\ * 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
          ReadyToServe,            equal to 0 before any primary is elected
          SyncSource               SyncSource[s]: sync source of server node s

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

 Helpers
------------------------------------------------------------------------------------

HLCLt(x, y) ≜ IF x.p < y.p
                THEN TRUE
               ELSE IF x.p = y.p
                THEN IF x.l < y.l
                         THEN TRUE
                      ELSE FALSE
                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, ServerMsg,
          Pt, Cp, State, CurrentTerm, ReadyToServe, SyncSource⟩

RECURSIVE CreateState(_, _)  init state
CreateState(len, seq) ≜
    IF len = 0 THEN seq
     ELSE CreateState(len − 1, Append(seq, [p ↦ 0, l ↦ 0]))

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])
```

```
        $len\_j \triangleq Len(Oplog[j])$
    IN  $\text{IF } i \in Secondary \land len\_i < len\_j$
                    $\text{THEN } SubSeq(Oplog[j],\ len\_i + 1,\ len\_j)$
                    $\text{ELSE } \langle\rangle$
```

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

```
$CanRollback(i, j) \triangleq$ $\land\ Len(Oplog[i]) > 0$
                      $\land\ Len(Oplog[j]) > 0$
                      $\land\ CurrentTerm[i] < CurrentTerm[j]$
                      $\land$
                         $\lor\ Len(Oplog[i]) > Len(Oplog[j])$
                         $\lor\ \land\ Len(Oplog[i]) \leq Len(Oplog[j])$
                            $\land\ CurrentTerm[i] \neq LogTerm(j,\ Len(Oplog[i]))$
```

Returns the highest common index between two divergent logs.
If there is no common index between the logs, returns 0.

```
$RollbackCommonPoint(i, j) \triangleq$
    LET $commonIndices \triangleq \{k \in \text{DOMAIN } Oplog[i] :$
                               $\land\ k \leq Len(Oplog[j])$
                               $\land\ Oplog[i][k] = Oplog[j][k]\}$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.

```
$Quorum \triangleq \{i \in \text{SUBSET } (Server) : Cardinality(i) * 2 > Cardinality(Server)\}$

$QuorumAgreeInSameTerm(states) \triangleq$
    LET $quorums \triangleq \{Q \in Quorum :$
                      Make sure all nodes in quorum have actually applied some entries.
                      $\land\ \lor\ \forall s \in Q : states[s].p > 0$
                         $\lor\ \land\ \forall s \in Q : states[s].p = 0$
                            $\land\ \forall s \in Q : states[s].l > 0$
                      Make sure every applied entry in quorum has the same term.
                      $\land\ \forall s,\ t \in Q :$
                         $s \neq t \Rightarrow states[s].term = states[s].term\}$
    IN  $quorums$
```

Init Part

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

```
$InitPt \triangleq Pt = [s \in Server \mapsto 1]$
$InitCp \triangleq Cp = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitState \triangleq State = [s \in Server \mapsto [s0 \in Server \mapsto$
                                    $[p \mapsto 0, l \mapsto 0, term \mapsto 0]]]$
$InitCurrentTerm \triangleq CurrentTerm = [s \in Server \mapsto 0]$
$InitReadyToServe \triangleq ReadyToServe = 0$
$InitSyncSource \triangleq SyncSource = [s \in Server \mapsto Nil]$

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

Next *State* Actions

Replication Protocol: possible actions

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

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

Todo: *Stepdown* when receiving a higher term heartbeat

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
        $\land Primary' = \text{LET } possiblePrimary \triangleq Primary \setminus majorNodes$
```

```
                 IN   possiblePrimary ∪ {i}
          add voted nodes into secondaries
          ∧ Secondary' = LET possibleSecondary ≜ Secondary ∪ majorNodes
                          IN   possibleSecondary \ {i}
          ∧ CurrentTerm' = [index ∈ Server ↦ IF index ∈ (majorNodes ∪ {i})
                                               THEN CurrentTerm[i] + 1
                                               ELSE CurrentTerm[index]]
           A primary node do not have any sync source
          ∧ SyncSource' = [SyncSource EXCEPT ![i] = Nil]
          ∧ UNCHANGED ⟨Oplog, Store, Ct, Ot, ServerMsg, Pt, Cp, State, ReadyToServe⟩

AdvanceCp ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Primary :
       LET newCp ≜
            LET quorumAgree ≜ QuorumAgreeInSameTerm(State[s])
            IN   IF Cardinality(quorumAgree) > 0
                    THEN LET QuorumSet ≜ CHOOSE i ∈ quorumAgree : TRUE
                             serverInQuorum ≜ CHOOSE j ∈ QuorumSet : TRUE
                             termOfQuorum ≜ State[s][serverInQuorum].term
                             StateSet ≜ {[p ↦ State[s][j].p, l ↦ State[s][j].l] : j ∈ QuorumSet}
                             newCommitPoint ≜ HLCMinSet(StateSet)
                             oldCommitPoint  ≜ [p ↦ Cp[s].p, l ↦ Cp[s].l]
                              newCp must be greater than current Cp for primary to advance it
                         IN   IF termOfQuorum = CurrentTerm[s] ∧ HLCLt(oldCommitPoint, newCommit
                                  THEN [p ↦ newCommitPoint.p, l ↦ newCommitPoint.l, term ↦ termOf
                               ELSE  Cp[s]
                 ELSE  Cp[s]
       IN   Cp' = [Cp EXCEPT ![s] = newCp]
    ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ct, Ot,
                   ServerMsg,  Pt, State, CurrentTerm, ReadyToServe, SyncSource⟩

 heartbeatoplogOtstore
ServerTakeHeartbeat ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Server :
        ∧ Len(ServerMsg[s]) ≠ 0   message channel is not empty
        ∧ ServerMsg[s][1].type = "heartbeat"
        ∧ CurrentTerm[s] = ServerMsg[s][1].term
        ∧ Ct' = [Ct EXCEPT ![s] = HLCMax(Ct[s], ServerMsg[s][1].ct)]
        ∧ State' =
           LET newState ≜ [
                   p ↦ ServerMsg[s][1].aot.p,
                   l  ↦ ServerMsg[s][1].aot.l,
                   term ↦ ServerMsg[s][1].term
```

```
              ]
        IN  LET SubHbState ≜ State[s]
                hb ≜ [SubHbState EXCEPT ![ServerMsg[s][1].s] = newState]
            IN  [State EXCEPT ![s] = hb]
   ∧ Cp′ = LET newcp ≜
              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 ≜ CHOOSE j ∈ QuorumSet : TRUE
                                     termOfQuorum ≜ State[s][serverInQuorum].term
                                     StateSet ≜ {[p ↦ State[s][j].p, l ↦ State[s][j].l] : j ∈ QuorumSet
                                     newCommitPoint ≜ HLCMinSet(StateSet)
                                 IN  IF termOfQuorum = CurrentTerm[s]
                                          THEN
                                              [p ↦ newCommitPoint.p, l ↦ newCommitPoint.l, term ↦
                                        ELSE Cp[s]
                            ELSE Cp[s]
              secondary node: update mcp
                ELSE IF LET msgCP ≜ [p ↦ ServerMsg[s][1].cp.p, l ↦ ServerMsg[s][1].cp.l]IN
                          ∧ ¬HLCLt(msgCP, Cp[s])
                          ∧ ¬HLCLt(Ot[s], msgCP)
                          The term of cp must equal to the CurrentTerm of that node to advance it
                          ∧ ServerMsg[s].cp.term = CurrentTerm[s]
                     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 newState ≜ [
                p ↦ ServerMsg[s][1].aot.p,
                l ↦ ServerMsg[s][1].aot.l,
                term ↦ ServerMsg[s][1].term
            ]
```

```
          IN   LET SubHbState ≜ State[s]
                   hb ≜ [SubHbState EXCEPT ![ServerMsg[s][1].s] = newState]
               IN  [State EXCEPT ![s] = hb]
      ∧ Cp′ = LET newcp ≜
                 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 ≜ CHOOSE j ∈ QuorumSet : TRUE
                                        termOfQuorum ≜ State[s][serverInQuorum].term
                                        StateSet ≜ {[p ↦ State[s][j].p, l ↦ State[s][j].l] : j ∈ QuorumSet
                                        newCommitPoint ≜ HLCMinSet(StateSet)
                                    IN  IF termOfQuorum = CurrentTerm[s]
                                             THEN
                                                 [p ↦ newCommitPoint.p, l ↦ newCommitPoint.l, term ↦
                                          ELSE  Cp[s]
                           ELSE  Cp[s]
                 secondary node: update mcp
                     ELSE  IF LET msgCP ≜ [p ↦ ServerMsg[s][1].cp.p, l ↦ ServerMsg[s][1].cp.l]IN
                                ∧ ¬HLCLt(msgCP, Cp[s])
                                ∧ ¬HLCLt(Ot[s], msgCP)
                           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))
    ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ot,
                 Pt, ReadyToServe, SyncSource⟩

NTPSync ≜   simplify NTP protocal
    ∧ ReadyToServe > 0
    ∧ Pt′ = [s ∈ Server ↦ MaxPt]
    ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ct, Ot,
                 ServerMsg, Cp, State, CurrentTerm, ReadyToServe, SyncSource⟩

AdvancePt ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Server :
           ∧ s ∈ Primary                              for simplicity
           ∧ Pt[s] ≤ PtStop
           ∧ Pt′ = [Pt EXCEPT ![s] = @ + 1]  advance physical time
           ∧ BroadcastHeartbeat(s)                    broadcast heartbeat periodly
        ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ct, Ot, State,
                        Cp, CurrentTerm, ReadyToServe, SyncSource⟩

Replicate oplog from node j to node i, and update related structures accordingly
Replicate ≜
    ∧ ReadyToServe > 0
    ∧ ∃ i, j ∈ Server :
        ∧ CanSyncFrom(i, j)  i can sync from j only need not to rollback
        ∧ i ∈ Secondary
        ∧ ReplicateOplog(i, j) ≠ ⟨⟩
        ∧ Oplog′ = [Oplog EXCEPT ![i] = @ ∘ ReplicateOplog(i, j)]
        ∧ Store′  = [Store EXCEPT ![i]  =  Store[j]]
        ∧ Ct′  = [Ct EXCEPT ![i] = HLCMax(Ct[i], Ct[j])]  update Ct[i]
        ∧ Ot′  = [Ot EXCEPT ![i] = HLCMax(Ot[i], Ot[j])]  update Ot[i]
        ∧ Cp′ = [Cp EXCEPT ![i] = HLCMax(Cp[i], Cp[j])]  update Cp[i]
        ∧ CurrentTerm′ = [CurrentTerm EXCEPT ![i] = Max(CurrentTerm[i], CurrentTerm[j])]  update Curren
        ∧ State′ =
            LET newState ≜ [
                       p ↦ Ot[j].p,
                       l  ↦ Ot[j].l,
                       term ↦ CurrentTerm[j]
                   ]
            IN   LET SubHbState ≜ State[i]
                         hb ≜ [SubHbState EXCEPT ![j] = newState]
                   IN   [State EXCEPT ![i] = hb]  update j's state i knows
        ∧ LET msg ≜ [type ↦ "update_position", s ↦ i, aot ↦ Ot′[i], ct ↦ Ct′[i], cp ↦ Cp′[i], term ↦ Curr
           IN   ServerMsg′ = [ServerMsg EXCEPT ![j] = Append(ServerMsg[j], msg)]
        ∧ SyncSource′ = [SyncSource EXCEPT ![i] = j]
      ∧ CalState′ =  [CalState  EXCEPT ![i] = CalState[j]]
        ∧ UNCHANGED ⟨Primary, Secondary, Pt, ReadyToServe⟩

Rollback i's oplog and recover it to j's state
Recover to j's state immediately to prevent internal client request
RollbackAndRecover ≜
    ∧ ReadyToServe > 0
    ∧ ∃ i, j ∈ Server :
        ∧ i ∈ Secondary
        ∧ CanRollback(i, j)
        ∧ LET cmp ≜ RollbackCommonPoint(i, j) IN
           LET commonLog ≜ SubSeq(Oplog[i], 1, cmp)
                  appendLog   ≜ 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 $newStatei \triangleq [$
                $p \mapsto Ot'[i].p,$
                $l \mapsto Ot'[j].l,$
                $term \mapsto CurrentTerm'[i]$
            $]$
            $newStatej \triangleq [$
                $p \mapsto Ot[j].p,$
                $l \mapsto Ot[j].l,$
                $term \mapsto CurrentTerm[j]$
            $]$
        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  $[State \text{ EXCEPT } ![i] = hb1]$
      $\land$ 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  $ServerMsg' = [ServerMsg \text{ EXCEPT } ![j] = Append(ServerMsg[j], msg)]$
      $\land SyncSource' = [SyncSource \text{ EXCEPT } ![i] = j]$
      $\land$ UNCHANGED $\langle Primary, Secondary, Pt, ReadyToServe \rangle$

$ClientRequest \triangleq$
   $\land ReadyToServe > 0$
   $\land \exists s \in Server, k \in Key, v \in Value :$
      $\land s \in Primary$
      $\land Tick(s)$
      $\land Ot' = [Ot \text{ EXCEPT } ![s] = Ct'[s]]$
      $\land Store' = [Store \text{ EXCEPT } ![s][k] = v]$
      $\land Oplog' =$ LET $entry \triangleq [k \mapsto k, v \mapsto v, ot \mapsto Ot'[s], term \mapsto CurrentTerm[s]]$
                           $newLog \triangleq Append(Oplog[s], entry)$
                       IN  $[Oplog \text{ EXCEPT } ![s] = newLog]$
      $\land State' =$
        LET $newState \triangleq [$
                $p \mapsto Ot'[s].p,$
                $l \mapsto Ot'[s].l,$
                $term \mapsto CurrentTerm[s]$
            $]$
        IN  LET $SubHbState \triangleq State[s]$
                $hb \triangleq [SubHbState \text{ EXCEPT } ![s] = newState]$
            IN  $[State \text{ EXCEPT } ![s] = hb]$ update $i$'s state
```

$\quad\quad \land$ UNCHANGED $\langle Primary,\ Secondary,\ ServerMsg,$
$\quad\quad\quad\quad Pt,\ Cp,$
$\quad\quad\quad\quad CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

Next state for all configurations

$Next \triangleq \lor Replicate$
$\quad\quad \lor AdvancePt$
$\quad\quad \lor AdvanceCp$
$\quad\quad \lor ServerTakeHeartbeat$
$\quad\quad \lor ServerTakeUpdatePosition$
$\quad\quad \lor Stepdown$
$\quad\quad \lor RollbackAndRecover$
$\quad\quad \lor TurnOnReadyToServe$
$\quad\quad \lor ElectPrimary$
$\quad\quad \lor ClientRequest$
$\quad\quad \lor NTPSync$

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

Properties to check?

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

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

$LastTermsEquivalentImplyPrefixes \triangleq$
$\quad \forall\, i,\ j \in Server :$
$\quad\quad LogTerm(i,\ Len(Oplog[i])) = LogTerm(j,\ Len(Oplog[j])) \Rightarrow$
$\quad\quad IsLogPrefix(i,\ j) \lor IsLogPrefix(j,\ i)$

Check whether terms are incremented monotonicly (from Will

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

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

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

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

$NeverRollbackCommonPoint \triangleq$
$\quad \exists\, i,\ j \in Server : CanRollback(i,\ j) \Rightarrow$
$\quad\quad$ LET $commonPoint \triangleq RollbackCommonPoint(i,\ j)$
$\quad\quad\quad\quad lastOplog \triangleq Oplog[i][commonPoint]$
$\quad\quad$ 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*

$$\begin{aligned}
&TwoPrimariesInSameTerm \triangleq\\
&\quad \exists\, i, j \in Server :\\
&\qquad \land\ i \neq j\\
&\qquad \land\ CurrentTerm[i] = CurrentTerm[j]\\
&\qquad \land\ i \in Primary\\
&\qquad \land\ j \in Primary
\end{aligned}$$

$$NoTwoPrimariesInSameTerm \triangleq \neg TwoPrimariesInSameTerm$$

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

$$\begin{aligned}
&SyncSourceCycleTwoNode \triangleq\\
&\quad \exists\, s, t \in Server :\\
&\qquad \land\ s \neq t\\
&\qquad \land\ SyncSource[s] = t\\
&\qquad \land\ SyncSource[t] = s
\end{aligned}$$

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

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

$$\begin{aligned}
&SyncSourcePath(i, j) \triangleq\\
&\quad \exists\, p \in SyncSourcePaths :\\
&\qquad \land\ Len(p) > 1\\
&\qquad \land\ p[1] = i\\
&\qquad \land\ p[Len(p)] = j
\end{aligned}$$

$$\begin{aligned}
&SyncSourceCycle \triangleq\\
&\quad \exists\, s \in Server : SyncSourcePath(s, s)
\end{aligned}$$

$$\begin{aligned}
&NonTrivialSyncCycle \triangleq SyncSourceCycle \land \neg SyncSourceCycleTwoNode\\
&NoNonTrivialSyncCycle \triangleq \neg NonTrivialSyncCycle
\end{aligned}$$

\ ∗ Modification *History*
\ ∗ Last modified *Wed* May 04 15:41:10 *CST* 2022 by *dh*
\ ∗ Created *Mon Apr* 18 11:38:53 *CST* 2022 by *dh*