```
\ * 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
```

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

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 $x \in s : \forall y \in s : x \geq y$
$HLCMinSet(s) \triangleq$ CHOOSE $x \in s : \forall y \in s : \neg HLCLt(y, x)$

 commit point
 RECURSIVE $AddState(\_, \_, \_)$
 $AddState(new, state, index) \triangleq$
   IF $index = 1 \land HLCLt(new, state[1])$
      THEN $\langle new\rangle \circ state \setminus *$ 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  $Pt[x]$

$Tick(s) \triangleq Ct' =$ IF $Ct[s].p \geq Pt[s]$
```

```
                    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.
                                    $\land\ \lor \forall s \in Q : states[s].p > 0$
                                    $\quad\ \lor \land \forall s \in Q : states[s].p = 0$
                                    $\qquad\ \land \forall s \in Q : states[s].l > 0$
                                    Make sure every applied entry in quorum has the same term.
                                    $\land \forall s, t \in Q :$
                                    $\quad s \neq t \Rightarrow states[s].term = states[s].term\}$
  IN  IF  $quorums = \{\}$ THEN $Nil$
      ELSE  CHOOSE $x \in quorums$ : TRUE
      ELSE  $quorums$
    IN    $quorums$
```

Init Part

```
$InitPrimary \triangleq Primary = \{\text{CHOOSE } s \in Server : \text{TRUE}\}$
$InitSecondary \triangleq Secondary = Server \setminus Primary$
$InitOplog \triangleq Oplog = [s \in Server \mapsto \langle\rangle]$
$InitStore \triangleq Store = [n \in Server \cup Client \mapsto [k \in Key \mapsto Nil]]$
$InitCt \triangleq Ct = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitOt \triangleq Ot = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitServerMsg \triangleq ServerMsg = [s \in Server \mapsto \langle\rangle]$
$InitPt \triangleq Pt = [s \in Server \mapsto 1]$
$InitCp \triangleq Cp = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitCalState \triangleq CalState = [s \in Server \mapsto CreateState(Cardinality(Server), \langle\rangle)]$
                                   create initial $state(for\ calculate)$
$InitState \triangleq State = [s \in Server \mapsto [s0 \in Server \mapsto$
                                            $[p \mapsto 0, l \mapsto 0, term \mapsto 0]]]$
$InitCurrentTerm \triangleq CurrentTerm = [s \in Server \mapsto 0]$
$InitReadyToServe \triangleq ReadyToServe = 0$
$InitSyncSource \triangleq SyncSource = [s \in Server \mapsto Nil]$

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

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

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

$AdvanceCp\ \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, i \in Primary :$
```

```
        LET quorumAgree ≜ QuorumAgreeInSameTerm(State[i])IN
            ∧ quorumAgree ≠ Nil
            ∧ LET serverInQuorum ≜ CHOOSE s ∈ quorumAgree : TRUE
                  termOfQuorum ≜ State[i][serverInQuorum][3]
                  newCommitPoint ≜ HLCMinSet({[p ↦ State[i][s][1], l ↦ State[i][s][2]] : s ∈ quorumAgre
              IN  ∧ termOfQuorum = CurrentTerm[i]
                  ∧ LET newCP ≜ [p ↦ newCommitPoint.p, l ↦ newCommitPoint.l, term ↦ termOfQuor
                        Cp′ = [Cp EXCEPT ![i] = 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"
        ∧ 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 serverInQuorum ≜ CHOOSE i ∈ quorumAgree : TRUE
                             termOfQuorum ≜ State[s][serverInQuorum].term
                             newCommitPoint ≜ HLCMinSet({[p ↦ State[s][j].p, l ↦ State[s][j].l] : j ∈ quorumAgree})
                      IN  IF termOfQuorum = CurrentTerm[s]
                             THEN [p ↦ newCommitPoint.p, l ↦ newCommitPoint.l, term ↦ termOfQuorum]
                             ELSE Cp[s]
                                THEN [p ↦ 2, l ↦ 2, term ↦ 2]
                                ELSE Cp[s]
\ *                        ∧ quorumAgree ≠ Nil
                        ∧ Cardinality(quorumAgree) > 0
\ *                        ∧ LET serverInQuorum ≜ CHOOSE i ∈ quorumAgree: TRUE
\ *                              termOfQuorum ≜ State[s][serverInQuorum].term
\ *                              newCommitPoint ≜ HLCMinSet({[p ↦ State[s][i].p, l ↦ State[s][i].l] : i ∈ quorumAgree})
\ *                                IN   IF  termOfQuorum = CurrentTerm[s]
                    ∧ LET serverInQuorum ≜ CHOOSE i ∈ quorumAgree: TRUE
                          termOfQuorum ≜ State[s][serverInQuorum].term
                        IN   IF  termOfQuorum = CurrentTerm[s]
\ *                                   THEN  [ p ↦ newCommitPoint.p, l ↦ newCommitPoint.l, term ↦ termOfQuorum ]
                              THEN  [p ↦ 5, l ↦ 5, term ↦ termOfQuorum]
                              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]
         ∧ 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 ∃ serverInQuorum ∈ quorumAgree :
                                    LET termOfQuorum ≜ State[s][serverInQuorum].term
                               StateSet ≜ {[p ↦ State[s][j].p, l ↦ State[s][j].l] : j ∈ quorumAgree}
                               newCommitPoint ≜ HLCMinSet(StateSet)
                                        newCommitPoint ≜ [p ↦ State[s][serverInQuorum].p, l ↦ State[s][se
                                IN   IF termOfQuorum = CurrentTerm[s]
```

```
                                        THEN $[p \mapsto newCommitPoint.p,\ l \mapsto newCommitPoint.l,\ term \mapsto term$
                                       ELSE $Cp[s]$
                                ELSE $Cp[s]$
                         secondary node: update mcp
                            ELSE IF LET $msgCP \triangleq [p \mapsto ServerMsg[s][1].cp.p,\ l \mapsto ServerMsg[s][1].cp.l]$ IN
                                 $\land \neg HLCLt(msgCP,\ Cp[s])$
                                 $\land \neg HLCLt(Ot[s],\ msgCP)$
                                THEN $ServerMsg[s][1].cp$
                                ELSE $Cp[s]$
                         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 $![SyncSource[s]] = Appenc$
                                     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 $\langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,$
                  $ServerMsg,\ Cp,\ State,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

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

 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 $![i] = Store[j]]$
    ∧ 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 EXCEPT ![i] = commonLog ∘ appendLog]
    ∧ CurrentTerm′ = [CurrentTerm EXCEPT ![i] = Max(CurrentTerm[i], CurrentTerm[j])]   update Curren
    ∧ 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]
    ∧ State′ =
       LET newStatei ≜ [
                p ↦ Ot′[i].p,
                l ↦ Ot′[j].l,
                term ↦ CurrentTerm′[i]
            ]
            newStatej ≜ [
                p ↦ Ot[j].p,
                l ↦ Ot[j].l,
                term ↦ CurrentTerm[j]
            ]
        IN  LET SubHbState ≜ State[i]
                hb ≜ [SubHbState EXCEPT ![i] = newStatei]  update i's self state (used in mcp computation
                hb1 ≜ [hb EXCEPT ![j] = newStatej]  update j's state i knows
            IN  [State EXCEPT ![i] = hb1]
     ∧ 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]
     ∧ UNCHANGED ⟨Primary, Secondary, Pt, ReadyToServe⟩

ClientRequest ≜
   ∧ ReadyToServe > 0
   ∧ ∃ s ∈ Server, k ∈ Key, v ∈ Value :
      ∧ s ∈ Primary
      ∧ Tick(s)
      ∧ Ot' = [Ot EXCEPT ![s] = Ct'[s]]
      ∧ Store' = [Store EXCEPT ![s][k] = v]
      ∧ Oplog' = LET entry ≜ [k ↦ k, v ↦ v, ot ↦ Ot'[s], term ↦ CurrentTerm[s]]
                     newLog ≜ Append(Oplog[s], entry)
                 IN  [Oplog EXCEPT ![s] = newLog]
      ∧ State' =
         LET newState ≜ [
                p ↦ Ot'[s].p,
                l ↦ Ot'[s].l,
                term ↦ CurrentTerm[s]
             ]
         IN  LET SubHbState ≜ State[s]
                 hb ≜ [SubHbState EXCEPT ![s] = newState]
             IN  [State EXCEPT ![s] = hb]  update i's state
      ∧ UNCHANGED ⟨Primary, Secondary, ServerMsg,
                   Pt, Cp,
                   CurrentTerm, ReadyToServe, SyncSource⟩

Next state for all configurations
Next ≜ ∨ Replicate
       ∨ AdvancePt
       ∨ ServerTakeHeartbeat
       ∨ ServerTakeUpdatePosition
       ∨ Stepdown
       ∨ RollbackAndRecover
       ∨ TurnOnReadyToServe
       ∨ ElectPrimary
       ∨ ClientRequest
       ∨ NTPSync

Spec ≜ Init ∧ □[Next]_vars
```

Properties to check?

$$\begin{array}{l}
\mathit{IsLogPrefix}(i,\, j) \;\triangleq \\
\quad \land\; \mathit{Len}(\mathit{Oplog}[i]) \leq \mathit{Len}(\mathit{Oplog}[j]) \\
\quad \land\; \mathit{Oplog}[i] = \mathit{SubSeq}(\mathit{Oplog}[j],\, 1,\, \mathit{Len}(\mathit{Oplog}[i]))
\end{array}$$

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

$$\begin{array}{l}
\mathit{LastTermsEquivalentImplyPrefixes} \;\triangleq \\
\quad \forall\, i,\, j \in \mathit{Server} : \\
\qquad \mathit{LogTerm}(i,\, \mathit{Len}(\mathit{Oplog}[i])) = \mathit{LogTerm}(j,\, \mathit{Len}(\mathit{Oplog}[j])) \Rightarrow \\
\qquad \mathit{IsLogPrefix}(i,\, j) \lor \mathit{IsLogPrefix}(j,\, i)
\end{array}$$

Check whether terms are incremented monotoniclly (from Will

$$\begin{array}{l}
\mathit{TermsMonotonic} \;\triangleq \\
\quad \Box[\forall\, s \in \mathit{Server} : \mathit{CurrentTerm}'[s] \geq \mathit{CurrentTerm}[s]]_{vars}
\end{array}$$

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

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

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

$$\begin{array}{l}
\mathit{NeverRollbackCommonPoint} \;\triangleq \\
\quad \exists\, i,\, j \in \mathit{Server} : \mathit{CanRollback}(i,\, j) \Rightarrow \\
\qquad \textsc{let}\;\; \mathit{commonPoint} \;\triangleq\; \mathit{RollbackCommonPoint}(i,\, j) \\
\qquad \phantom{\textsc{let}}\;\; \mathit{lastOplog} \;\triangleq\; \mathit{Oplog}[i][\mathit{commonPoint}] \\
\qquad \textsc{in}\;\;\; \mathit{HLCLt}(\mathit{Cp}[i],\, \mathit{lastOplog.ot})
\end{array}$$

Eventually *log* correctness (from Will

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

(from *RaftMongo*

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

$$\mathit{NoTwoPrimariesInSameTerm} \;\triangleq\; \neg \mathit{TwoPrimariesInSameTerm}$$

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

$$\begin{array}{l}
\mathit{SyncSourceCycleTwoNode} \;\triangleq \\
\quad \exists\, s,\, t \in \mathit{Server} : \\
\qquad \land\; s \neq t \\
\qquad \land\; \mathit{SyncSource}[s] = t \\
\qquad \land\; \mathit{SyncSource}[t] = s
\end{array}$$

$$\mathit{BoundedSeq}(s,\ n)\ \triangleq\ [1 \mathrel{..} n \rightarrow s]$$

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

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

$$\begin{array}{l}
\mathit{SyncSourceCycle}\ \triangleq \\
\quad \exists\, s \in \mathit{Server} : \mathit{SyncSourcePath}(s,\ s)
\end{array}$$

$$\begin{array}{l}
\mathit{NonTrivialSyncCycle}\ \triangleq\ \mathit{SyncSourceCycle} \land \neg\mathit{SyncSourceCycleTwoNode} \\
\mathit{NoNonTrivialSyncCycle}\ \triangleq\ \neg\mathit{NonTrivialSyncCycle}
\end{array}$$

\ * Modification *History*

\ * Last modified *Fri Apr* 29 14:33:41 *CST* 2022 by *dh*

\ * Created *Mon Apr* 18 11:38:53 *CST* 2022 by *dh*