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
Number *writeConcern* number

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*
InMsgc, *InMsgc*[*c*]: the channel of messages at client $c \in Client$
InMsgs, *InMsgc*[*s*]: the channel of messages at server $s \in Server$
ServerMsg, *ServerMsg*[*s*]: the channel of heartbeat msgs at server *s*
BlockedClient, *BlockedClient*: *Client* operations in progress
BlockedThread, *BlockedThread*: blocked user thread and content
OpCount, *OpCount*[*c*]: *op* count for client *c*
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
CalState, *CalState*: sorted *State*[*Primary*]
SnapshotTable, *SnapshotTable*[*s*] : snapshot mapping table at server *s*
History, *History*[*c*]: *History* sequence at client *c*
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*

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

Helpers

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

vars ≜ ⟨Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc,
        InMsgs, ServerMsg, BlockedClient, BlockedThread,
        OpCount, Pt, Cp, CalState, State, SnapshotTable,
        History, CurrentTerm, ReadyToServe, SyncSource⟩

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

 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) ≜ LogTerm(i, Len(Oplog[i]))

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

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

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

 commit point
RECURSIVE AddState(_, _, _)
AddState(new, state, index) ≜
    IF index = 1 ∧ HLCLt(new, state[1])
        THEN ⟨new⟩ ∘ state  less than the first
     ELSE IF index = Len(state) + 1
```

```
        THEN state ∘ ⟨new⟩
    ELSE IF HLCLt(new, state[index])
        THEN SubSeq(state, 1, index − 1) ∘ ⟨new⟩ ∘ SubSeq(state, index, Len(state))
    ELSE AddState(new, state, index + 1)

RECURSIVE RemoveState(_, _, _)
RemoveState(old, state, index) ≜
    IF state[index] = old
        THEN SubSeq(state, 1, index − 1) ∘ SubSeq(state, index + 1, Len(state))
    ELSE RemoveState(old, state, index + 1)

AdvanceState(new, old, state) ≜ AddState(new, RemoveState(old, state, 1), 1)
```

clock

```
UnchangedExPt ≜ ⟨Primary, Secondary, InMsgc, InMsgs, ServerMsg, Oplog, Store,
                 Ct, Ot, BlockedClient, OpCount⟩
UnchangedExCt ≜ ⟨Primary, Secondary, InMsgc, InMsgs, ServerMsg, Oplog, Store,
                 Pt, Ot, BlockedClient, OpCount⟩
MaxPt ≜ LET x ≜ CHOOSE s ∈ Server : ∀ s1 ∈ Server \ {s} :
                    Pt[s] ≥ Pt[s1]IN   Pt[x]

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

heartbeat

Only *Primary* node sends heartbeat once advance pt

```
BroadcastHeartbeat(s) ≜
    LET msg ≜ [type ↦ "heartbeat", s ↦ s, aot ↦ Ot[s],
               ct ↦ Ct[s], cp     ↦ Cp[s], term ↦ CurrentTerm[s]]
    IN  ServerMsg′ = [x ∈ Server ↦ IF x = s THEN ServerMsg[x]
                                    ELSE Append(ServerMsg[x], msg)]
```

Can node i sync from node j?

```
CanSyncFrom(i, j) ≜
    ∧ Len(Oplog[i]) < Len(Oplog[j])
    ∧ LastTerm(i) = LogTerm(j, Len(Oplog[i]))
```

Oplog entries needed to replicate from j to i

```
ReplicateOplog(i, j) ≜
    LET len_i ≜ Len(Oplog[i])
        len_j ≜ Len(Oplog[j])
    IN  IF i ≠ Primary ∧ 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
```

$\land\ Len(Oplog[j]) > 0$
$\land\ LastTerm(i) < LastTerm(j)$
$\land$
$\quad \lor\ Len(Oplog[i]) > Len(Oplog[j])$
$\quad \lor\ \land\ Len(Oplog[i]) \leq Len(Oplog[j])$
$\quad\quad \land\ LastTerm(i) \neq LogTerm(j,\ Len(Oplog[i]))$
$Len(Oplog[i])Len(Oplog[i] + 1? - 1?)$

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

$RollbackCommonPoint(i,\ j) \triangleq$
$\quad$ LET $commonIndices \triangleq \{k \in$ DOMAIN $Oplog[i] :$
$\quad\quad\quad\quad \land\ k \leq Len(Oplog[j])$
$\quad\quad\quad\quad \land\ Oplog[i][k] = Oplog[j][k]\}$IN
$\quad\quad$ IF $commonIndices = \{\}$ THEN 0 ELSE $MaxVal(commonIndices)$

Init Part

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

$Init \triangleq$
$\quad \land\ InitPrimary \land InitSecondary \land InitOplog \land InitStore \land InitCt$
$\quad \land\ InitOt \land InitPt \land InitCp \land InitCalState \land InitInMsgc \land InitInMsgs$

$\land$ *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 $\triangleq$
$\land$ *ReadyToServe* > 0
$\land \exists\, s \in Server :$
SnapshotTable$'$ = [*SnapshotTable* EXCEPT ![s] =
Append(@, [*ot* $\mapsto$ *Ot*[s], *store* $\mapsto$ *Store*[s]])]
create a new snapshot
$\land$ UNCHANGED $\langle$*Primary*, *Secondary*, *Oplog*, *Store*, *Ct*, *Ot*, *InMsgc*,
InMsgs, *ServerMsg*, *BlockedClient*, *BlockedThread*,
OpCount, *Pt*, *Cp*, *CalState*, *State*, *History*, *CurrentTerm*,
ReadyToServe, *SyncSource*$\rangle$

Stepdown $\triangleq$
$\land$ *ReadyToServe* > 0
$\land \exists\, s \in Primary :$
$\land$ *Primary*$'$ = *Primary* $\setminus \{s\}$
$\land$ *Secondary*$'$ = *Secondary* $\cup \{s\}$
$\land$ UNCHANGED $\langle$*Oplog*, *Store*, *Ct*, *Ot*, *InMsgc*, *InMsgs*, *ServerMsg*,
BlockedClient, *BlockedThread*, *OpCount*, *Pt*, *Cp*,
State, *CalState*, *SnapshotTable*, *History*, *CurrentTerm*,
ReadyToServe, *SyncSource*$\rangle$

There are majority nodes agree to elect node i to become primary
ElectPrimary(i, *majorNodes*) $\triangleq$
$\land$ *ReadyToServe* > 0
$\land \forall\, j \in majorNodes :$ $\land$ *NotBehind*(i, j)
$\land$ *CurrentTerm*[i] $\geq$ *CurrentTerm*[j]
$\land$ *IsMajority*(*majorNodes*)
voted nodes for i cannot be primary anymore
$\land$ *Primary*$'$ = LET *possiblePrimary* $\triangleq$ *Primary* $\setminus$ *majorNodes*
IN *possiblePrimary* $\cup \{i\}$
add voted nodes into secondaries
$\land$ *Secondary*$'$ = LET *possibleSecondary* $\triangleq$ *Secondary* $\cup$ *majorNodes*
IN *possibleSecondary* $\setminus \{i\}$
$\land$ *CurrentTerm*$'$ = [*index* $\in$ *Server* $\mapsto$ IF *index* $\in$ (*majorNodes* $\cup \{i\}$)
THEN *CurrentTerm*[i] + 1
ELSE *CurrentTerm*[*index*]]

$\land$ UNCHANGED $\langle Oplog, Store, Ct, Ot, InMsgc, InMsgs, ServerMsg, BlockedClient, BlockedThread, OpCount, Pt, Cp, State, CalState, SnapshotTable, History, CurrentTerm, ReadyToServe, SyncSource\rangle$

$TurnOnReadyToServe \triangleq$
$\land\ ReadyToServe = 0$
$\land \exists\, s \in Primary :$
$\land\ CurrentTerm' = [CurrentTerm$ EXCEPT $![s] = CurrentTerm[s] + 1]$
$\land\ ReadyToServe' = ReadyToServe + 1$
$\land$ UNCHANGED $\langle Primary,\ Secondary, Oplog, Store, Ct, Ot, InMsgc, InMsgs, ServerMsg, BlockedClient, BlockedThread, OpCount, Pt, Cp, State, CalState, SnapshotTable, History, SyncSource\rangle$

$AdvanceCp \triangleq$
$\land\ ReadyToServe > 0$
$\land\ Cp' = [Cp$ EXCEPT $![Primary] = CalState[Cardinality(Server) \div 2 + 1]]$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, InMsgs, ServerMsg, BlockedClient, BlockedThread, OpCount, Pt, CalState, State, SnapshotTable, History, CurrentTerm, ReadyToServe, SyncSource\rangle$

$ServerTakeHeartbeat \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, s \in Server :$
$\land\ Len(ServerMsg[s]) \neq 0$ message channel is not empty
$\land\ ServerMsg[s].type$ = "heartbeat"
$\land\ Ct' = [Ct$ EXCEPT $![s] = HLCMax(Ct[s], ServerMsg[s][1].ct)]$
$\land\ State' =$
LET $SubHbState \triangleq State[s]$
$hb \triangleq [SubHbState$ EXCEPT $![ServerMsg[s][1].s] = ServerMsg[s][1].aot]$
IN $[State$ EXCEPT $![s] = hb]$
$\land\ CalState' =$ LET $newcal \triangleq$
IF $s \in Primary$ primary node: update *CalState*
THEN $AdvanceState(ServerMsg[s][1].aot, State[s][ServerMsg[s][1].s], CalState)$
ELSE $CalState$ IN $newcal$
$\land\ Cp' =$ LET $newcp \triangleq$
primary node: compute new mcp
IF $s \in Primary$ THEN $CalState'[Cardinality(Server) \div 2 + 1]$
secondary node: update mcp
ELSE IF $\neg HLCLt(ServerMsg[s][1].cp, Cp[s])$
$\land\ \neg HLCLt(Ot[s], ServerMsg[s][1].cp)$
THEN $ServerMsg[s][1].cp$
ELSE $Cp[s]$
IN $[Cp$ EXCEPT $![s] = newcp]$

$\land\ ServerMsg' = [ServerMsg \text{ EXCEPT } ![s] = Tail(@)]$
$\land\ CurrentTerm' = [CurrentTerm \text{ EXCEPT } ![s] = Max(CurrentTerm[s], ServerMsg[s][1].term)]$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ot, InMsgc,$
$InMsgs, BlockedClient, BlockedThread, OpCount, Pt,$
$SnapshotTable, History, ReadyToServe, SyncSource\rangle$

$ServerTakeUpdatePosition \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, s \in Server :$
$\land\ Len(ServerMsg[s]) \neq 0$ message channel is not empty
$\land\ ServerMsg[s].type = \text{“update\_position”}$
$\land\ Ct' = [Ct \text{ EXCEPT } ![s] = HLCMax(Ct[s], ServerMsg[s][1].ct)]$ update ct accordingly
$\land\ State' =$
LET $SubHbState \triangleq State[s]$
$hb \triangleq [SubHbState \text{ EXCEPT } ![ServerMsg[s][1].s] =$
$ServerMsg[s][1].aot]$
IN $[State \text{ EXCEPT } ![s] = hb]$
$\land\ CalState' =$ LET $newcal \triangleq$
IF $s \in Primary$ primary node: update *CalState*
THEN $AdvanceState(ServerMsg[s][1].aot,$
$State[s][ServerMsg[s][1].s], CalState)$
ELSE $CalState$ IN $newcal$
$\land\ Cp' =$ LET $newcp \triangleq$
primary node: compute new mcp
IF $s \in Primary$ THEN $CalState'[Cardinality(Server) \div 2 + 1]$
secondary node: update mcp
ELSE IF $\neg HLCLt(ServerMsg[s][1].cp, Cp[s])$
$\land\ \neg HLCLt(Ot[s], ServerMsg[s][1].cp)$
THEN $ServerMsg[s][1].cp$
ELSE $Cp[s]$
IN $[Cp \text{ EXCEPT } ![s] = newcp]$
$\land\ ServerMsg' =$ LET $newServerMsg \triangleq [ServerMsg \text{ EXCEPT } ![s] = Tail(@)]$
$appendMsg \triangleq [type \mapsto \text{“update\_position”}, s \mapsto s, aot \mapsto ServerMsg[s][1].aot,$
$ct \mapsto ServerMsg[s][1].ct, cp \mapsto ServerMsg[s][1].cp, term \mapsto Server$
$newMsg \triangleq$ IF $s \in Primary$
THEN $newServerMsg$ If s is primary, accept the msg, else forward it to its sy
ELSE $[newServerMsg \text{ EXCEPT } ![SyncSource[s]] = Append(newServer$
IN $newMsg$
$\land\ CurrentTerm' = [CurrentTerm \text{ EXCEPT } ![s] = Max(CurrentTerm[s], ServerMsg[s][1].term)]$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ot, InMsgc, InMsgs,$
$BlockedClient, BlockedThread, OpCount, Pt, SnapshotTable,$
$History, ReadyToServe, SyncSource\rangle$

NTPPrimary
$NTPSync \triangleq$ simplify *NTP* protocal

$\land$ $ReadyToServe > 0$
$\land$ $Pt' = [s \in Server \mapsto MaxPt]$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, InMsgs, ServerMsg, BlockedClient, BlockedThread, OpCount, Cp, CalState, State, SnapshotTable, History, CurrentTerm, ReadyToServe, SyncSource\rangle$

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

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
$Replicate(i, j) \triangleq$
$\land$ $ReadyToServe > 0$
$\land$ $CanSyncFrom(i, j)$
$\land$ $i \in Secondary$
$\land$ $ReplicateOplog(i, j) \neq \langle\rangle$
$\land$ $Oplog' = [Oplog$ EXCEPT $![i] = @ \circ ReplicateOplog(i, j)]$
$\land$ $Store' = [Store$ EXCEPT $![i] = Store[j]]$
$\land$ $Ct' = [Ct$ EXCEPT $![i] = HLCMax(Ct[i], Ct[j])]$ update $Ct[i]$
$\land$ $Ot' = [Ot$ EXCEPT $![i] = HLCMax(Ot[i], Ot[j])]$ update $Ot[i]$
$\land$ $Cp' = [Cp$ EXCEPT $![i] = HLCMax(Cp[i], Cp[j])]$ update $Cp[i]$
$\land$ $CurrentTerm' = [CurrentTerm$ EXCEPT $![i] = Max(CurrentTerm[i], CurrentTerm[j])]$ update $CurrentTer$
$\land$ $State' =$
LET $SubHbState \triangleq State[i]$
$hb \triangleq [SubHbState$ EXCEPT $![j] = Ot[j]]$
IN $[State$ EXCEPT $![i] = hb]$ update j's state i knows
$\land$ LET $msg \triangleq [type \mapsto$ "update_position", $s \mapsto i$, $aot \mapsto Ot'[i]$, $ct \mapsto Ct'[i]$, $cp \mapsto Cp'[i]]$
IN $ServerMsg' = [ServerMsg$ EXCEPT $![j] = Append(ServerMsg[j], msg)]$
$\land$ $SyncSource' = [SyncSource$ EXCEPT $![i] = j]$
$\land$ UNCHANGED $\langle Primary, Secondary, InMsgc, InMsgs, BlockedClient, BlockedThread, OpCount, Pt, CalState, SnapshotTable, History, ReadyToServe\rangle$

$RollbackOplog(i, j) \triangleq$

$\land\ i \in Secondary$
$\land\ CanRollback(i,\ j)$
$\land$ LET $cmp \triangleq RollbackCommonPoint(i,\ j)$IN
$\quad\land\ Oplog' = [Oplog$ EXCEPT $![i] = SubSeq(Oplog[i],\ 1,\ cmp)]$
$\quad\land\ CurrentTerm' = [CurrentTerm$ EXCEPT $![i] = LogTerm(j,\ cmp)]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Store,\ Ct,\ Ot,\ InMsgc,\ InMsgs,$
$ServerMsg,\ BlockedClient,\ BlockedThread,\ OpCount,$
$Pt,\ Cp,\ State,\ CalState,\ SnapshotTable,\ History,$
$ReadyToServe,\ SyncSource\rangle$

Tunable Protocol: *Server* Actions

Server Get

$ServerGetReply\_local\_sleep \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, s \in Server :$
$\quad\land\ Len(InMsgs[s]) \neq 0$ message channel is not empty
$\quad\land\ InMsgs[s][1].op =$ "get" *msg* type: get
$\quad\land\ InMsgs[s][1].rc =$ "local" Read Concern: local
$\quad\land\ Ct' = [Ct$ EXCEPT $![s] = HLCMax(Ct[s],\ InMsgs[s][1].ct)]$ Update *Ct* according to *InMsg*
$\quad\land\ BlockedThread' = [BlockedThread$ EXCEPT $![InMsgs[s][1].c] =$
$[type \mapsto$ "read_local", $s \mapsto s,\ k \mapsto InMsgs[s][1].k,\ ot \mapsto InMsgs[s][1].ot]]$
$\quad\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Tail(@)]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Oplog,\ Store,\ Ot,\ InMsgc,\ ServerMsg,$
$BlockedClient,\ OpCount,\ Pt,\ Cp,$
$CalState,\ State,\ SnapshotTable,\ History,$
$CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

$ServerGetReply\_local\_wake \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, c \in Client :$
$\quad\land\ BlockedThread[c] \neq Nil$
$\quad\land\ BlockedThread[c].type =$ "read_local"
$\quad\land\ \neg HLCLt(Ot[BlockedThread[c].s],\ BlockedThread[c].ot)$ wait until $Ot[s] \geq$ target *ot*
$\quad\land\ InMsgc' = [InMsgc$ EXCEPT $![c] = Append(@,\ [op \mapsto$ "get", $k \mapsto BlockedThread[c].k,\ v \mapsto$
$Store[BlockedThread[c].s][BlockedThread[c].k],$
$ct \mapsto Ct[BlockedThread[c].s],\ ot \mapsto Ot[BlockedThread[c].s]])]$
send *msg* to client
$\quad\land\ BlockedThread' = [BlockedThread$ EXCEPT $![c] = Nil]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,\ InMsgs,\ ServerMsg,$
$BlockedClient,\ OpCount,\ Pt,\ Cp,$
$CalState,\ State,\ SnapshotTable,\ History,$
$CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

$ServerGetReply\_majority\_sleep \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, s \in Server :$
$\land\ Len(InMsgs[s]) \neq 0$ message channel is not empty
$\land\ InMsgs[s][1].op =$ "get" *msg* type: get
$\land\ InMsgs[s][1].rc =$ "majority" Read Concern: majority
$\land\ Ct' = [Ct$ EXCEPT $![s] = HLCMax(Ct[s], InMsgs[s][1].ct)]$
$\land\ BlockedThread' = [BlockedThread$ EXCEPT $![InMsgs[s][1].c] =$
$[type \mapsto$ "read_major", $s \mapsto s,\ k \mapsto InMsgs[s][1].k,\ ot \mapsto InMsgs[s][1].ot]$
$\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Tail(@)]$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ot, InMsgc, ServerMsg,$
$BlockedClient, OpCount, Pt, Cp,$
$CalState, State, SnapshotTable, History,$
$CurrentTerm, ReadyToServe, SyncSource\rangle$

$ServerGetReply\_majority\_wake \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, c \in Client :$
$\land\ BlockedThread[c] \neq Nil$
$\land\ BlockedThread[c].type =$ "read_major"
$\land\ \neg HLCLt(Ot[BlockedThread[c].s], BlockedThread[c].ot)$ wait until $Ot[s] \geq$ target *ot*
$\land\ InMsgc' = [InMsgc$ EXCEPT $![c] = Append(@, [op \mapsto$ "get", $k \mapsto BlockedThread[c].k,\ v \mapsto$
$SelectSnapshot(SnapshotTable[BlockedThread[c].s],\ Cp[BlockedThread[c].s])[BlockedThr$
$ct \mapsto Ct[BlockedThread[c].s],\ ot \mapsto Cp[BlockedThread[c].s]])]$
send *msg* to client
$\land\ BlockedThread' = [BlockedThread$ EXCEPT $![c] = Nil]$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgs, ServerMsg,$
$BlockedClient, OpCount, Pt, Cp,$
$CalState, State, SnapshotTable, History,$
$CurrentTerm, ReadyToServe, SyncSource\rangle$

$ServerGetReply\_linearizable\_sleep \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, s \in Server :$
$\land\ s = Primary$
$\land\ Len(InMsgs[s]) \neq 0$
$\land\ InMsgs[s][1].op =$ "get"
$\land\ InMsgs[s][1].rc =$ "linea" Read Concern: linearizable
$\land\ Tick(s)$ advance cluster time
$\land\ Oplog' = [Oplog$ EXCEPT $![Primary] =$
$Append(@, \langle Nil,\ Nil,\ Ct'[s]\rangle)]$
append noop operation to *oplog*$[s]$
$\land\ Ot' = [Ot$ EXCEPT $![s] = \ Ct'[s]]$
advance the last applied operation time $Ot[s]$
$\land\ State' =$

```
        LET SubHbState ≜ State[s]
            hb ≜ [SubHbState EXCEPT ![Primary] = Ot'[Primary]]
        IN  [State EXCEPT ![s] = hb]   update primary state
    ∧ CalState' = AdvanceState(Ot'[Primary], Ot[Primary], CalState)
    ∧ InMsgs' = [InMsgs EXCEPT ![s] = Tail(@)]
    ∧ BlockedThread' = [BlockedThread EXCEPT ![InMsgs[s][1].c] =
                          [type ↦ "read_linea", ot ↦ Ct'[s], s ↦ s, k
                           ↦ InMsgs[s][1].k, v ↦ Store[s][InMsgs[s][1].k]]]
                  add the user thread to BlockedThread[c]
∧ UNCHANGED ⟨Primary, Secondary, Store, InMsgc, ServerMsg, BlockedClient,
             OpCount, Pt, Cp, SnapshotTable, History,
             CurrentTerm, ReadyToServe, SyncSource⟩

ServerGetReply_linearizable_wake ≜
    ∧ ReadyToServe > 0
    ∧ ∃ c ∈ Client :
      ∧ BlockedThread[c] ≠ Nil
      ∧ BlockedThread[c].type = "read_linea"
      ∧ ¬HLCLt(Cp[BlockedThread[c].s], BlockedThread[c].ot)   cp[s] ≥ target ot
      ∧ InMsgc' = [InMsgc EXCEPT ![c] = Append(@, [op ↦ "get", k
                    ↦ BlockedThread[c].k, v ↦ BlockedThread[c].v, ct
                    ↦ Ct[BlockedThread[c].s], ot ↦ BlockedThread[c].ot])]
      ∧ BlockedThread' = [BlockedThread EXCEPT ![c] = Nil]   remove blocked state
    ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ct, Ot, InMsgs,
                 ServerMsg, BlockedClient, OpCount, Pt, Cp,
                 CalState, State, SnapshotTable, History,
                 CurrentTerm, ReadyToServe, SyncSource⟩

 Server Put
 serveroplog

ServerPutReply_zero ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Primary :
      ∧ Len(InMsgs[s]) ≠ 0          message channel is not empty
      ∧ InMsgs[s][1].op = "put"     msg type: put
      ∧ InMsgs[s][1].wc = "zero"    Write Concern: 0
      ∧ Tick(s)                     advance cluster time
      ∧ Ot' = [Ot EXCEPT ![s] = Ct'[s]]
                  advance the last applied operation time Ot[Primary]
      ∧ Store' = [Store EXCEPT ![s][InMsgs[s][1].k] = InMsgs[s][1].v]
                  append operation to oplog[primary]
      ∧ Oplog' = LET entry ≜ [k ↦ InMsgs[s][1].k, v ↦ InMsgs[s][1].v,
                              ot ↦ Ot'[s], term ↦ CurrentTerm[s]]
                     newLog ≜ Append(Oplog[s], entry)
                 IN  [Oplog EXCEPT ![s] = newLog]
```

$\land\ State' =$
LET $SubHbState \triangleq State[s]$
$hb \triangleq [SubHbState$ EXCEPT $![s] = Ot'[s]]$
IN $[State$ EXCEPT $![s] = hb]$ update primary state
$\land\ CalState' = AdvanceState(Ot'[s],\ Ot[s],\ CalState)$
$\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Tail(@)]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ InMsgc,\ ServerMsg,\ BlockedClient,$
$BlockedThread,\ OpCount,\ Pt,\ Cp,\ SnapshotTable,$
$History,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

****
DH modified: Add k and v message when block thread, and return them when wake
****

$ServerPutReply\_number\_sleep \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, s \in Primary :$
$\land\ Len(InMsgs[s]) \neq 0$ message channel is not empty
$\land\ InMsgs[s][1].op =$ "put" *msg* type: put
$\land\ InMsgs[s][1].wc =$ "num" Write Concern: *num*
$\land\ Tick(s)$ advance cluster time
$\land\ Ot' = [Ot$ EXCEPT $![s] = Ct'[s]]$
advance the last applied operation time $Ot[Primary]$
$\land\ Store' = [Store$ EXCEPT $![s][InMsgs[s][1].k] = InMsgs[s][1].v]$
$\land$ 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 $Oplog' = [Oplog$ EXCEPT $![s] = newLog]$
$\land\ State' =$
LET $SubHbState \triangleq State[s]$
$hb \triangleq [SubHbState$ EXCEPT $![s] = Ot'[s]]$
IN $[State$ EXCEPT $![s] = hb]$ update primary state
$\land\ CalState' = AdvanceState(Ot'[s],\ Ot[s],\ CalState)$
$\land\ BlockedThread' = [BlockedThread$ EXCEPT $![InMsgs[s][1].c] = [type \mapsto$ "write_num",
$ot \mapsto Ot'[s],\ s \mapsto s,\ numnode \mapsto InMsgs[s][1].num,$
$k \mapsto InMsgs[s][1].k,\ v \mapsto InMsgs[s][1].v]]$
add the user *thHistory* to $BlockedThread[c]$
$\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Tail(@)]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ InMsgc,\ ServerMsg,\ BlockedClient,$
$OpCount,\ Pt,\ Cp,\ SnapshotTable,$
$History,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

$ServerPutReply\_number\_wake \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, c \in Client :$
$\land\ BlockedThread[c] \neq Nil$

$\land\ BlockedThread[c].type = \text{“write\_num”}$
$\land\ \neg HLCLt(CalState[Cardinality(Server) - BlockedThread[c].numnode + 1],$
$\quad BlockedThread[c].ot)$ $CalState[s][n - num + 1] \geq$ target ot
$\land\ InMsgc' = [InMsgc$ EXCEPT $![c] =\ Append(@, [op \mapsto \text{“put”}, ct$
$\quad \mapsto Ct[Primary], ot \mapsto BlockedThread[c].ot, k \mapsto BlockedThread[c].k, v \mapsto BlockedThrea$
$\land\ BlockedThread' =\ [BlockedThread$ EXCEPT $![c] = Nil]$
remove blocked state
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgs,$
$ServerMsg, BlockedClient, OpCount, Pt, Cp,$
$CalState, State, SnapshotTable, History,$
$CurrentTerm, ReadyToServe, SyncSource\rangle$

****
DH modified: Add k and v message when block thread, and return them when wake
****

$ServerPutReply\_majority\_sleep \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, s \in Primary :$
$\land\ Len(InMsgs[s]) \neq 0$
$\land\ InMsgs[s][1].op = \text{“put”}$
$\land\ InMsgs[s][1].wc = \text{“major”}$
$\land\ Tick(s)$
$\land\ Ot' = [Ot$ EXCEPT $![s] =\ Ct'[s]]$
$\land\ Store' = [Store$ EXCEPT $![s][InMsgs[s][1].k] = InMsgs[s][1].v]$
$\land\ Oplog' =$
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 $SubHbState \triangleq State[s]$
$hb \triangleq [SubHbState$ EXCEPT $![s] = Ot'[s]]$
IN $[State$ EXCEPT $![s] = hb]$ update primary state
$\land\ CalState' = AdvanceState(Ot'[s], Ot[s], CalState)$
$\land\ BlockedThread' = [BlockedThread$ EXCEPT $![InMsgs[s][1].c] = [type \mapsto \text{“write\_major”}, ot$
$\quad \mapsto Ot'[s], s \mapsto s, k \mapsto InMsgs[s][1].k, v \mapsto InMsgs[s][1].v]$
$\land\ InMsgs' = [InMsgs$ EXCEPT $![s] = Tail(@)]$
$\land$ UNCHANGED $\langle Primary, Secondary, InMsgc, ServerMsg, BlockedClient, OpCount,$
$Pt, Cp, SnapshotTable, History, CurrentTerm, ReadyToServe, SyncSource\rangle$

$ServerPutReply\_majority\_wake \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, c \in Client :$
$\land\ BlockedThread[c] \neq\ Nil$
$\land\ BlockedThread[c].type = \text{“write\_major”}$
$\land\ \neg HLCLt(Cp[Primary], BlockedThread[c].ot)$

$\land\ InMsgc' = [InMsgc \text{ EXCEPT } ![c] =$
$\quad Append(@, [op \mapsto \text{“put”}, ct \mapsto Ct[BlockedThread[c].s], ot \mapsto BlockedThread[c].ot,$
$\qquad k \mapsto BlockedThread[c].k, v \mapsto BlockedThread[c].v])]$
$\land\ BlockedThread' = [BlockedThread \text{ EXCEPT } ![c] = Nil]$
$\land \text{ UNCHANGED } \langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgs, ServerMsg,$
$\qquad BlockedClient, OpCount, Pt, Cp, CalState, State, SnapshotTable,$
$\qquad History, CurrentTerm, ReadyToServe, SyncSource\rangle$

Tunable Protocol: *Client* Actions

Client Get

$ClientGetRequest\_local\_primary \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, k \in Key, c \in Client \setminus BlockedClient, s \in Primary :$
$\quad \land\ InMsgs' = [InMsgs \text{ EXCEPT } ![s] = Append(@,$
$\qquad [op \mapsto \text{“get”}, c \mapsto c, rc \mapsto \text{“local”}, k \mapsto k, ct \mapsto Ct[c], ot \mapsto Ot[c]])]$
$\quad \land\ BlockedClient' = BlockedClient \cup \{c\}$
$\land \text{ UNCHANGED } \langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, ServerMsg,$
$\qquad BlockedThread, OpCount, Pt, Cp, CalState,$
$\qquad State, SnapshotTable, History,$
$\qquad CurrentTerm, ReadyToServe, SyncSource\rangle$

$ClientGetRequest\_local\_secondary \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, k \in Key, c \in Client \setminus BlockedClient, s \in Secondary :$
$\quad \land\ InMsgs' = [InMsgs \text{ EXCEPT } ![s] = Append(@,$
$\qquad [op \mapsto \text{“get”}, c \mapsto c, rc \mapsto \text{“local”}, k \mapsto k, ct \mapsto Ct[c], ot \mapsto Ot[c]])]$
$\quad \land\ BlockedClient' = BlockedClient \cup \{c\}$
$\land \text{ UNCHANGED } \langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, ServerMsg,$
$\qquad BlockedThread, OpCount, Pt, Cp, CalState, State, SnapshotTable,$
$\qquad History, CurrentTerm, ReadyToServe, SyncSource\rangle$

$ClientGetRequest\_majority\_primary \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, k \in Key, c \in Client \setminus BlockedClient, s \in Primary :$
$\quad \land\ InMsgs' = [InMsgs \text{ EXCEPT } ![s] = Append(@,$
$\qquad [op \mapsto \text{“get”}, c \mapsto c, rc \mapsto \text{“major”}, k \mapsto k, ct \mapsto Ct[c], ot \mapsto Ot[c]])]$
$\quad \land\ BlockedClient' = BlockedClient \cup \{c\}$
$\land \text{ UNCHANGED } \langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, ServerMsg,$
$\qquad BlockedThread, OpCount, Pt, Cp, CalState, State, SnapshotTable,$
$\qquad History, CurrentTerm, ReadyToServe, SyncSource\rangle$

$ClientGetRequest\_majority\_secondary \triangleq$
$\land\ ReadyToServe > 0$

$\land \exists\, k \in Key,\ c \in Client \setminus BlockedClient,\ s \in Secondary :$
$\land\ InMsgs' = [InMsgs \text{ EXCEPT } ![s] = Append(@,$
$[op \mapsto \text{“get”},\ c \mapsto c,\ rc \mapsto \text{“major”},\ k \mapsto k,\ ct \mapsto Ct[c],\ ot \mapsto Ot[c]])]$
$\land\ BlockedClient' = BlockedClient \cup \{c\}$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, ServerMsg,$
$BlockedThread, OpCount, Pt, Cp, CalState, State, SnapshotTable,$
$History, CurrentTerm, ReadyToServe, SyncSource\rangle$

$ClientGetRequest\_linearizable \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, k \in Key,\ c \in Client \setminus BlockedClient,\ s \in Primary :$
$\land\ InMsgs' = [InMsgs \text{ EXCEPT } ![s] = Append(@,$
$[op \mapsto \text{“get”},\ c \mapsto c,\ rc \mapsto \text{“linea”},\ k \mapsto k,\ ct \mapsto Ct[c],\ ot \mapsto Ot[c]])]$
$\land\ BlockedClient' = BlockedClient \cup \{c\}$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, ServerMsg,$
$BlockedThread, OpCount, Pt, Cp, CalState, State, SnapshotTable,$
$History, CurrentTerm, ReadyToServe, SyncSource\rangle$

Client Put

****
DH modified: change the state of history when write with w:0
****

$ClientPutRequest\_zero \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, k \in Key,\ v \in Value,\ c \in Client \setminus BlockedClient,\ s \in Primary :$
$\land\ OpCount[c] \neq 0$
$\land\ InMsgs' = [InMsgs \text{ EXCEPT } ![s] =$
$Append(@, [op \mapsto \text{“put”},\ c \mapsto c,\ wc \mapsto \text{“zero”},\ k$
$\mapsto k,\ v \mapsto v,\ ct \mapsto Ct[c]])]$
$\land\ OpCount' = [OpCount \text{ EXCEPT } ![c] = @ - 1]$
$\land\ History' = [History \text{ EXCEPT } ![c] = Append(@, [op \mapsto \text{“put”},\ ts \mapsto InMsgc[c][1].ot,\ k \mapsto k,\ v \mapsto v])]$
$\land$ UNCHANGED $\langle Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc, ServerMsg,$
$BlockedClient, BlockedThread, Pt, Cp, CalState, State, SnapshotTable,$
$CurrentTerm, ReadyToServe, SyncSource\rangle$

$ClientPutRequest\_number \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, k \in Key,\ v \in Value,\ c \in Client \setminus BlockedClient,\ num \in Number,\ s \in Primary :$
$\land\ InMsgs' = [InMsgs \text{ EXCEPT } ![s] =$
$Append(@, [op \mapsto \text{“put”},\ c \mapsto c,\ wc \mapsto \text{“num”},\ num \mapsto num,\ k \mapsto k,\ v \mapsto v,\ ct \mapsto Ct[c]])]$
$\land\ BlockedClient' = BlockedClient \cup \{c\}$
$\land$ UNCHANGED $\langle OpCount, Primary, Secondary, Oplog, Store, Ct, Ot, InMsgc,$
$ServerMsg, BlockedThread, Pt, Cp, CalState, State, SnapshotTable,$
$History, CurrentTerm, ReadyToServe, SyncSource\rangle$

$ClientPutRequest\_majority \triangleq$

$\land$ $ReadyToServe > 0$
$\land \exists\, k \in Key,\ v \in Value,\ c \in Client \setminus BlockedClient,\ s \in Primary$:
$\land$ $InMsgs' = [InMsgs$ EXCEPT $![s] =$
$Append(@,\ [op \mapsto$ "put", $c \mapsto c,\ wc \mapsto$ "major", $k \mapsto k,\ v \mapsto v,\ ct \mapsto Ct[c]])]$
$\land$ $BlockedClient' = BlockedClient \cup \{c\}$
$\land$ UNCHANGED $\langle OpCount,\ Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,\ InMsgc,$
$ServerMsg,\ BlockedThread,\ Pt,\ Cp,\ CalState,\ State,\ SnapshotTable,$
$History,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

****
DH modified: record the k and v in msg to history
****

$ClientGetResponse \triangleq$
$\land$ $ReadyToServe > 0$
$\land \exists\, c \in Client$:
$\land$ $OpCount[c] \neq 0$ 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 $\langle Primary,\ Secondary,\ Oplog,\ Ct,\ Ot,\ InMsgs,\ ServerMsg,$
$BlockedThread,\ Pt,\ Cp,\ CalState,\ State,\ SnapshotTable,$
$CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

****
DH modified: record the k and v in msg to history record ot from server
****

$ClientPutResponse \triangleq$
$\land$ $ReadyToServe > 0$
$\land \exists\, c \in Client$:
$\land$ $OpCount[c] \neq 0$ 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", $ts \mapsto InMsgc[c][1].ot,\ k \mapsto InMsgc[c][1].k,\ v \mapsto InMsgc[c][1].v])]$

$\land\ InMsgc' = [InMsgc \text{ EXCEPT } ![c] = Tail(@)]$
$\land\ BlockedClient' = \text{IF } Len(InMsgc'[c]) = 0$
$\quad \text{THEN } BlockedClient \setminus \{c\}$
$\quad \text{ELSE } BlockedClient$ remove blocked state
$\land\ OpCount' = [OpCount \text{ EXCEPT } ![c] = @ - 1]$
$\land \text{UNCHANGED } \langle Primary, Secondary, Oplog, Store, InMsgs, ServerMsg,$
$BlockedThread, Pt, Cp, CalState, State, SnapshotTable,$
$CurrentTerm, ReadyToServe, SyncSource\rangle$

Action Wrapper

$ClientGetRequest\_local \triangleq \lor ClientGetRequest\_local\_primary$
$\lor ClientGetRequest\_local\_secondary$
$ClientGetRequest\_majority \triangleq \lor ClientGetRequest\_majority\_primary$
$\lor ClientGetRequest\_majority\_secondary$

all possible client get actions
$ClientGetRequest \triangleq \lor ClientGetRequest\_local$
$\lor ClientGetRequest\_majority$
$\lor ClientGetRequest\_linearizable$

all possible clent put actions
$ClientPutRequest \triangleq \lor ClientPutRequest\_zero$
$\lor ClientPutRequest\_number$
$\lor ClientPutRequest\_majority$

all possible server get actions
$ServerGetReply \triangleq \lor ServerGetReply\_local\_sleep$
$\lor ServerGetReply\_local\_wake$
$\lor ServerGetReply\_majority\_sleep$
$\lor ServerGetReply\_majority\_wake$
$\lor ServerGetReply\_linearizable\_sleep$
$\lor ServerGetReply\_linearizable\_wake$

all possible server put actions
$ServerPutReply \triangleq \lor ServerPutReply\_zero$
$\lor ServerPutReply\_number\_sleep$
$\lor ServerPutReply\_majority\_sleep$
$\lor ServerPutReply\_number\_wake$
$\lor ServerPutReply\_majority\_wake$

$RollbackOplogAction \triangleq \exists\, i, j \in Server : RollbackOplog(i, j)$

$ReplicateAction \triangleq \exists\, i, j \in Server : Replicate(i, j)$

$ElectPrimaryAction \triangleq$
$\exists\, i \in Server : \exists\, majorNodes \in \text{SUBSET } (Server) : ElectPrimary(i, majorNodes)$

Next state for all configurations

$$
\begin{aligned}
Next \triangleq\ & \lor ClientGetRequest \lor ClientPutRequest \\
& \lor ClientGetResponse \lor ClientPutResponse \\
& \lor ServerGetReply \lor ServerPutReply \\
& \lor ReplicateAction \\
& \lor AdvancePt \\
& \lor ServerTakeHeartbeat \\
& \lor ServerTakeUpdatePosition \\
& \lor Snapshot \\
& \lor Stepdown \\
& \lor RollbackOplogAction \\
& \lor TurnOnReadyToServe \\
& \lor ElectPrimaryAction
\end{aligned}
$$

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

$$
\begin{aligned}
Next\_Except\_ClientRequset \triangleq\ & \lor ClientGetResponse \\
& \lor ClientPutResponse \\
& \lor ServerGetReply \\
& \lor ServerPutReply \\
& \lor ReplicateAction \\
& \lor AdvancePt \\
& \lor ServerTakeHeartbeat \\
& \lor ServerTakeUpdatePosition \\
& \lor Snapshot \\
& \lor Stepdown \\
& \lor RollbackOplogAction \\
& \lor TurnOnReadyToServe \\
& \lor ElectPrimaryAction
\end{aligned}
$$

$$
\begin{aligned}
ClientRequset\_1 \triangleq\ & \lor ClientPutRequest\_majority \\
& \lor ClientGetRequest\_local\_primary
\end{aligned}
$$

$$
\begin{aligned}
ClientRequset\_2 \triangleq\ & \lor ClientPutRequest\_majority \\
& \lor ClientGetRequest\_local\_secondary
\end{aligned}
$$

$$
\begin{aligned}
ClientRequset\_3 \triangleq\ & \lor ClientPutRequest\_majority \\
& \lor ClientGetRequest\_majority\_primary
\end{aligned}
$$

$$
\begin{aligned}
ClientRequset\_4 \triangleq\ & \lor ClientPutRequest\_majority \\
& \lor ClientGetRequest\_majority\_secondary
\end{aligned}
$$

$$
\begin{aligned}
ClientRequset\_5 \triangleq\ & \lor ClientPutRequest\_majority \\
& \lor ClientGetRequest\_linearizable
\end{aligned}
$$

$$
\begin{aligned}
ClientRequset\_6 \triangleq\ & \lor ClientPutRequest\_number
\end{aligned}
$$

$$\lor\ ClientGetRequest\_local\_primary$$

$$ClientRequset\_7 \triangleq \lor\ ClientPutRequest\_number \lor\ ClientGetRequest\_local\_secondary$$

$$ClientRequset\_8 \triangleq \lor\ ClientPutRequest\_number \lor\ ClientGetRequest\_majority\_primary$$

$$ClientRequset\_9 \triangleq \lor\ ClientPutRequest\_number \lor\ ClientGetRequest\_majority\_secondary$$

$$ClientRequset\_10 \triangleq \lor\ ClientPutRequest\_number \lor\ ClientGetRequest\_linearizable$$

$$Next1 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_1$$

$$Next2 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_2$$

$$Next3 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_3$$

$$Next4 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_4$$

$$Next5 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_5$$

$$Next6 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_6$$

$$Next7 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_7$$

$$Next8 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_8$$

$$Next9 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_9$$

$$Next10 \triangleq \lor\ Next\_Except\_ClientRequset \lor\ ClientRequset\_10$$

$$Spec1 \triangleq Init \land \Box[Next1]_{vars}$$
$$Spec2 \triangleq Init \land \Box[Next2]_{vars}$$
$$Spec3 \triangleq Init \land \Box[Next3]_{vars}$$
$$Spec4 \triangleq Init \land \Box[Next4]_{vars}$$
$$Spec5 \triangleq Init \land \Box[Next5]_{vars}$$
$$Spec6 \triangleq Init \land \Box[Next6]_{vars}$$

$Spec7 \triangleq Init \wedge \Box[Next7]_{vars}$
$Spec8 \triangleq Init \wedge \Box[Next8]_{vars}$
$Spec9 \triangleq Init \wedge \Box[Next9]_{vars}$
$Spec10 \triangleq Init \wedge \Box[Next10]_{vars}$

Causal Specifications

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

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

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

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

\ ∗ Modification *History*
\ ∗ Last modified *Thu Apr* 07 10:28:09 *CST* 2022 by *dh*
\ ∗ Created *Thu Mar* 31 20:33:19 *CST* 2022 by *dh*