\ * To fix: heartbeat

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
CurrentTerm, *CurrentTerm*[*s*]: current election term at server *s*
→ updated in *update_position*, heartbeat and replicate
SyncSource *SyncSource*[*s*]: sync source of server node *s*

---

group related *vars* to optimize code

$electionVars \triangleq \langle Primary, Secondary \rangle$ *vars* that are related to election
$storageVars \triangleq \langle Oplog, Store \rangle$ *vars* that are related to storage
$messageVar \triangleq \langle ServerMsg \rangle$ var that is related to message
$serverVars \triangleq \langle Ot, SyncSource \rangle$ *vars* that each server node holds for itself
$learnableVars \triangleq \langle Ct, State, Cp, CurrentTerm \rangle$ *vars* that must learn from msgs
$timeVar \triangleq \langle Pt \rangle$ var that is used for timing

---

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

$$
\begin{aligned}
HLCLt(x, y) \triangleq\ & \text{IF } x.p < y.p \\
& \quad \text{THEN TRUE} \\
& \quad \text{ELSE IF } x.p = y.p \\
& \qquad \text{THEN IF } x.l < y.l \\
& \qquad\quad \text{THEN TRUE} \\
& \qquad\quad \text{ELSE FALSE}
\end{aligned}
$$

```
                    ELSE  FALSE

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

vars ≜ ⟨Primary, Secondary, Oplog, Store, Ct, Ot, messageVar,
        Pt, Cp, State, CurrentTerm, SyncSource⟩

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

Is node i ahead of node j
NotBehind(i, j) ≜ ∨ LastTerm(i) > LastTerm(j)
                  ∨ ∧ LastTerm(i) = LastTerm(j)
                    ∧ Len(Oplog[i]) ≥ Len(Oplog[j])

IsMajority(servers) ≜ Cardinality(servers) * 2 > Cardinality(Server)

Return the maximum value from a set, or undefined if the set is empty.
MaxVal(s) ≜ CHOOSE x ∈ s : ∀ y ∈ s : x ≥ y
HLCMinSet(s) ≜ CHOOSE x ∈ s : ∀ y ∈ s : ¬HLCLt(y, x)

clock
MaxPt ≜ LET x ≜ CHOOSE s ∈ Server : ∀ s1 ∈ Server \ {s} :
                                      Pt[s] ≥ Pt[s1]
        IN  Pt[x]

Tick(s) ≜ Ct' = IF Ct[s].p ≥ Pt[s]
                THEN [Ct EXCEPT ![s] = [p ↦ @.p, l ↦ @.l + 1]]
                ELSE [Ct EXCEPT ![s] = [p ↦ Pt[s], l ↦ 0]]

heartbeat
Only Primary node sends heartbeat once advance pt
BroadcastHeartbeat(s) ≜
  LET msg ≜ [type ↦ "heartbeat", s ↦ s, aot ↦ Ot[s],
             ct ↦ Ct[s], cp  ↦ Cp[s], term ↦ CurrentTerm[s]]
  IN  ServerMsg' = [x ∈ Server ↦ IF x = s THEN ServerMsg[x]
                                  ELSE Append(ServerMsg[x], msg)]

Can node i sync from node j?
CanSyncFrom(i, j) ≜
  ∧ Len(Oplog[i]) < Len(Oplog[j])
  ∧ LastTerm(i) = LogTerm(j, Len(Oplog[i]))

Oplog entries needed to replicate from j to i
```

```
ReplicateOplog(i, j) ≜
    LET len_i ≜ Len(Oplog[i])
        len_j ≜ Len(Oplog[j])
    IN  IF i ∈ Secondary ∧ len_i < len_j
                THEN SubSeq(Oplog[j], len_i + 1, len_j)
                ELSE ⟨⟩

 Can node i rollback its log based on j's log
CanRollback(i, j) ≜ ∧ Len(Oplog[i]) > 0
                    ∧ Len(Oplog[j]) > 0
                    ∧ CurrentTerm[i] < CurrentTerm[j]
                    ∧
                       ∨ Len(Oplog[i]) > Len(Oplog[j])
                       ∨ ∧ Len(Oplog[i]) ≤ Len(Oplog[j])
                         ∧ CurrentTerm[i] ≠ LogTerm(j, Len(Oplog[i]))

 Returns the highest common index between two divergent logs.
 If there is no common index between the logs, returns 0.
RollbackCommonPoint(i, j) ≜
    LET commonIndices ≜ {k ∈ DOMAIN Oplog[i] :
                            ∧ k ≤ Len(Oplog[j])
                            ∧ Oplog[i][k] = Oplog[j][k]}IN
        IF commonIndices = {} THEN 0 ELSE MaxVal(commonIndices)

 The set of all quorums. This just calculates simple majorities, but the only
 important property is that every quorum overlaps with every other.
Quorum ≜ {i ∈ SUBSET (Server) : Cardinality(i) * 2 > Cardinality(Server)}

QuorumAgreeInSameTerm(states) ≜
    LET quorums ≜ {Q ∈ Quorum :
                    Make sure all nodes in quorum have actually applied some entries.
                   ∧ ∨ ∀ s ∈ Q : states[s].p > 0
                     ∨ ∧ ∀ s ∈ Q : states[s].p = 0
                       ∧ ∀ s ∈ Q : states[s].l > 0
                    Make sure every applied entry in quorum has the same term.
                   ∧ ∀ s, t ∈ Q :
                     s ≠ t ⇒ states[s].term = states[s].term}
    IN  quorums

 compute a new common point according to new update position msg
ComputeNewCp(s) ≜
     primary node: compute new mcp
    IF s ∈ Primary THEN
        LET quorumAgree ≜ QuorumAgreeInSameTerm(State[s])IN
        IF Cardinality(quorumAgree) > 0
            THEN LET QuorumSet ≜ CHOOSE i ∈ quorumAgree : TRUE
```

$serverInQuorum \triangleq$ CHOOSE $j \in QuorumSet$: TRUE
$termOfQuorum \triangleq State[s][serverInQuorum].term$
$StateSet \triangleq \{[p \mapsto State[s][j].p,\ l \mapsto State[s][j].l] : j \in QuorumSet\}$
$newCommitPoint \triangleq HLCMinSet(StateSet)$
IN IF $termOfQuorum = CurrentTerm[s]$
THEN $[p \mapsto newCommitPoint.p,\ l \mapsto newCommitPoint.l,\ term \mapsto termOfQuorum]$
ELSE $Cp[s]$
ELSE $Cp[s]$
secondary node: update mcp
ELSE IF $Len(ServerMsg[s]) \neq 0$ THEN
LET $msgCP \triangleq [p \mapsto ServerMsg[s][1].cp.p,\ l \mapsto ServerMsg[s][1].cp.l]$IN
IF $\land \neg HLCLt(msgCP,\ Cp[s])$
$\land \neg HLCLt(Ot[s],\ msgCP)$
The term of *cp* must equal to the *CurrentTerm* of that node to advance it
$\land\ ServerMsg[s][1].term = CurrentTerm[s]$
THEN $ServerMsg[s][1].cp$
ELSE $Cp[s]$
ELSE $Cp[s]$

$GetNewState(s,\ d,\ np,\ nl,\ nterm) \triangleq$
LET $newSubState \triangleq [p \mapsto np,\ l \mapsto nl,\ term \mapsto nterm]$
$sState \triangleq State[s]$
IN $[sState$ EXCEPT $![d] = newSubState]$

Init Part

$InitPrimary \triangleq Primary = \{$CHOOSE $s \in Server$: TRUE$\}$
$InitSecondary \triangleq Secondary = Server \setminus Primary$
$InitOplog \triangleq Oplog = [s \in Server \mapsto \langle\rangle]$
$InitStore \triangleq Store = [n \in Server \cup Client \mapsto [k \in Key \mapsto Nil]]$
$InitCt \triangleq Ct = [n \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]$
$InitOt \triangleq Ot = [n \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]$
$InitServerMsg \triangleq ServerMsg = [s \in Server \mapsto \langle\rangle]$
$InitPt \triangleq Pt = [s \in Server \mapsto 1]$
$InitCp \triangleq Cp = [n \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]$
$InitState \triangleq State = [s \in Server \mapsto [s0 \in Server \mapsto [p \mapsto 0,\ l \mapsto 0,\ term \mapsto 0]]]$
$InitCurrentTerm \triangleq CurrentTerm = [p \in Primary \mapsto 1]$ @@ $[s \in Server \mapsto 0]$
$InitSyncSource \triangleq SyncSource = [s \in Server \mapsto Nil]$

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

Next *State* Actions
Replication Protocol: possible actions

$Stepdown \triangleq$
$\land \exists\, s \in Primary :$
$\land\ Primary' = Primary \setminus \{s\}$
$\land\ Secondary' = Secondary \cup \{s\}$
$\land$ UNCHANGED $\langle storageVars,\ serverVars,\ Ct,\ messageVar,\ timeVar,\ Cp,\ State,\ CurrentTerm\rangle$

Todo: *Stepdown* when receiving a higher term heartbeat

There are majority nodes agree to elect node i to become primary
$ElectPrimary \triangleq$
$\land \exists\, i \in Server : \exists\, majorNodes \in$ 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
$\land\ Primary' =$ LET $possiblePrimary \triangleq Primary \setminus majorNodes$
IN $possiblePrimary \cup \{i\}$
add voted nodes into secondaries
$\land\ Secondary' =$ LET $possibleSecondary \triangleq Secondary \cup majorNodes$
IN $possibleSecondary \setminus \{i\}$
$\land\ CurrentTerm' = [index \in Server \mapsto$ IF $index \in (majorNodes \cup \{i\})$
THEN $CurrentTerm[i] + 1$
ELSE $CurrentTerm[index]]$
A primary node do not have any sync source
$\land\ SyncSource' = [SyncSource$ EXCEPT $![i] = Nil]$
$\land$ UNCHANGED $\langle storageVars,\ Ct,\ Ot,\ messageVar,\ timeVar,\ Cp,\ State\rangle$

$AdvanceCp \triangleq$
$\land \exists\, s \in Primary :$
LET $newCp \triangleq ComputeNewCp(s)$
IN $Cp' = [Cp$ EXCEPT $![s] = newCp]$
$\land$ UNCHANGED $\langle electionVars,\ storageVars,\ serverVars,\ Ct,\ messageVar,\ timeVar,\ State,\ CurrentTerm\rangle$

heartbeatoplog*Ot*store
$ServerTakeHeartbeat \triangleq$
$\land \exists\, s \in Server :$
$\land\ Len(ServerMsg[s]) \neq 0$ message channel is not empty
$\land\ ServerMsg[s][1].type =$ "heartbeat"
$\land\ CurrentTerm[s] = ServerMsg[s][1].term$
$\land\ Ct' = [Ct$ EXCEPT $![s] = HLCMax(Ct[s],\ ServerMsg[s][1].ct)]$
$\land\ State' =$ LET $newState \triangleq GetNewState(s,\ ServerMsg[s][1].s,\ ServerMsg[s][1].aot.p,\ ServerMsg[s][1]$
IN $[State$ EXCEPT $![s] = newState]$
$\land\ Cp' =$ LET $newcp \triangleq ComputeNewCp(s)$

IN $[Cp$ EXCEPT $![s] = newcp]$
$\land\ ServerMsg' = [ServerMsg$ EXCEPT $![s] = Tail(@)]$
$\land\ CurrentTerm' = [CurrentTerm$ EXCEPT $![s] = Max(CurrentTerm[s],\ ServerMsg[s][1].term)]$
$\land$ UNCHANGED $\langle electionVars,\ storageVars,\ serverVars,\ timeVar\rangle$

$ServerTakeUpdatePosition\ \triangleq$
$\land\ \exists\, s \in Server :$
$\land\ Len(ServerMsg[s]) \neq 0$ message channel is not empty
$\land\ ServerMsg[s][1].type =$ "update_position"
$\land\ Ct' = [Ct$ EXCEPT $![s] = HLCMax(Ct[s],\ ServerMsg[s][1].ct)]$ update ct accordingly
$\land\ State' =$ LET $newState \triangleq GetNewState(s,\ ServerMsg[s][1].s,\ ServerMsg[s][1].aot.p,\ ServerMsg[s][1]$
IN $[State$ EXCEPT $![s] = newState]$
$\land\ Cp' =$ LET $newcp \triangleq ComputeNewCp(s)$
IN $[Cp$ EXCEPT $![s] = newcp]$
$\land\ CurrentTerm' = [CurrentTerm$ EXCEPT $![s] = Max(CurrentTerm[s],\ ServerMsg[s][1].term)]$
$\land\ ServerMsg' =$ LET $newServerMsg \triangleq [ServerMsg$ EXCEPT $![s] = Tail(@)]$
IN (LET $appendMsg \triangleq [type \mapsto$ "update_position", $s \mapsto ServerMsg[s][1].s,\ aot \mapsto Se$
$ct \mapsto ServerMsg[s][1].ct,\ cp \mapsto ServerMsg[s][1].cp,\ term \mapsto ServerM$
IN (LET $newMsg \triangleq$ IF $s \in Primary \lor SyncSource[s] = Nil$
THEN $newServerMsg$ If s is primary, accept the msg, else f
ELSE $[newServerMsg$ EXCEPT $![SyncSource[s]] = Append$
IN $newMsg))$
$\land$ UNCHANGED $\langle electionVars,\ storageVars,\ serverVars,\ timeVar\rangle$

$NTPSync\ \triangleq$ simplify NTP protocal
$\land\ Pt' = [s \in Server \mapsto MaxPt]$
$\land$ UNCHANGED $\langle electionVars,\ storageVars,\ serverVars,\ learnableVars,\ messageVar\rangle$

$AdvancePt\ \triangleq$
$\land\ \exists\, s \in Server :$
$\land\ s \in Primary$ for simplicity
$\land\ Pt[s] \leq PtStop$
$\land\ Pt' = [Pt$ EXCEPT $![s] = @ + 1]$ advance physical time
$\land\ BroadcastHeartbeat(s)$ broadcast heartbeat periodly
$\land$ UNCHANGED $\langle electionVars,\ storageVars,\ serverVars,\ learnableVars\rangle$

Replicate oplog from node j to node i, and update related structures accordingly
$Replicate\ \triangleq$
$\land\ \exists\, i, j \in Server :$
$\land\ CanSyncFrom(i, j)$ 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 $Ot[i]$

$\land\ Cp' = [Cp \text{ EXCEPT } ![i] = HLCMax(Cp[i],\ Cp[j])]$ update $Cp[i]$
$\land\ CurrentTerm' = [CurrentTerm \text{ EXCEPT } ![i] = Max(CurrentTerm[i],\ CurrentTerm[j])]$ update $Curren$
$\land\ State' = \text{LET } newState \triangleq GetNewState(i,\ j,\ Ot[j].p,\ Ot[j].l,\ CurrentTerm[j])$ update j's state i knows
IN $[State \text{ EXCEPT } ![i] = newState]$
$\land \text{LET } msg \triangleq [type \mapsto$ "update_position", $s \mapsto i,\ aot \mapsto Ot'[i],\ ct \mapsto Ct'[i],\ cp \mapsto Cp'[i],\ term \mapsto Curr$
IN $ServerMsg' = [ServerMsg \text{ EXCEPT } ![j] = Append(ServerMsg[j],\ msg)]$
$\land\ SyncSource' = [SyncSource \text{ EXCEPT } ![i] = j]$
$\land \text{UNCHANGED } \langle electionVars,\ timeVar \rangle$

Rollback i's oplog and recover it to j's state
Recover to j's state immediately to prevent internal client request
$RollbackAndRecover \triangleq$
$\land \exists\, i, j \in Server :$
$\land\ i \in Secondary$
$\land\ CanRollback(i,\ j)$
$\land \text{LET } cmp \triangleq RollbackCommonPoint(i,\ j) \text{ IN}$
LET $commonLog \triangleq SubSeq(Oplog[i],\ 1,\ cmp)$
$appendLog \triangleq SubSeq(Oplog[j],\ cmp + 1,\ Len(Oplog[j]))$
IN $Oplog' = [Oplog \text{ EXCEPT } ![i] = commonLog \circ appendLog]$
$\land\ CurrentTerm' = [CurrentTerm \text{ EXCEPT } ![i] = Max(CurrentTerm[i],\ CurrentTerm[j])]$ update $Curren$
$\land\ Store' = [Store \text{ EXCEPT } ![i] = Store[j]]$
$\land\ Ct' = [Ct \text{ EXCEPT } ![i] = HLCMax(Ct[i],\ Ct[j])]$ update $Ct[i]$
$\land\ Ot' = [Ot \text{ EXCEPT } ![i] = HLCMax(Ot[i],\ Ot[j])]$ update $Ot[i]$
$\land\ Cp' = [Cp \text{ EXCEPT } ![i] = HLCMax(Cp[i],\ Cp[j])]$ update $Cp[i]$
$\land\ State' =$
LET $newStatei \triangleq [p \mapsto Ot'[i].p,\ l \mapsto Ot'[i].l,\ term \mapsto CurrentTerm'[i]]$
$newStatej \triangleq [p \mapsto Ot[j].p,\ l \mapsto Ot[j].l,\ term \mapsto CurrentTerm[j]]$
IN LET $SubHbState \triangleq State[i]$
$hb \triangleq [SubHbState \text{ EXCEPT } ![i] = newStatei]$ update i's self state (used in mcp computation
$hb1 \triangleq [hb \text{ EXCEPT } ![j] = newStatej]$ update j's state i knows
IN $[State \text{ EXCEPT } ![i] = hb1]$
$\land \text{LET } msg \triangleq [type \mapsto$ "update_position", $s \mapsto i,\ aot \mapsto Ot'[i],\ ct \mapsto Ct'[i],\ cp \mapsto Cp'[i],\ term \mapsto Curr$
IN $ServerMsg' = [ServerMsg \text{ EXCEPT } ![j] = Append(ServerMsg[j],\ msg)]$
$\land\ SyncSource' = [SyncSource \text{ EXCEPT } ![i] = j]$
$\land \text{UNCHANGED } \langle electionVars,\ timeVar \rangle$

$ClientRequest \triangleq$
$\land \exists\, s \in Server,\ k \in Key,\ v \in Value :$
$\land\ s \in Primary$
$\land\ Tick(s)$
$\land\ Ot' = [Ot \text{ EXCEPT } ![s] = Ct'[s]]$
$\land\ Store' = [Store \text{ EXCEPT } ![s][k] = v]$
$\land\ Oplog' = \text{LET } entry \triangleq [k \mapsto k,\ v \mapsto v,\ ot \mapsto Ot'[s],\ term \mapsto CurrentTerm[s]]$
$newLog \triangleq Append(Oplog[s],\ entry)$
IN $[Oplog \text{ EXCEPT } ![s] = newLog]$

$\wedge State' = \text{LET } newState \triangleq GetNewState(s, s, Ot'[s].p, Ot'[s].l, CurrentTerm[s])$ update i's state
$\quad\quad \text{IN } [State \text{ EXCEPT } ![s] = newState]$
$\wedge \text{UNCHANGED } \langle electionVars, messageVar, timeVar, Cp, CurrentTerm, SyncSource \rangle$

Next state for all configurations

$$
\begin{aligned}
Next \triangleq\ & \vee Replicate \\
& \vee AdvancePt \\
& \vee AdvanceCp \\
& \vee ServerTakeHeartbeat \\
& \vee ServerTakeUpdatePosition \\
& \vee Stepdown \\
& \vee RollbackAndRecover \\
& \vee ElectPrimary \\
& \vee ClientRequest \\
& \vee NTPSync
\end{aligned}
$$

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

Properties to check?

$$
\begin{aligned}
& IsLogPrefix(i, j) \triangleq \\
& \quad \wedge Len(Oplog[i]) \leq Len(Oplog[j]) \\
& \quad \wedge Oplog[i] = SubSeq(Oplog[j], 1, Len(Oplog[i]))
\end{aligned}
$$

If two logs have the same last log entry term, then one is a prefix of the other (from Will

$$
\begin{aligned}
& LastTermsEquivalentImplyPrefixes \triangleq \\
& \quad \forall i, j \in Server : \\
& \quad\quad LogTerm(i, Len(Oplog[i])) = LogTerm(j, Len(Oplog[j])) \Rightarrow \\
& \quad\quad IsLogPrefix(i, j) \vee IsLogPrefix(j, i)
\end{aligned}
$$

Check whether terms are incremented monotonicly (from Will

$$
\begin{aligned}
& TermsMonotonic \triangleq \\
& \quad \Box[\forall s \in Server : CurrentTerm'[s] \geq CurrentTerm[s]]_{vars}
\end{aligned}
$$

Check the log in *Primary* node is append only (from Will

$$
\begin{aligned}
& PrimaryAppendOnly \triangleq \\
& \quad \Box[\forall s \in Server : s \in Primary \Rightarrow Len(Oplog'[s]) \geq Len(Oplog[s])]_{vars}
\end{aligned}
$$

Never rollback oplog before common point (from Will & Raft *Mongo*

$$
\begin{aligned}
& NeverRollbackCommonPoint \triangleq \\
& \quad \exists i, j \in Server : CanRollback(i, j) \Rightarrow \\
& \quad\quad \text{LET } commonPoint \triangleq RollbackCommonPoint(i, j) \\
& \quad\quad\quad\quad lastOplog \triangleq Oplog[i][commonPoint] \\
& \quad\quad \text{IN } HLCLt(Cp[i], lastOplog.ot)
\end{aligned}
$$

Eventually log correctness (from Will

$$
\begin{aligned}
& EventuallyLogsConverge \triangleq \Diamond\Box[\forall s, t \in Server : s \neq t \Rightarrow Oplog[s] = Oplog[t]]_{vars} \\
& EventuallyLogsNonEmpty \triangleq \Diamond(\exists s \in Server : Len(Oplog[s]) > 0)
\end{aligned}
$$

(from *RaftMongo*

$$TwoPrimariesInSameTerm \triangleq$$
$$\exists\, i,\, j \in Server :$$
$$\wedge\ i \neq j$$
$$\wedge\ CurrentTerm[i] = CurrentTerm[j]$$
$$\wedge\ i \in Primary$$
$$\wedge\ j \in Primary$$

$$NoTwoPrimariesInSameTerm \triangleq \neg TwoPrimariesInSameTerm$$

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

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

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

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

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

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

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

\ ∗ Modification *History*
\ ∗ Last modified *Tue* May 24 16:55:51 *CST* 2022 by *dh*
\ ∗ Created *Mon Apr* 18 11:38:53 *CST* 2022 by *dh*