```
──────────────────── MODULE TunableMongoDB_RBK ────────────────────
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
          OpTimes,                     op count at most
          PtStop,                      max physical time
          WriteNumber,                 Para: writeConcern number → should be set even when w:maj
          ReadConcern,                 Para: read concern
          ReadPreference,              Para: read preference
          WriteConcern                 Para: write concern

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,          sync source of server node s
           Following are the Tunable related vars
          BlockedClient,       BlockedClient: Client operations in progress
          BlockedThread,       BlockedThread: blocked user thread and content
          History,             History[c]: History sequence at client c
          InMsgc,              InMsgc[c]: the channel of messages at client c ∈ Client
          InMsgs,              InMsgc[s]: the channel of messages at server s ∈ Server
          OpCount,             OpCount[c]: op count for client c
          SnapshotTable        SnapshotTable[s] : snapshot mapping table at server s

├───────────────────────────────────────────────────────────────────┤
 group related vars to optimize code
electionVars ≜ ⟨Primary, Secondary⟩                    vars that are related to election
storageVars ≜ ⟨Oplog, Store⟩                           vars that are related to storage
messageVar ≜ ⟨ServerMsg⟩                               var that is related to message
servernodeVars ≜ ⟨Ot, SyncSource, SnapshotTable⟩               vars that each server node holds for itself
learnableVars ≜ ⟨Ct, State, Cp, CurrentTerm⟩            vars that must learn from msgs
timeVar ≜ ⟨Pt⟩                                          var that is used for timing
functionalVar ≜ ⟨ReadyToServe⟩                          var that is used for some extra function
```

```
clientnodeVars ≜ ⟨History, OpCount⟩

tmessageVars ≜ ⟨InMsgc, InMsgs⟩
tfunctionalVars ≜ ⟨BlockedClient, BlockedThread⟩

serverVars ≜ ⟨electionVars, storageVars, messageVar, servernodeVars, learnableVars, timeVar, functionalV
tunableVars ≜ ⟨BlockedClient, BlockedThread, History, InMsgc, InMsgs, OpCount, SnapshotTable⟩
├──────────────────────────────────────────────────────────────────────────┤
ASSUME Cardinality(Client) ≥ 1   at least one client
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
ASSUME ReadConcern ∈ {"local", "majority", "linearizable"}
ASSUME WriteConcern ∈ {"zero", "num", "majority"}
ASSUME ReadPreference ∈ {"primary", "secondary"}
├──────────────────────────────────────────────────────────────────────────┤
 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]
HLCMinSet(s) ≜ CHOOSE x ∈ s : ∀ y ∈ s : ¬HLCLt(y, x)
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, InMsgc,
          InMsgs, ServerMsg, BlockedClient, BlockedThread,
          OpCount, Pt, Cp, State, SnapshotTable,
          History, CurrentTerm, ReadyToServe, SyncSource⟩
 snapshot helpers
RECURSIVE SelectSnapshot_rec(_, _, _)
SelectSnapshot_rec(stable, cp, index) ≜
    IF HLCLt(cp, stable[index].ot) THEN stable[index − 1].store
     ELSE IF index = Len(stable) THEN stable[index].store
     ELSE SelectSnapshot_rec(stable, cp, index + 1)

SelectSnapshot(stable, cp) ≜ SelectSnapshot_rec(stable, cp, 1)

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

$$MaxVal(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s : x \geq y$$

clock

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

```
$Tick(s) \triangleq Ct' = \text{IF } Ct[s].p \geq Pt[s] \text{ THEN } [Ct \text{ EXCEPT } ![s] = [p \mapsto @.p, l \mapsto @.l + 1]]$
                                        $\text{ELSE } [Ct \text{ EXCEPT } ![s] = [p \mapsto Pt[s], l \mapsto 0]]$
```

heartbeat − Only *Primary* node sends heartbeat once advance pt

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

Can node i sync from node j?

```
$CanSyncFrom(i, j) \triangleq$
   $\land Len(Oplog[i]) < Len(Oplog[j])$
   $\land LastTerm(i) = LogTerm(j, Len(Oplog[i]))$
```

Oplog entries needed to replicate from j to i

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

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 ≜ {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

ReplicatedServers(states, ot) ≜
   LET serverSet ≜ {subServers ∈ SUBSET (Server) : ∀ s ∈ subServers : LET stateTime ≜ [p ↦ states[s].p,
                                                                       IN  ¬HLCLt(stateTime, ot)}
   IN  CHOOSE maxSet ∈ serverSet : ∀ otherSet ∈ serverSet : Cardinality(otherSet) < Cardinality(maxSet)

 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 ≜ 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 ↦ termOfQuorum]
                    ELSE  Cp[s]
         ELSE  Cp[s]
    secondary node: update mcp
    ELSE IF Len(ServerMsg[s]) ≠ 0 THEN
            LET msgCP ≜ [p ↦ ServerMsg[s][1].cp.p, l ↦ ServerMsg[s][1].cp.l]IN
            IF ∧ ¬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][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
$InitSecondary \triangleq Secondary = Server \setminus \{Primary\}$
$InitOplog \triangleq Oplog = [s \in Server \mapsto \langle\rangle]$
$InitStore \triangleq Store = [n \in Server \cup Client \mapsto [k \in Key \mapsto Nil]]$
$InitCt \triangleq Ct = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitOt \triangleq Ot = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitInMsgc \triangleq InMsgc = [c \in Client \mapsto \langle\rangle]$
$InitInMsgs \triangleq InMsgs = [s \in Server \mapsto \langle\rangle]$
$InitServerMsg \triangleq ServerMsg = [s \in Server \mapsto \langle\rangle]$
$InitBlockedClient \triangleq BlockedClient = \{\}$
$InitBlockedThread \triangleq BlockedThread = [s \in Client \mapsto Nil]$
$InitOpCount \triangleq OpCount = [c \in Client \mapsto OpTimes]$
$InitPt \triangleq Pt = [s \in Server \mapsto 1]$
$InitCp \triangleq Cp = [n \in Server \cup Client \mapsto [p \mapsto 0, l \mapsto 0]]$
$InitState \triangleq State = [s \in Server \mapsto [s0 \in Server \mapsto$
                                       $[p \mapsto 0, l \mapsto 0, term \mapsto 0]]]$
$InitSnap \triangleq SnapshotTable = [s \in Server \mapsto \langle[ot \mapsto [p \mapsto 0, l \mapsto 0],$
                                           $store \mapsto [k \in Key \mapsto Nil]]\rangle]$
$InitHistory \triangleq History = [c \in Client \mapsto \langle\rangle]$  History operation seq is empty
$InitCurrentTerm \triangleq CurrentTerm = [s \in Server \mapsto 0]$
$InitReadyToServe \triangleq ReadyToServe = 0$
$InitSyncSource \triangleq SyncSource = [s \in Server \mapsto Nil]$

$Init \triangleq$
    $\land InitPrimary \land InitSecondary \land InitOplog \land InitStore \land InitCt$
    $\land InitOt \land InitPt \land InitCp \land InitInMsgc \land InitInMsgs$
    $\land InitServerMsg \land InitBlockedClient \land InitBlockedThread \land InitOpCount$
    $\land InitState \land InitSnap \land InitHistory \land InitCurrentTerm \land InitReadyToServe$
    $\land InitSyncSource$
```

Next *State* Actions

Replication Protocol: possible actions

snapshot periodly

```
$Snapshot \triangleq$
```

```
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Server :
          SnapshotTable′ = [SnapshotTable EXCEPT ![s] =
                            Append(@, [ot ↦ Ot[s], store ↦ Store[s]])]
                              create a new snapshot
    ∧ UNCHANGED ⟨serverVars, InMsgc, InMsgs, BlockedClient, BlockedThread, OpCount, History⟩

Stepdown ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Primary :
        ∧ Primary′ = Primary \ {s}
        ∧ Secondary′ = Secondary ∪ {s}
    ∧ UNCHANGED ⟨storageVars, serverVars, Ct, messageVar, timeVar, Cp, State, CurrentTerm, functionalV

 There are majority nodes agree to elect node i to become primary
ElectPrimary ≜
    ∧ ReadyToServe > 0
    ∧ ∃ i ∈ Server : ∃ majorNodes ∈ SUBSET (Server) :
        ∧ ∀ j ∈ majorNodes : ∧ NotBehind(i, j)
                             ∧ CurrentTerm[i] ≥ CurrentTerm[j]
        ∧ IsMajority(majorNodes)
        voted nodes for i cannot be primary anymore
        ∧ Primary′ = LET possiblePrimary ≜ Primary \ 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 ⟨storageVars, Ct, Ot, messageVar, timeVar, Cp, State, functionalVar, tunableVars⟩

TurnOnReadyToServe ≜
    ∧ ReadyToServe = 0
    ∧ ∃ s ∈ Primary :
        ∧ CurrentTerm′ = [CurrentTerm EXCEPT ![s] = CurrentTerm[s] + 1]
      ∧ CurrentTerm′ = [s ∈ Server ↦ 1] ?
         ∧ ReadyToServe′ = ReadyToServe + 1
    ∧ UNCHANGED ⟨electionVars, storageVars, serverVars, Ct, messageVar, timeVar, Cp, State, tunableVars

AdvanceCp ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Primary :
        LET newCp ≜ ComputeNewCp(s)
        IN   $Cp' = [Cp$ EXCEPT $![s] = newCp]$
    $\land$ UNCHANGED $\langle electionVars, storageVars, serverVars, Ct, messageVar, timeVar, State, CurrentTerm, fu$

heartbeatoplog$Ot$store
$ServerTakeHeartbeat \triangleq$
    $\land ReadyToServe > 0$
    $\land \exists\, s \in Server :$
        $\land Len(ServerMsg[s]) \neq 0$   message channel is not empty
        $\land ServerMsg[s][1].type =$ "heartbeat"
        $\land CurrentTerm[s] = ServerMsg[s][1].term$   only consider heartbeat $msg$ in same 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 $\langle electionVars, storageVars, serverVars, timeVar, functionalVar, tunableVars\rangle$

$ServerTakeUpdatePosition \triangleq$
    $\land ReadyToServe > 0$
    $\land \exists\, s \in Server :$
        $\land Len(ServerMsg[s]) \neq 0$   message channel is not empty
        $\land ServerMsg[s][1].type =$ "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 $\langle electionVars, storageVars, serverVars, timeVar, functionalVar, tunableVars\rangle$

$NTPSync \triangleq$   simplify $NTP$ protocal
    $\land ReadyToServe > 0$
    $\land Pt' = [s \in Server \mapsto MaxPt]$
    $\land$ UNCHANGED $\langle electionVars, storageVars, serverVars, learnableVars, messageVar, functionalVar, tunabl$

$AdvancePt \triangleq$
    $\land ReadyToServe > 0$
    ∧ ∃ s ∈ Server :
          ∧ s ∈ Primary                              for simplicity
          ∧ Pt[s] ≤ PtStop
          ∧ Pt′ = [Pt EXCEPT ![s] = @ + 1]  advance physical time
          ∧ BroadcastHeartbeat(s)               broadcast heartbeat periodly
    ∧ UNCHANGED ⟨electionVars, storageVars, serverVars, learnableVars, functionalVar, tunableVars⟩

Replication
Idea: replicatecanSyncFromlogterm
SyncSource[s]SyncSourceUpdatePosition
UpdatePositionactiontypeupdatePosition
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 ≜ GetNewState(i, j, Ot[j].p, Ot[j].l, CurrentTerm[j])
                   IN  [State EXCEPT ![i] = newState]  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]
    ∧ UNCHANGED ⟨electionVars, timeVar, functionalVar, tunableVars⟩

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′[i].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 ⟨electionVars, timeVar, functionalVar, tunableVars⟩

  Tunable Protocol: Server Actions
├──────────────────────────────────────────────────────────────────────┤
  Server Get
ServerGetReply_sleep ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Server :
        ∧ Len(InMsgs[s]) ≠ 0
        ∧ InMsgs[s][1].op = "get"
        ∧ IF InMsgs[s][1].rc = "linearizable"
            THEN  ∧ s ∈ Primary
                  ∧ Tick(s)   advance cluster time
                  ∧ Oplog′ = [Oplog EXCEPT ![s] = Append(@, ⟨Nil, Nil, Ct′[s]⟩)]
                     append noop operation to oplog[s]
                  ∧ Ot′ = [Ot EXCEPT ![s] =  Ct′[s]]
                     advance the last applied operation time Ot[s]
                  ∧ State′ = LET newState ≜ GetNewState(s, s, Ot′[s].p, Ot′[s].l, CurrentTerm[s])
                             IN  [State EXCEPT ![s] = newState]        update primary state
                  ∧ InMsgs′ = [InMsgs EXCEPT ![s] = Tail(@)]
                  ∧ BlockedThread′ = [BlockedThread EXCEPT ![InMsgs[s][1].c] =
                                      [type ↦ "read", rc ↦ InMsgs[s][1].rc, ot ↦ Ct′[s], s ↦ s,
                                       k ↦ InMsgs[s][1].k, v ↦ Store[s][InMsgs[s][1].k]]]
                     add the user thread to BlockedThread[c]
           ELSE  ∧ Ct′ = [Ct EXCEPT ![s] = HLCMax(Ct[s], InMsgs[s][1].ct)]  rc = local or major
                 ∧ BlockedThread′ = [BlockedThread EXCEPT ![InMsgs[s][1].c] =
                           [type ↦ "read", rc ↦ InMsgs[s][1].rc, s ↦ s, k ↦ InMsgs[s][1].k, ot ↦ InMsgs[s][
                 ∧ Oplog′ = Oplog
                 ∧ Ot′ = Ot
                 ∧ State′ = State
        ∧ InMsgs′ = [InMsgs EXCEPT ![s] = Tail(@)]
    ∧ UNCHANGED ⟨electionVars, functionalVar, Cp, CurrentTerm, messageVar, SyncSource, Store, timeVar
                 InMsgc, BlockedClient, clientnodeVars, SnapshotTable⟩
```

```
$ServerGetReply\_wake \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, c \in Client :$
        $\land\ BlockedThread[c] \neq Nil$
        $\land\ BlockedThread[c].type = \text{“read”}$
        $\land\ \text{IF}\ BlockedThread[c].rc = \text{“linearizable”}$
            $\text{THEN}\ \land \lnot HLCLt(Cp[BlockedThread[c].s],\ BlockedThread[c].ot)$ wait until $cp[s] \geq$ target $ot$
                  $\land\ InMsgc' = [InMsgc\ \text{EXCEPT}\ ![c] = Append(@,\ [op \mapsto \text{“get”},$
                                   $k \mapsto BlockedThread[c].k,\ v \mapsto BlockedThread[c].v,$
                                   $ct \mapsto Ct[BlockedThread[c].s],\ ot \mapsto BlockedThread[c].ot])]$
          $\text{ELSE}\ \land \lnot HLCLt(Ot[BlockedThread[c].s],\ BlockedThread[c].ot)$ wait until $Ot[s] \geq$ target $ot$
                $\land\ \text{IF}\ BlockedThread[c].rc = \text{“local”}$
                    $\text{THEN}\ InMsgc' = [InMsgc\ \text{EXCEPT}\ ![c] = Append(@,\ [op \mapsto \text{“get”},\ k \mapsto BlockedThread[c]$
                                     $v \mapsto Store[BlockedThread[c].s][BlockedThread[c].k],$
                                     $ct \mapsto Ct[BlockedThread[c].s],\ ot \qquad \mapsto Ot[BlockedThread[c].s]])]$
                    $\text{ELSE}\ \ InMsgc' = [InMsgc\ \text{EXCEPT}\ ![c] = Append(@,\ [op \mapsto \text{“get”},\ k \mapsto BlockedThread[c].k$
                                    $v \mapsto SelectSnapshot(SnapshotTable[BlockedThread[c].s],\ Cp[BlockedThre$
                                    $ct \mapsto Ct[BlockedThread[c].s],\ ot \mapsto Cp[BlockedThread[c].s]])]$
        $\land\ BlockedThread' = [BlockedThread\ \text{EXCEPT}\ ![c] = Nil]$
    $\land\ \text{UNCHANGED}\ \langle serverVars,\ clientnodeVars,\ BlockedClient,\ InMsgs,\ SnapshotTable \rangle$

Server Put serveroplog
$ServerPutReply\_sleep \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, s \in Primary :$
        $\land\ Len(InMsgs[s]) \neq 0$
        $\land\ InMsgs[s][1].op = \text{“put”}$
        $\land\ Tick(s)$
        $\land\ Ot' = [Ot\ \text{EXCEPT}\ ![s] = \ Ct'[s]]$ advance the last applied operation time $Ot[s]$
        $\land\ Store' \ = [Store\ \text{EXCEPT}\ ![s][InMsgs[s][1].k] = InMsgs[s][1].v]$ append operation to $oplog[s]$
        $\land\ Oplog' = \text{LET}\ entry \triangleq [k \mapsto InMsgs[s][1].k,\ v \mapsto InMsgs[s][1].v,$
                                   $ot \mapsto Ot'[s],\ term \mapsto CurrentTerm[s]]$
                          $newLog \triangleq Append(Oplog[s],\ entry)$
                      $\text{IN}\ \ [Oplog\ \text{EXCEPT}\ ![s] = newLog]$
        $\land\ State' \ = \text{LET}\ newState \triangleq GetNewState(s,\ s,\ Ot'[s].p,\ Ot'[s].l,\ CurrentTerm[s])$
                      $\text{IN}\ \ [State\ \text{EXCEPT}\ ![s] = newState]$        update primary state
        $\land\ \text{IF}\ InMsgs[s][1].wc = \text{“zero”}$ If w:0, do not sleep
            $\text{THEN}\ BlockedThread' = BlockedThread$
            $\text{ELSE}\ \ BlockedThread' = [BlockedThread\ \text{EXCEPT}\ ![InMsgs[s][1].c] = [type \mapsto \text{“write”},\ wc \mapsto InMsgs$
                                      $numnode \mapsto InMsgs[s][1].num,\ ot \mapsto Ot'[s],\ s \mapsto s,$
                                      $k \mapsto InMsgs[s][1].k,\ v \mapsto InMsgs[s][1].v]]$ add the user $History$ to $BlockedThre$
        $\land\ InMsgs' = [InMsgs\ \text{EXCEPT}\ ![s] = Tail(@)]$
    $\land\ \text{UNCHANGED}\ \langle electionVars,\ functionalVar,\ Cp,\ CurrentTerm,\ messageVar,\ SyncSource,\ timeVar,$
                      $InMsgc,\ BlockedClient,\ clientnodeVars,\ SnapshotTable \rangle$
```

```
$ServerPutReply\_wake \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, c \in Client :$
        $\land\ BlockedThread[c] \neq Nil$
        $\land\ BlockedThread[c].type = \text{“write”}$
        $\land$ IF $BlockedThread[c].wc = \text{“num”}$   w:$num$
            THEN LET $replicatedServers \triangleq ReplicatedServers(State[BlockedThread[c].s], BlockedThread[c].ot)$
                 IN  $Cardinality(replicatedServers) \geq BlockedThread[c].numnode$
            ELSE $\neg HLCLt(Cp[BlockedThread[c].s], BlockedThread[c].ot)$   w:majority
        $\land\ InMsgc' = [InMsgc$ EXCEPT $![c] = Append(@, [op \mapsto \text{“put”},$
                 $ct \mapsto Ct[BlockedThread[c].s], ot \mapsto BlockedThread[c].ot,$
                 $k \mapsto BlockedThread[c].k, v \mapsto BlockedThread[c].v])]$
        $\land\ BlockedThread' = [BlockedThread$ EXCEPT $![c] = Nil]$   remove blocked state
    $\land$ UNCHANGED $\langle serverVars, clientnodeVars, BlockedClient, InMsgs, SnapshotTable \rangle$
```

Tunable Protocol: *Client* Actions

Client Get

```
$ClientGetRequest \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, k \in Key, c \in Client \setminus BlockedClient, rConcern \in ReadConcern, rp \in ReadPreference :$
        $\land$ IF $rConcern = \text{“linearizable”}$   In this case, read can only be sent to primary
           THEN $\exists\, s \in Primary :$
                $InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$
                $[op \mapsto \text{“get”}, c \mapsto c, rc \mapsto rConcern, k \mapsto k, ct \mapsto Ct[c], ot \mapsto Ot[c]])]$
           ELSE IF $rp = \text{“primary”}$   $rp$ can be only primary or secondary
                   THEN $\exists\, s \in Primary :$
                        $InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$
                        $[op \mapsto \text{“get”}, c \mapsto c, rc \mapsto rConcern, k \mapsto k, ct \mapsto Ct[c], ot \mapsto Ot[c]])]$
                 ELSE $\exists\, s \in Secondary :$
                      $InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$
                      $[op \mapsto \text{“get”}, c \mapsto c, rc \mapsto rConcern, k \mapsto k, ct \mapsto Ct[c], ot \mapsto Ot[c]])]$
        $\land\ BlockedClient' = BlockedClient \cup \{c\}$
    $\land$ UNCHANGED $\langle serverVars, clientnodeVars, BlockedThread, InMsgc, SnapshotTable \rangle$
```

Client Put

```
$ClientPutRequest \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, k \in Key, v \in Value, c \in Client \setminus BlockedClient, wConcern \in WriteConcern, wNum \in WriteNumber,$
        $\land\ OpCount[c] \neq 0$
        $\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$
                 $[op \mapsto \text{“put”}, c \mapsto c, wc \mapsto wConcern, num \mapsto wNum, k \mapsto k, v \mapsto v, ct \mapsto Ct[c]])]$
        $\land$ IF $wConcern = \text{“zero”}$   If w:0, decrease $op$ count and record history
            THEN $\land\ OpCount' = [OpCount$ EXCEPT $![c] = @ - 1]$
                 $\land\ History' = [History$ EXCEPT $![c] = Append(@, [op \mapsto \text{“put”}, ts \mapsto Ot[c], k \mapsto k, v \mapsto v]$
                  $\land\ BlockedClient' = BlockedClient$
          ELSE  $\land\ BlockedClient' = BlockedClient \cup \{c\}$   Else wait for server reply
                $\land\ OpCount' = OpCount$
                $\land\ History' \ \ = History$
    $\land$ UNCHANGED $\langle serverVars,\ BlockedThread,\ InMsgc,\ SnapshotTable \rangle$

do we need to update $Ct[c]$ here?
$ClientGetResponse\ \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, c \in Client :$
        $\land\ OpCount[c] \neq 0$                  client $c$ has operation times
        $\land\ Len(InMsgc[c]) \neq 0$              message channel is not empty
        $\land\ InMsgc[c][1].op =$ "get"            $msg$ type: get
        $\land\ Store' = [Store$ EXCEPT $![c][InMsgc[c][1].k] = InMsgc[c][1].v]$   store data
        $\land\ History' = [History$ EXCEPT $![c] = Append(@,\ [op \mapsto$ "get",
                    $ts \mapsto InMsgc[c][1].ot,\ k \mapsto InMsgc[c][1].k,\ v \mapsto InMsgc[c][1].v])]$
        $\land\ InMsgc' = [InMsgc$ EXCEPT $![c] = Tail(@)]$
        $\land\ BlockedClient' =$ IF $Len(InMsgc'[c]) = 0$
                                    THEN $BlockedClient \setminus \{c\}$
                                 ELSE $BlockedClient$   remove blocked state
        $\land\ OpCount' = [OpCount$ EXCEPT $![c] = @ - 1]$
    $\land$ UNCHANGED $\langle electionVars,\ functionalVar,\ learnableVars,\ messageVar,\ servernodeVars,\ Oplog,\ timeVar$
                      $BlockedThread,\ InMsgs,\ SnapshotTable \rangle$

$ClientPutResponse\ \triangleq$
    $\land\ ReadyToServe > 0$
    $\land\ \exists\, c \in Client :$
        $\land\ OpCount[c] \neq 0$                  client $c$ has operation times
        $\land\ Len(InMsgc[c]) \neq 0$              message channel is not empty
        $\land\ InMsgc[c][1].op =$ "put"            $msg$ type: put
        $\land\ Ct' = [Ct$ EXCEPT $![c] = HLCMax(@,\ InMsgc[c][1].ct)]$
        $\land\ Ot' = [Ot$ EXCEPT $![c] = HLCMax(@,\ InMsgc[c][1].ot)]$  Update $Ot$ to record "my write" $ot$
        $\land\ History' = [History$ EXCEPT $![c] = Append(@,\ [op$
                      $\mapsto$ "put", $ts \mapsto InMsgc[c][1].ot,\ k \mapsto InMsgc[c][1].k,\ v \mapsto InMsgc[c][1].v])]$
        $\land\ InMsgc' = [InMsgc$ EXCEPT $![c] = Tail(@)]$
        $\land\ BlockedClient' =$ IF $Len(InMsgc'[c]) = 0$
                                    THEN $BlockedClient \setminus \{c\}$
                                 ELSE $BlockedClient$   remove blocked state
        $\land\ OpCount' = [OpCount$ EXCEPT $![c] = @ - 1]$
    $\land$ UNCHANGED $\langle electionVars,\ functionalVar,\ Cp,\ CurrentTerm,\ State,\ messageVar,\ SyncSource,\ storageV$
                      $BlockedThread,\ InMsgs,\ SnapshotTable \rangle$

Action Wrapper
|------------------------------------------------------------------------|
all possible server get actions
$ServerGetReply\ \triangleq\ \lor\ ServerGetReply\_sleep$
```

$$\lor\ \mathit{ServerGetReply\_wake}$$

all possible server put actions

$$\begin{aligned}\mathit{ServerPutReply} \triangleq\ & \lor\ \mathit{ServerPutReply\_sleep}\\ & \lor\ \mathit{ServerPutReply\_wake}\end{aligned}$$

Next state for all configurations

$$\begin{aligned}\mathit{Next} \triangleq\ & \lor\ \mathit{ClientGetRequest} \lor \mathit{ClientPutRequest}\\ & \lor\ \mathit{ClientGetResponse} \lor \mathit{ClientPutResponse}\\ & \lor\ \mathit{ServerGetReply} \lor \mathit{ServerPutReply}\\ & \lor\ \mathit{Replicate}\\ & \lor\ \mathit{AdvancePt}\\ & \lor\ \mathit{ServerTakeHeartbeat}\\ & \lor\ \mathit{ServerTakeUpdatePosition}\\ & \lor\ \mathit{Snapshot}\\ & \lor\ \mathit{Stepdown}\\ & \lor\ \mathit{RollbackAndRecover}\\ & \lor\ \mathit{TurnOnReadyToServe}\\ & \lor\ \mathit{ElectPrimary}\\ & \lor\ \mathit{AdvanceCp}\end{aligned}$$

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

Causal Specifications

$$\begin{aligned}\mathit{MonotonicRead} \triangleq\ & \forall\, c \in \mathit{Client} : \forall\, i, j \in \textsc{domain}\ \mathit{History}[c] :\\ & \quad \land\ i < j\\ & \quad \land\ \mathit{History}[c][i].\mathit{op} = \text{“get”}\\ & \quad \land\ \mathit{History}[c][j].\mathit{op} = \text{“get”}\\ & \quad \Rightarrow \neg \mathit{HLCLt}(\mathit{History}[c][j].\mathit{ts},\ \mathit{History}[c][i].\mathit{ts})\end{aligned}$$

$$\begin{aligned}\mathit{MonotonicWrite} \triangleq\ & \forall\, c \in \mathit{Client} : \forall\, i, j \in \textsc{domain}\ \mathit{History}[c] :\\ & \quad \land\ i < j\\ & \quad \land\ \mathit{History}[c][i].\mathit{op} = \text{“put”}\\ & \quad \land\ \mathit{History}[c][j].\mathit{op} = \text{“put”}\\ & \quad \Rightarrow \neg \mathit{HLCLt}(\mathit{History}[c][j].\mathit{ts},\ \mathit{History}[c][i].\mathit{ts})\end{aligned}$$

$$\begin{aligned}\mathit{ReadYourWrite} \triangleq\ & \forall\, c \in \mathit{Client} : \forall\, i, j \in \textsc{domain}\ \mathit{History}[c] :\\ & \land\ i < j\\ & \land\ \mathit{History}[c][i].\mathit{op} = \text{“put”}\\ & \land\ \mathit{History}[c][j].\mathit{op} = \text{“get”}\\ & \Rightarrow \neg \mathit{HLCLt}(\mathit{History}[c][j].\mathit{ts},\ \mathit{History}[c][i].\mathit{ts})\end{aligned}$$

$$\begin{aligned}\mathit{WriteFollowRead} \triangleq\ & \forall\, c \in \mathit{Client} : \forall\, i, j \in \textsc{domain}\ \mathit{History}[c] :\\ & \land\quad i < j\\ & \land\quad \mathit{History}[c][i].\mathit{op} = \text{“get”}\\ & \land\quad \mathit{History}[c][j].\mathit{op} = \text{“put”}\end{aligned}$$

$$\Rightarrow \neg HLCLt(History[c][j].ts,\ History[c][i].ts)$$

\ $*$ Modification *History*
\ $*$ Last modified *Tue* May 17 17:12:38 *CST* 2022 by *dh*
\ $*$ Created *Thu Mar* 31 20:33:19 *CST* 2022 by *dh*