary :$
$Cp' = [Cp\ \text{EXCEPT}\ ![s] = CalState[s][Cardinality(Server) \div 2 + 1]]$
$\land\ \text{UNCHANGED}\ \langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,$
$ServerMsg,\ \ Pt,\ CalState,$
$State,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

heartbeatoplogOtstore
$ServerTakeHeartbeat \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, s \in Server :$
$\land\ Len(ServerMsg[s]) \neq 0$ message channel is not empty
$\land\ ServerMsg[s][1].type =$ "heartbeat"
$\land\ Ct' = [Ct\ \text{EXCEPT}\ ![s] = HLCMax(Ct[s],\ ServerMsg[s][1].ct)]$
$\land\ State' =$
$\text{LET}\ SubHbState \triangleq State[s]$
$hb \triangleq [SubHbState\ \text{EXCEPT}\ ![ServerMsg[s][1].s] =$
$ServerMsg[s][1].aot]$
$\text{IN}\quad [State\ \text{EXCEPT}\ ![s] = hb]$
$\land\ CalState' = \text{LET}\ newcal \triangleq$
$\text{IF}\ s \in Primary$ primary node: update $CalState$
$\text{THEN}\ [CalState\ \text{EXCEPT}\ ![s] =$
$AdvanceState(ServerMsg[s][1].aot,\ State[s][ServerMsg[s][1].s],\ CalState$
$\text{ELSE}\ CalState$
$\text{IN}\quad newcal$
$\land\ Cp' = \text{LET}\ newcp \triangleq$
primary node: compute new mcp
$\text{IF}\ s \in Primary\ \text{THEN}\ CalState'[s][Cardinality(Server) \div 2 + 1]$

```
                    secondary node: update mcp
                ELSE IF ¬HLCLt(ServerMsg[s][1].cp, Cp[s])
                        ∧ ¬HLCLt(Ot[s], ServerMsg[s][1].cp)
                    THEN ServerMsg[s][1].cp
                    ELSE Cp[s]
            IN  [Cp EXCEPT ![s] = newcp]
        ∧ ServerMsg' = [ServerMsg EXCEPT ![s] = Tail(@)]
        ∧ CurrentTerm' = [CurrentTerm EXCEPT ![s] = Max(CurrentTerm[s], ServerMsg[s][1].term)]
    ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ot, Pt,
                 ReadyToServe, SyncSource⟩

ServerTakeUpdatePosition ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Server :
        ∧ Len(ServerMsg[s]) ≠ 0   message channel is not empty
        ∧ ServerMsg[s][1].type = "update_position"
        ∧ Ct' = [Ct EXCEPT ![s] = HLCMax(Ct[s], ServerMsg[s][1].ct)]   update ct accordingly
        ∧ State' =
            LET SubHbState ≜ State[s]
                hb ≜ [SubHbState EXCEPT ![ServerMsg[s][1].s] =
                      ServerMsg[s][1].aot]
            IN  [State EXCEPT ![s] = hb]
        ∧ CalState' = LET newcal ≜
                          IF s ∈ Primary   primary node: update CalState
                              THEN [CalState EXCEPT ![s] =
                                        AdvanceState(ServerMsg[s][1].aot, State[s][ServerMsg[s][1].s], CalState
                              ELSE CalState
                      IN  newcal
        ∧ Cp' = LET newcp ≜
                    primary node: compute new mcp
                IF s ∈ Primary THEN CalState'[s][Cardinality(Server) ÷ 2 + 1]
                    secondary node: update mcp
                ELSE IF ¬HLCLt(ServerMsg[s][1].cp, Cp[s])
                        ∧ ¬HLCLt(Ot[s], ServerMsg[s][1].cp)
                THEN ServerMsg[s][1].cp
                ELSE Cp[s]
                IN  [Cp EXCEPT ![s] = newcp]
        ∧ CurrentTerm' = [CurrentTerm EXCEPT ![s] = Max(CurrentTerm[s], ServerMsg[s][1].term)]
        ∧ ServerMsg' = LET newServerMsg ≜ [ServerMsg EXCEPT ![s] = Tail(@)]
                       IN  (LET appendMsg ≜ [type ↦ "update_position", s ↦ ServerMsg[s][1].s, aot ↦ Se
                                             ct ↦ ServerMsg[s][1].ct, cp ↦ ServerMsg[s][1].cp, term ↦ ServerM
                            IN  (LET newMsg ≜ IF s ∈ Primary ∨ SyncSource[s] = Nil
                                              THEN newServerMsg   If s is primary, accept the msg, else f
                                              ELSE [newServerMsg EXCEPT ![SyncSource[s]] = Append
                                 IN  newMsg))
```

$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Oplog,\ Store,\ Ot,$
$Pt,\ ReadyToServe,\ SyncSource\rangle$

$NTPSync \triangleq$ simplify *NTP* protocal
$\land\ ReadyToServe > 0$
$\land\ Pt' = [s \in Server \mapsto MaxPt]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,$
$ServerMsg,\ Cp,$
$CalState,\ State,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

$AdvancePt \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, s \in Server :$
$\land\ s \in Primary$ for simplicity
$\land\ Pt[s] \leq PtStop$
$\land\ Pt' = [Pt$ EXCEPT $![s] = @ + 1]$ advance physical time
$\land\ BroadcastHeartbeat(s)$ broadcast heartbeat periodly
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,\ State,$
$Cp,\ CalState,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

Replicate oplog from node j to node i, and update related structures accordingly
$Replicate \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, i,\ j \in Server :$
$\land\ CanSyncFrom(i,\ j)$ i can sync from j only need not to rollback
$\land\ i \in Secondary$
$\land\ ReplicateOplog(i,\ j) \neq \langle\rangle$
$\land\ Oplog' = [Oplog$ EXCEPT $![i] = @ \circ ReplicateOplog(i,\ j)]$
$\land\ Store' = [Store$ EXCEPT $![i] = Store[j]]$
$\land\ Ct' = [Ct$ EXCEPT $![i] = HLCMax(Ct[i],\ Ct[j])]$ update $Ct[i]$
$\land\ Ot' = [Ot$ EXCEPT $![i] = HLCMax(Ot[i],\ Ot[j])]$ update $Ot[i]$
$\land\ Cp' = [Cp$ EXCEPT $![i] = HLCMax(Cp[i],\ Cp[j])]$ update $Cp[i]$
$\land\ CurrentTerm' = [CurrentTerm$ EXCEPT $![i] = Max(CurrentTerm[i],\ CurrentTerm[j])]$ update *Curren*
$\land\ State' =$
LET $SubHbState \triangleq State[i]$
$hb \triangleq [SubHbState$ EXCEPT $![j] = Ot[j]]$
IN $[State$ EXCEPT $![i] = hb]$ update j's state i knows
$\land$ LET $msg \triangleq [type \mapsto$ "update_position", $s \mapsto i,\ aot \mapsto Ot'[i],\ ct \mapsto Ct'[i],\ cp \mapsto Cp'[i],\ term \mapsto Curr$
IN $ServerMsg' = [ServerMsg$ EXCEPT $![j] = Append(ServerMsg[j],\ msg)]$
$\land\ SyncSource' = [SyncSource$ EXCEPT $![i] = j]$
$\land\ CalState' = [CalState$ EXCEPT $![i] = CalState[j]]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Pt,\ ReadyToServe\rangle$

Rollback i's oplog and recover it to j's state
Recover to j's state immediately to prevent internal client request
$RollbackAndRecover \triangleq$

$\land\ ReadyToServe > 0$
$\land\ \exists\, i, j \in Server :$
$\land\ i \in Secondary$
$\land\ CanRollback(i, j)$
$\land$ LET $cmp \triangleq RollbackCommonPoint(i, j)$ IN
LET $commonLog \triangleq SubSeq(Oplog[i], 1, cmp)$
$appendLog \triangleq SubSeq(Oplog[j], cmp + 1, Len(Oplog[j]))$
IN $Oplog' = [Oplog$ EXCEPT $![i] = commonLog \circ appendLog]$
$\land\ CurrentTerm' = [CurrentTerm$ EXCEPT $![i] = Max(CurrentTerm[i], CurrentTerm[j])]$ update *Curren*
$\land\ Store' = [Store$ EXCEPT $![i] = Store[j]]$
$\land\ Ct' = [Ct$ EXCEPT $![i] = HLCMax(Ct[i], Ct[j])]$ update $Ct[i]$
$\land\ Ot' = [Ot$ EXCEPT $![i] = HLCMax(Ot[i], Ot[j])]$ update $Ot[i]$
$\land\ Cp' = [Cp$ EXCEPT $![i] = HLCMax(Cp[i], Cp[j])]$ update $Cp[i]$
$\land\ State' =$ LET $SubHbState \triangleq State[i]$
$hb \triangleq [SubHbState$ EXCEPT $![j] = Ot[j]]$
IN $[State$ EXCEPT $![i] = hb]$ update j's state i knows
$\land$ LET $msg \triangleq [type \mapsto$ "update_position", $s \mapsto i,\ aot \mapsto Ot'[i],\ ct \mapsto Ct'[i],\ cp \mapsto Cp'[i]]$
IN $ServerMsg' = [ServerMsg$ EXCEPT $![j] = Append(ServerMsg[j], msg)]$
$\land\ SyncSource' = [SyncSource$ EXCEPT $![i] = j]$
$\land$ UNCHANGED $\langle Primary, Secondary, Pt, CalState,$
$ReadyToServe\rangle$

$ClientRequest \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, s \in Server,\ k \in Key,\ v \in Value :$
$\land\ s \in Primary$
$\land\ Tick(s)$
$\land\ Ot' = [Ot$ EXCEPT $![s] = Ct'[s]]$
$\land\ Store' = [Store$ EXCEPT $![s][k] = v]$
$\land\ Oplog' =$ LET $entry \triangleq [k \mapsto k,\ v \mapsto v,\ ot \mapsto Ot'[s],\ term \mapsto CurrentTerm[s]]$
$newLog \triangleq Append(Oplog[s], entry)$
IN $[Oplog$ EXCEPT $![s] = newLog]$
$\land\ State' =$ LET $SubHbState \triangleq State[s]$
$hb \triangleq [SubHbState$ EXCEPT $![s] = Ot'[s]]$
IN $[State$ EXCEPT $![s] = hb]$
$\land\ CalState' = [CalState$ EXCEPT $![s] =$
$AdvanceState(Ot'[s], Ot[s], CalState[s])]$
$\land$ UNCHANGED $\langle Primary,\ Secondary, ServerMsg,$
$Pt, Cp,$
$CurrentTerm, ReadyToServe, SyncSource\rangle$

Next state for all configurations
$Next \triangleq \lor Replicate$
$\lor AdvancePt$
$\lor ServerTakeHeartbeat$

$\lor$ *ServerTakeUpdatePosition*
$\lor$ *Stepdown*
$\lor$ *RollbackAndRecover*
$\lor$ *TurnOnReadyToServe*
$\lor$ *ElectPrimary*
$\lor$ *ClientRequest*
$\lor$ *NTPSync*

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

Properties to check?

$IsLogPrefix(i, j) \triangleq$
$\land Len(Oplog[i]) \leq Len(Oplog[j])$
$\land Oplog[i] = SubSeq(Oplog[j], 1, Len(Oplog[i]))$

If two logs have the same last *log* entry term, then one is a prefix of the other (from Will)
$LastTermsEquivalentImplyPrefixes \triangleq$
$\forall i, j \in Server :$
$LogTerm(i, Len(Oplog[i])) = LogTerm(j, Len(Oplog[j])) \Rightarrow$
$IsLogPrefix(i, j) \lor IsLogPrefix(j, i)$

Check whether terms are incremented monotoniclly (from Will
$TermsMonotonic \triangleq$
$\Box[\forall s \in Server : CurrentTerm'[s] \geq CurrentTerm[s]]_{vars}$

Check the *log* in *Primary* node is append only (from Will
$PrimaryAppendOnly \triangleq$
$\Box[\forall s \in Server : s \in Primary \Rightarrow Len(Oplog'[s]) \geq Len(Oplog[s])]_{vars}$

Never rollback oplog before common point (from Will & Raft *Mongo*
$NeverRollbackCommonPoint \triangleq$
$\exists i, j \in Server : CanRollback(i, j) \Rightarrow$
LET $commonPoint \triangleq RollbackCommonPoint(i, j)$
$lastOplog \triangleq Oplog[i][commonPoint]$
IN $HLCLt(Cp[i], lastOplog.ot)$

Eventually *log* correctness (from Will
$EventuallyLogsConverge \triangleq \Diamond\Box[\forall s, t \in Server : s \neq t \Rightarrow Oplog[s] = Oplog[t]]_{vars}$
$EventuallyLogsNonEmpty \triangleq \Diamond(\exists s \in Server : Len(Oplog[s]) > 0)$

(from *RaftMongo*
$TwoPrimariesInSameTerm \triangleq$
$\exists i, j \in Server :$
$\land i \neq j$
$\land CurrentTerm[i] = CurrentTerm[j]$
$\land i \in Primary$

$\land\ j \in Primary$

$NoTwoPrimariesInSameTerm \triangleq \neg TwoPrimariesInSameTerm$

Check if there is any cycle of sync source path (from *RaftMongo Sync*

$SyncSourceCycleTwoNode \triangleq$
$\quad \exists\, s,\, t \in Server :$
$\qquad \land\ s \neq t$
$\qquad \land\ SyncSource[s] = t$
$\qquad \land\ SyncSource[t] = s$

$BoundedSeq(s,\, n) \triangleq [1 \,..\, n \to s]$

$SyncSourcePaths \triangleq$
$\quad \{p \in BoundedSeq(Server,\, Cardinality(Server)) :$
$\qquad \forall\, i \in 1 \,..\, (Len(p) - 1) : SyncSource[p[i]] = p[i + 1]\}$

$SyncSourcePath(i,\, j) \triangleq$
$\quad \exists\, p \in SyncSourcePaths :$
$\qquad \land\ Len(p) > 1$
$\qquad \land\ p[1] = i$
$\qquad \land\ p[Len(p)] = j$

$SyncSourceCycle \triangleq$
$\quad \exists\, s \in Server : SyncSourcePath(s,\, s)$

$NonTrivialSyncCycle \triangleq SyncSourceCycle \land \neg SyncSourceCycleTwoNode$
$NoNonTrivialSyncCycle \triangleq \neg NonTrivialSyncCycle$

\ ∗ Modification *History*
\ ∗ Last modified *Fri Apr* 22 10:10:51 *CST* 2022 by *dh*
\ ∗ Created *Mon Apr* 18 11:38:53 *CST* 2022 by *dh*