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
- *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 $c \in Client$
- *InMsgs*, — *InMsgc*[*s*]: the channel of messages at server $s \in Server$
- *OpCount*, — *OpCount*[*c*]: *op* count for client *c*
- *SnapshotTable* — *SnapshotTable*[*s*] : snapshot mapping table at server *s*

---

group related *vars* to optimize code

- $electionVars \triangleq \langle Primary, Secondary \rangle$ — *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
- $servernodeVars \triangleq \langle Ot, SyncSource \rangle$ — *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
- $clientnodeVars \triangleq \langle History, OpCount \rangle$

$serverVars \triangleq \langle electionVars,\ storageVars,\ messageVar,\ servernodeVars,\ learnableVars,\ timeVar \rangle$
$tunableVars \triangleq \langle BlockedClient,\ BlockedThread,\ History,\ InMsgc,\ InMsgs,\ OpCount,\ SnapshotTable \rangle$

ASSUME $Cardinality(Client) \geq 1$ at least one client
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
ASSUME $ReadConcern \in \{$"local", "majority", "linearizable"$\}$
ASSUME $WriteConcern \in \{$"zero", "num", "majority"$\}$
ASSUME $ReadPreference \in \{$"primary", "secondary"$\}$
ASSUME $WriteNumber \leq Cardinality(Server)$ w:*num* cannot be greater than server number

---

Helpers

$HLCLt(x,\ y) \triangleq$ 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

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

$vars \triangleq \langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,\ InMsgc,$
$InMsgs,\ ServerMsg,\ BlockedClient,\ BlockedThread,$
$OpCount,\ Pt,\ Cp,\ State,\ SnapshotTable,$
$History,\ CurrentTerm,\ SyncSource \rangle$

snapshot helpers

RECURSIVE $SelectSnapshot\_rec(\_,\ \_,\ \_)$
$SelectSnapshot\_rec(stable,\ cp,\ index) \triangleq$
IF $HLCLt(cp,\ stable[index].ot)$ THEN $stable[index - 1].store$
ELSE IF $index = Len(stable)$ THEN $stable[index].store$
ELSE $SelectSnapshot\_rec(stable,\ cp,\ index + 1)$

$SelectSnapshot(stable,\ cp) \triangleq SelectSnapshot\_rec(stable,\ cp,\ 1)$

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

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) ≜ CHOOSE x ∈ s : ∀ y ∈ s : x ≥ y

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]]
UpdateAndTick(s, msgCt) ≜
    LET newCt ≜ [Ct EXCEPT ![s] = HLCMax(Ct[s], msgCt)]  Update ct first according to msg
    IN  Ct' = IF newCt[s].p ≥ Pt[s] THEN [newCt EXCEPT ![s] = [p ↦ @.p, l ↦ @.l + 1]]
                                    ELSE [newCt 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])
```

$\land\ Oplog[i][k] = Oplog[j][k]\}$IN
IF $commonIndices = \{\}$ THEN 0 ELSE $MaxVal(commonIndices)$

The set of all *quorums*. This just calculates simple majorities, but the only important property that every quorum overlaps with every other.

$Quorum \triangleq \{i \in$ 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 $quorums$

$ReplicatedServers(states,\ ot) \triangleq$
LET $serverSet \triangleq \{subServers \in$ SUBSET $(Server) : \forall\, s \in subServers :$ LET $stateTime \triangleq [p \mapsto states[s].p,$
IN $\neg HLCLt(stateTime,\ ot)\}$
IN CHOOSE $maxSet \in serverSet : \forall\, otherSet \in serverSet : Cardinality(otherSet) \leq Cardinality(maxSet)$
IN IF $Cardinality(serverSet) = 0$ THEN $\{\}$
ELSE IF $Cardinality(serverSet) = 1$ THEN $serverSet$
ELSE CHOOSE $maxSet \in serverSet$: $\forall\, otherSet \in serverSet$: $Cardinality(otherSet) \leq Cardinality(maxSet)$

Compute a new common point according to new update position *msg*
$ComputeNewCp(s) \triangleq$
primary node: compute new mcp
IF $s \in Primary$ THEN
LET $quorumAgree \triangleq QuorumAgreeInSameTerm(State[s])$IN
IF $Cardinality(quorumAgree) > 0$
THEN LET $QuorumSet \triangleq$ CHOOSE $i \in quorumAgree$: TRUE
$serverInQuorum \triangleq$ CHOOSE $j \in QuorumSet$: TRUE
$termOfQuorum \triangleq State[s][serverInQuorum].term$ never commit log entries from previous t
$StateSet \triangleq \{[p \mapsto State[s][j].p,\ l \mapsto State[s][j].l] : j \in QuorumSet\}$
$newCommitPoint \triangleq HLCMinSet(StateSet)$
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

$InitPrimary\ \triangleq\ Primary = \{\text{CHOOSE } s \in Server : \text{TRUE}\}$
$InitSecondary\ \triangleq\ Secondary = Server \setminus Primary$
$InitOplog\ \triangleq\ Oplog = [s\ \in Server \mapsto \langle\rangle]$
$InitStore\ \triangleq\ Store\ = [n \in Server \cup Client\ \mapsto [k \in Key \mapsto Nil]]$
$InitCt\ \triangleq\ Ct = [n\ \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]$
$InitOt\ \triangleq\ Ot = [n\ \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]$
$InitInMsgc\ \triangleq\ InMsgc = [c\ \in Client \mapsto \langle\rangle]$
$InitInMsgs\ \triangleq\ InMsgs = [s\ \in Server \mapsto \langle\rangle]$
$InitServerMsg\ \triangleq\ ServerMsg = [s \in Server \mapsto \langle\rangle]$
$InitBlockedClient\ \triangleq\ BlockedClient\ = \{\}$
$InitBlockedThread\ \triangleq\ BlockedThread = [s \in Client \mapsto Nil]$
$InitOpCount\ \triangleq\ OpCount = [c \in Client \mapsto OpTimes]$
$InitPt\ \triangleq\ Pt\ = [s\ \in Server \mapsto 1]$
$InitCp\ \triangleq\ Cp = [n \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]$
$InitState\ \triangleq\ State\ = [s \in Server \mapsto [s0 \in Server \mapsto$
$[p \mapsto 0,\ l \mapsto 0,\ term\ \mapsto 0]]]$
$InitSnap\ \triangleq\ SnapshotTable = [s \in Server \mapsto \langle[ot \mapsto [p \mapsto 0,\ l \mapsto 0],$
$store \mapsto [k \in Key \mapsto Nil]]\rangle]$
$InitHistory\ \triangleq\ History = [c \in Client \mapsto \langle\rangle]$ History operation seq is empty
$InitCurrentTerm\ \triangleq\ CurrentTerm = [p \in Primary \mapsto 1]\ @@\ [s \in Server \mapsto 0]$
$InitSyncSource\ \triangleq\ SyncSource = [s \in Server \mapsto Nil]$

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

Next *State* Actions
Replication Protocol: possible actions
snapshot periodly
$Snapshot\ \triangleq$
$\land\ \exists\, s \in Server :$

$SnapshotTable' = [SnapshotTable \text{ EXCEPT } ![s] = Append(@, [ot \mapsto Ot[s], store \mapsto Store[s]])]$

create a new snapshot

$\land$ UNCHANGED $\langle serverVars, InMsgc, InMsgs, BlockedClient, BlockedThread, OpCount, History\rangle$

$Stepdown \triangleq$

$\land \exists\, s \in Primary :$

$\land\ Primary' = Primary \setminus \{s\}$

$\land\ Secondary' = Secondary \cup \{s\}$

$\land$ UNCHANGED $\langle storageVars, servernodeVars, Ct, messageVar, timeVar, Cp, State, CurrentTerm, tunable$

There are majority nodes agree to elect node i to become primary

$ElectPrimary \triangleq$

$\land \exists\, i \in Server : \exists\, majorNodes \in \text{SUBSET } (Server) :$

$\land \forall\, j \in majorNodes : \land NotBehind(i, j)$

$\land\ CurrentTerm[i] \geq CurrentTerm[j]$

$\land\ IsMajority(majorNodes)$

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

$\land\ SyncSource' = [SyncSource \text{ EXCEPT } ![i] = Nil]$

$\land$ UNCHANGED $\langle storageVars, Ct, Ot, messageVar, timeVar, Cp, State, tunableVars\rangle$

$AdvanceCp \triangleq$

$\land \exists\, s \in Primary :$

$\text{LET } newCp \triangleq ComputeNewCp(s)$

$\text{IN } Cp' = [Cp \text{ EXCEPT } ![s] = newCp]$

$\land$ UNCHANGED $\langle electionVars, storageVars, servernodeVars, Ct, messageVar, timeVar, State, CurrentTer$

heartbeatoplogOtstore

$ServerTakeHeartbeat \triangleq$

$\land \exists\, s \in Server :$

$\land\ Len(ServerMsg[s]) \neq 0$ message channel is not empty

$\land\ ServerMsg[s][1].type =$ "heartbeat"

$\land\ CurrentTerm[s] = ServerMsg[s][1].term$ only consider heartbeat *msg* in same term

$\land\ Ct' = [Ct \text{ EXCEPT } ![s] = HLCMax(Ct[s], ServerMsg[s][1].ct)]$

$\land\ State' = \text{LET } newState \triangleq GetNewState(s, ServerMsg[s][1].s, ServerMsg[s][1].aot.p, ServerMsg[s][1]$

$\text{IN } [State \text{ EXCEPT } ![s] = newState]$

$\land\ Cp' = \text{LET } newcp \triangleq ComputeNewCp(s)$

IN $[Cp$ EXCEPT $![s] = newcp]$

$\land\ ServerMsg' = [ServerMsg$ EXCEPT $![s] = Tail(@)]$

$\land\ CurrentTerm' = [CurrentTerm$ EXCEPT $![s] = Max(CurrentTerm[s], ServerMsg[s][1].term)]$ $\rightarrow$

$\land$ UNCHANGED $\langle electionVars, storageVars, servernodeVars, timeVar, CurrentTerm, tunableVars\rangle$

$ServerTakeUpdatePosition \triangleq$

$\land \exists\, s \in Server :$

$\land\ Len(ServerMsg[s]) \neq 0$ 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 $\langle electionVars, storageVars, servernodeVars, timeVar, tunableVars\rangle$

$NTPSync \triangleq$ simplify NTP protocal

$\land\ Pt' = [s \in Server \mapsto MaxPt]$

$\land$ UNCHANGED $\langle electionVars, storageVars, servernodeVars, learnableVars, messageVar, tunableVars\rangle$

$AdvancePt \triangleq$

$\land \exists\, s \in Server :$

$\land\ s \in Primary$ for simplicity

$\land\ Pt[s] \leq PtStop$

$\land\ Pt' = [Pt$ EXCEPT $![s] = @ + 1]$ advance physical time

$\land\ BroadcastHeartbeat(s)$ broadcast heartbeat periodly

$\land$ UNCHANGED $\langle electionVars, storageVars, servernodeVars, learnableVars, tunableVars\rangle$

Replication

Idea: replicate$canSyncFrom$logterm

$SyncSource[s]SyncSource\,UpdatePosition$

$UpdatePosition$actiontype$updatePosition$

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

$\land\ i \in Secondary$

$\land\ ReplicateOplog(i, j) \neq \langle\rangle$

$\land\ Oplog' = [Oplog \text{ EXCEPT } ![i] = @ \circ ReplicateOplog(i, j)]$
$\land\ Store' = [Store \text{ EXCEPT } ![i] = Store[j]]$
$\land\ Ct' = [Ct \text{ EXCEPT } ![i] = HLCMax(Ct[i], Ct[j])]$ 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 $Cp[i]$
$\land\ CurrentTerm' = [CurrentTerm \text{ EXCEPT } ![i] = Max(CurrentTerm[i], CurrentTerm[j])]$ update $Curren$
$\land\ State' =$
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 $[State \text{ EXCEPT } ![i] = hb1]$
$\land$ 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 $\langle electionVars, timeVar, tunableVars\rangle$

Rollback i's oplog and recover it to j's state
Recover to j's state immediately to prevent internal client request
$RollbackAndRecover \triangleq$
$\land\ \exists\, i, j \in Server :$
$\land\ i \in Secondary$
$\land\ CanRollback(i, j)$
$\land$ LET $cmp \triangleq RollbackCommonPoint(i, j)$ IN
LET $commonLog \triangleq SubSeq(Oplog[i], 1, cmp)$
$appendLog \triangleq SubSeq(Oplog[j], cmp + 1, Len(Oplog[j]))$
IN $Oplog' = [Oplog \text{ EXCEPT } ![i] = commonLog \circ appendLog]$
$\land\ CurrentTerm' = [CurrentTerm \text{ EXCEPT } ![i] = Max(CurrentTerm[i], CurrentTerm[j])]$ 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 $[State \text{ EXCEPT } ![i] = hb1]$
$\land$ 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 $\langle electionVars, timeVar, tunableVars\rangle$

Tunable Protocol: *Server* Actions

Server Get

$ServerGetReply\_sleep \triangleq$

$\land \exists\, s \in Server :$

$\land\ Len(InMsgs[s]) \neq 0$

$\land\ InMsgs[s][1].op =$ "get"

$\land$ IF $InMsgs[s][1].rc =$ "linearizable"

THEN $\land\ s \in Primary$

$\land\ UpdateAndTick(s,\ InMsgs[s][1].ct)$ advance cluster time

$\land\ Oplog' =$ LET $entry \triangleq [k \mapsto Nil,\ v \mapsto Nil,\ ot \mapsto Ct'[s],\ term \mapsto CurrentTerm[s]]$

$newLog \triangleq Append(Oplog[s],\ entry)$

IN $[Oplog$ EXCEPT $![s] = newLog]$

append noop operation to *oplog*$[s]$

$\land\ Ot' = [Ot$ EXCEPT $![s] = Ct'[s]]$

advance the last applied operation time $Ot[s]$

$\land\ State' =$ LET $newState \triangleq GetNewState(s,\ s,\ Ot'[s].p,\ Ot'[s].l,\ CurrentTerm[s])$

IN $[State$ EXCEPT $![s] = newState]$ update primary state

$\land\ BlockedThread' = [BlockedThread$ EXCEPT $![InMsgs[s][1].c] =$

$[type \mapsto$ "read", $rc \mapsto InMsgs[s][1].rc,\ ot \mapsto Ct'[s],\ s \mapsto s,$

$k \mapsto InMsgs[s][1].k,\ v \mapsto Store[s][InMsgs[s][1].k]]]$

add the user thread to *BlockedThread*$[c]$

ELSE $\land\ Ct' = [Ct$ EXCEPT $![s] = HLCMax(Ct[s],\ InMsgs[s][1].ct)]$ $rc =$ *local* or major

$\land\ BlockedThread' = [BlockedThread$ EXCEPT $![InMsgs[s][1].c] =$

$[type \mapsto$ "read", $rc \mapsto InMsgs[s][1].rc,\ s \mapsto s,\ k \mapsto InMsgs[s][1].k,\ ot \mapsto InMsgs[s][$

$\land\ Oplog' = Oplog$

$\land\ Ot' = Ot$

$\land\ State' = State$

$\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Tail(@)]$

$\land$ UNCHANGED $\langle electionVars,\ Cp,\ CurrentTerm,\ messageVar,\ SyncSource,\ Store,\ timeVar,$

$InMsgc,\ BlockedClient,\ clientnodeVars,\ SnapshotTable\rangle$

$ServerGetReply\_wake \triangleq$

$\land \exists\, c \in Client :$

$\land\ BlockedThread[c] \neq Nil$

$\land\ BlockedThread[c].type =$ "read"

$\land$ IF $BlockedThread[c].rc =$ "local"

THEN $\land \neg HLCLt(Ot[BlockedThread[c].s],\ BlockedThread[c].ot)$ wait until $Ot[s] \geq$ target ot

$\land\ InMsgc' = [InMsgc$ EXCEPT $![c] = Append(@,\ [op \mapsto$ "get", $k \mapsto BlockedThread[c].k,$

$v \mapsto Store[BlockedThread[c].s][BlockedThread[c].k],$

$ct \mapsto Ct[BlockedThread[c].s],\ ot \mapsto Ot[BlockedThread[c].s]])]$

ELSE $\land \neg HLCLt(Cp[BlockedThread[c].s],\ BlockedThread[c].ot)$ wait until $cp[s] \geq$ target ot

$\land$ IF $BlockedThread[c].rc =$ "linearizable"

THEN $InMsgc' = [InMsgc$ EXCEPT $![c] = Append(@,\ [op \mapsto$ "get",

$k \mapsto BlockedThread[c].k,\ v \mapsto BlockedThread[c].v,$

$ct \mapsto Ct[BlockedThread[c].s],\ ot \mapsto BlockedThread[c].ot])]$

ELSE $InMsgc' = [InMsgc$ EXCEPT $![c] = Append(@, [op \mapsto$ "get", $k \mapsto BlockedThread[c].k$

$v \mapsto SelectSnapshot(SnapshotTable[BlockedThread[c].s],\ Cp[BlockedThre$

$ct \mapsto Ct[BlockedThread[c].s],\ ot \mapsto Cp[BlockedThread[c].s]])]$

$\land\ BlockedThread' = [BlockedThread$ EXCEPT $![c] = Nil]$

$\land$ UNCHANGED $\langle serverVars,\ clientnodeVars,\ BlockedClient,\ InMsgs,\ SnapshotTable\rangle$

Server Put serveroplog

$ServerPutReply\_sleep \triangleq$

$\land \exists\, s \in Primary :$

$\land\ Len(InMsgs[s]) \neq 0$

$\land\ InMsgs[s][1].op =$ "put"

$\land\ UpdateAndTick(s,\ InMsgs[s][1].ct)$

$\land\ Ot' = [Ot$ EXCEPT $![s] = Ct'[s]]$ advance the last applied operation time $Ot[s]$

$\land\ Store' = [Store$ EXCEPT $![s][InMsgs[s][1].k] = InMsgs[s][1].v]$ append operation to $oplog[s]$

$\land\ Oplog' =$ LET $entry \triangleq [k \mapsto InMsgs[s][1].k,\ v \mapsto InMsgs[s][1].v,$

$ot \mapsto Ot'[s],\ term \mapsto CurrentTerm[s]]$

$newLog \triangleq Append(Oplog[s],\ entry)$

IN $[Oplog$ EXCEPT $![s] = newLog]$

$\land\ State' =$ LET $newState \triangleq GetNewState(s,\ s,\ Ot'[s].p,\ Ot'[s].l,\ CurrentTerm[s])$

IN $[State$ EXCEPT $![s] = newState]$ update primary state

$\land$ IF $InMsgs[s][1].wc =$ "zero" If w:0, do not sleep

THEN $BlockedThread' = BlockedThread$

ELSE $BlockedThread' = [BlockedThread$ EXCEPT $![InMsgs[s][1].c] = [type \mapsto$ "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*

$\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Tail(@)]$

$\land$ UNCHANGED $\langle electionVars,\ Cp,\ CurrentTerm,\ messageVar,\ SyncSource,\ timeVar,$

$InMsgc,\ BlockedClient,\ clientnodeVars,\ SnapshotTable\rangle$

$ServerPutReply\_wake \triangleq$

$\land \exists\, c \in Client :$

$\land\ BlockedThread[c] \neq Nil$

$\land\ BlockedThread[c].type =$ "write"

$\land$ IF $BlockedThread[c].wc =$ "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 $![c] = Append(@, [op \mapsto$ "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

$ClientGetRequest \triangleq$

$\land \exists\, k \in Key,\ c \in Client \setminus BlockedClient :$

$\land\ OpCount[c] \neq 0$

$\land$ IF $ReadConcern =$ "linearizable" In this case, read can only be sent to primary

THEN $\exists\, s \in Primary :$

$InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$

$[op \mapsto$ "get", $c \mapsto c,\ rc \mapsto ReadConcern,\ k \mapsto k,\ ct \mapsto Ct[c],\ ot \mapsto Ot[c]])]$

ELSE IF $ReadPreference =$ "primary" rp can be only primary or secondary

THEN $\exists\, s \in Primary :$

$InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$

$[op \mapsto$ "get", $c \mapsto c,\ rc \mapsto ReadConcern,\ k \mapsto k,\ ct \mapsto Ct[c],\ ot \mapsto Ot[c]])]$

ELSE $\exists\, s \in Secondary :$

$InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$

$[op \mapsto$ "get", $c \mapsto c,\ rc \mapsto ReadConcern,\ k \mapsto k,\ ct \mapsto Ct[c],\ ot \mapsto Ot[c]])]$

$\land\ BlockedClient' = BlockedClient \cup \{c\}$

$\land$ UNCHANGED $\langle serverVars,\ clientnodeVars,\ BlockedThread,\ InMsgc,\ SnapshotTable\rangle$

Client Put

$ClientPutRequest \triangleq$

$\land \exists\, k \in Key,\ v \in Value,\ c \in Client \setminus BlockedClient,\ s \in Primary :$

$\land\ OpCount[c] \neq 0$

$\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Append(@,$

$[op \mapsto$ "put", $c \mapsto c,\ wc \mapsto WriteConcern,\ num \mapsto WriteNumber,\ k \mapsto k,\ v \mapsto v,\ ct \mapsto$

$\land$ IF $WriteConcern =$ "zero" If w:0, decrease *op* count and record history

THEN $\land\ OpCount' = [OpCount$ EXCEPT $![c] = @ - 1]$

$\land\ History' = [History$ EXCEPT $![c] = Append(@, [op \mapsto$ "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 $\langle serverVars,\ BlockedThread,\ InMsgc,\ SnapshotTable\rangle$

$ClientGetResponse \triangleq$

$\land \exists\, c \in Client :$

$\land\ OpCount[c] \neq 0$ 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

$\land\ Ct' = [Ct$ EXCEPT $![c] = HLCMax(@,\ InMsgc[c][1].ct)]$

$\land\ Ot' = [Ot$ EXCEPT $![c] = HLCMax(@,\ InMsgc[c][1].ot)]$

$\land\ Store' = [Store$ 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 \ {c}
                ELSE BlockedClient    remove blocked state
        ∧ OpCount' = [OpCount EXCEPT ![c] = @ − 1]
    ∧ UNCHANGED ⟨electionVars, State, Cp, CurrentTerm, messageVar, SyncSource, Oplog, timeVar,
                 BlockedThread, InMsgs, SnapshotTable⟩

ClientPutResponse ≜
    ∧ ∃ c ∈ Client :
        ∧ OpCount[c] ≠ 0                client c has operation times
        ∧ Len(InMsgc[c]) ≠ 0            message channel is not empty
        ∧ InMsgc[c][1].op = "put"       msg type: put
        ∧ Ct' = [Ct EXCEPT ![c] = HLCMax(@, InMsgc[c][1].ct)]
        ∧ Ot' = [Ot EXCEPT ![c] = HLCMax(@, InMsgc[c][1].ot)]  Update Ot to record "my write" ot
        ∧ History' = [History EXCEPT ![c] = Append(@, [op
                        ↦ "put", ts ↦ InMsgc[c][1].ot, k ↦ InMsgc[c][1].k, v ↦ InMsgc[c][1].v])]
        ∧ InMsgc' = [InMsgc EXCEPT ![c] = Tail(@)]
        ∧ BlockedClient' = IF Len(InMsgc'[c]) = 0
                THEN BlockedClient \ {c}
                ELSE BlockedClient    remove blocked state
        ∧ OpCount' = [OpCount EXCEPT ![c] = @ − 1]
    ∧ UNCHANGED ⟨electionVars, Cp, CurrentTerm, State, messageVar, SyncSource, storageVars, timeVar,
                 BlockedThread, InMsgs, SnapshotTable⟩

──────────────────────────────────────────────────────────────────────

Action Wrapper
all possible server get actions
ServerGetReply ≜ ∨ ServerGetReply_sleep
                 ∨ ServerGetReply_wake

all possible server put actions
ServerPutReply ≜ ∨ ServerPutReply_sleep
                 ∨ ServerPutReply_wake

──────────────────────────────────────────────────────────────────────

Next state for all configurations
Next ≜ ∨ ClientGetRequest ∨ ClientPutRequest
       ∨ ClientGetResponse ∨ ClientPutResponse
       ∨ ServerGetReply ∨ ServerPutReply
       ∨ Replicate
       ∨ AdvancePt
       ∨ ServerTakeHeartbeat
       ∨ ServerTakeUpdatePosition
       ∨ Snapshot
       ∨ Stepdown
       ∨ RollbackAndRecover
       ∨ ElectPrimary
       ∨ AdvanceCp
```

$\lor\ NTPSync$

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

Causal Specifications

$$MonotonicRead \triangleq \forall\, c \in Client : \forall\, i, j \in \text{DOMAIN } History[c] :$$
$$\land\ i < j$$
$$\land\ History[c][i].op = \text{“get”}$$
$$\land\ History[c][j].op = \text{“get”}$$
$$\Rightarrow \neg HLCLt(History[c][j].ts,\ History[c][i].ts)$$

$$MonotonicWrite \triangleq \forall\, c \in Client : \forall\, i, j \in \text{DOMAIN } History[c] :$$
$$\land\ i < j$$
$$\land\ History[c][i].op = \text{“put”}$$
$$\land\ History[c][j].op = \text{“put”}$$
$$\Rightarrow \neg HLCLt(History[c][j].ts,\ History[c][i].ts)$$

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

$$WriteFollowRead \triangleq \forall\, c \in Client : \forall\, i, j \in \text{DOMAIN } History[c] :$$
$$\land\ \ i < j$$
$$\land\ \ History[c][i].op = \text{“get”}$$
$$\land\ \ History[c][j].op = \text{“put”}$$
$$\Rightarrow\ \neg HLCLt(History[c][j].ts,\ History[c][i].ts)$$

\ * Modification *History*
\ * Last modified *Thu Jun* 16 17:05:05 *CST* 2022 by *dh*
\ * Created *Thu Mar* 31 20:33:19 *CST* 2022 by *dh*