MODULE *TunableMongoDB_Repl*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*

constants and variables

CONSTANTS *Client*, *Server*, the set of clients and servers
Key, *Value*, the set of keys and values
Nil, model value, place holder
PtStop max physical time

VARIABLES *Primary*, Primary node
Secondary, secondary nodes
Oplog, *oplog*[*s*]: *oplog* at *server*[*s*]
Store, *store*[*s*]: data stored at *server*[*s*]
Ct, *Ct*[*s*]: cluster time at node *s*
Ot, *Ot*[*s*]: the last applied operation time at server *s*
ServerMsg, *ServerMsg*[*s*]: the channel of heartbeat *msgs* at server *s*
Pt, *Pt*[*s*]: physical time at server *s*
Cp, *Cp*[*s*]: majority commit point at server *s*
State, *State*[*s*]: the latest *Ot* of all servers that server *s* knows
CalState, *CalState*: sorted *State*[*Primary*]
CurrentTerm, *CurrentTerm*[*s*]: current election term at server *s*
→ updated in *update_position*, heartbeat and replicate
ReadyToServe, equal to 0 before any primary is elected
SyncSource *SyncSource*[*s*]: sync source of server node *s*

ASSUME $Cardinality(Client) \geq 1$ 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

$$HLCLt(x, y) \triangleq \text{IF } x.p < y.p \text{ THEN TRUE ELSE IF } x.p = y.p \text{ THEN IF } x.l < y.l \text{ THEN TRUE ELSE FALSE ELSE FALSE}$$

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

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

RECURSIVE $CreateState(\_,\ \_)$ init state
$CreateState(len,\ seq) \triangleq$
IF $len = 0$ THEN seq
ELSE $CreateState(len - 1,\ Append(seq,\ [p \mapsto 0,\ l \mapsto 0]))$

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

Is node i ahead of node j
$NotBehind(i,\ j) \triangleq \lor\ LastTerm(i) > LastTerm(j)$
$\lor\ \land\ LastTerm(i) = LastTerm(j)$
$\land\ Len(Oplog[i]) \geq Len(Oplog[j])$

$IsMajority(servers) \triangleq Cardinality(servers) * 2 > Cardinality(Server)$

Return the maximum value from a set, or undefined if the set is empty.
$MaxVal(s) \triangleq$ CHOOSE $x \in s : \forall\, y \in s : x \geq y$

commit point
RECURSIVE $AddState(\_,\ \_,\ \_)$
$AddState(new,\ state,\ index) \triangleq$
IF $index = 1 \land HLCLt(new,\ state[1])$
THEN $\langle new\rangle \circ state$ less than the first
ELSE IF $index = Len(state) + 1$
THEN $state \circ \langle new\rangle$
ELSE IF $HLCLt(new,\ state[index])$
THEN $SubSeq(state,\ 1,\ index\ - 1) \circ \langle new\rangle \circ SubSeq(state,\ index,\ Len(state))$
ELSE $AddState(new,\ state,\ index + 1)$

RECURSIVE $RemoveState(\_,\ \_,\ \_)$
$RemoveState(old,\ state,\ index) \triangleq$
IF $state[index] = old$
THEN $SubSeq(state,\ 1,\ index - 1) \circ SubSeq(state,\ index + 1,\ Len(state))$
ELSE $RemoveState(old,\ state,\ index + 1)$

$AdvanceState(new,\ old,\ state) \triangleq AddState(new,\ RemoveState(old,\ state,\ 1),\ 1)$

clock

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

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

heartbeat

Only *Primary* node sends heartbeat once advance pt

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

Can node i sync from node j?

$$
\begin{array}{l}
CanSyncFrom(i,\ j) \triangleq \\
\quad \land Len(Oplog[i]) < Len(Oplog[j]) \\
\quad \land LastTerm(i) = LogTerm(j,\ Len(Oplog[i]))
\end{array}
$$

Oplog entries needed to replicate from j to i

$$
\begin{array}{l}
ReplicateOplog(i,\ j) \triangleq \\
\quad \text{LET } len\_i \triangleq Len(Oplog[i]) \\
\qquad len\_j \triangleq Len(Oplog[j]) \\
\quad \text{IN } \text{IF } i \neq Primary \land len\_i < len\_j \\
\qquad\qquad\qquad\qquad \text{THEN } SubSeq(Oplog[j],\ len\_i + 1,\ len\_j) \\
\qquad\qquad\qquad\qquad \text{ELSE } \langle\rangle
\end{array}
$$

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

$$
\begin{array}{ll}
CanRollback(i,\ j) \triangleq & \land Len(Oplog[i]) > 0 \\
& \land Len(Oplog[j]) > 0 \\
& \land CurrentTerm[i] < CurrentTerm[j] \\
& \land \\
& \quad \lor Len(Oplog[i]) > Len(Oplog[j]) \\
& \quad \lor \land Len(Oplog[i]) \leq Len(Oplog[j]) \\
& \quad\ \ \land CurrentTerm[i] \neq LogTerm(j,\ Len(Oplog[i]))
\end{array}
$$

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

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

Init Part

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

$InitServerMsg \triangleq ServerMsg = [s \in Server \mapsto \langle\rangle]$
$InitPt \triangleq Pt = [s \in Server \mapsto 1]$
$InitCp \triangleq Cp = [n \in Server \cup Client \mapsto [p \mapsto 0,\ l \mapsto 0]]$
$InitCalState \triangleq CalState = CreateState(Cardinality(Server),\ \langle\rangle)$
create initial *state(for calculate)*
$InitState \triangleq State = [s \in Server \mapsto [s0 \in Server \mapsto [p \mapsto 0,\ l \mapsto 0]]]$
$InitCurrentTerm \triangleq CurrentTerm = [s \in Server \mapsto 0]$
$InitReadyToServe \triangleq ReadyToServe = 0$
$InitSyncSource \triangleq SyncSource = [s \in Server \mapsto Nil]$

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

Next *State* Actions
Replication Protocol: possible actions

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

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

There are majority nodes agree to elect node i to become primary
$ElectPrimary \triangleq$
$\land\ ReadyToServe > 0$
$\land\ \exists\, i \in Server : \exists\, majorNodes \in \text{SUBSET } (Server) :$
$\land\ \forall\, j \in majorNodes : \land\ NotBehind(i, j)$
$\land\ CurrentTerm[i] \geq CurrentTerm[j]$

$\land\ IsMajority(majorNodes)$

voted nodes for i cannot be primary anymore

$\land\ Primary' = \text{LET } possiblePrimary \triangleq Primary \setminus majorNodes$
$\quad\quad \text{IN } possiblePrimary \cup \{i\}$

add voted nodes into secondaries

$\land\ Secondary' = \text{LET } possibleSecondary \triangleq Secondary \cup majorNodes$
$\quad\quad \text{IN } possibleSecondary \setminus \{i\}$

$\land\ CurrentTerm' = [index \in Server \mapsto \text{IF } index \in (majorNodes \cup \{i\})$
$\quad\quad \text{THEN } CurrentTerm[i] + 1$
$\quad\quad \text{ELSE } CurrentTerm[index]]$

$\land\ \text{UNCHANGED } \langle Oplog, Store, Ct, Ot, ServerMsg, Pt, Cp, State, CalState,$
$\quad\quad CurrentTerm, ReadyToServe, SyncSource\rangle$

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

heartbeatoplogOtstore

$ServerTakeHeartbeat \triangleq$
$\quad \land\ ReadyToServe > 0$
$\quad \land\ \exists s \in Server :$
$\quad\quad \land\ Len(ServerMsg[s]) \neq 0$ message channel is not empty
$\quad\quad \land\ ServerMsg[s].type = \text{“heartbeat”}$
$\quad\quad \land\ Ct' = [Ct \text{ EXCEPT } ![s] = HLCMax(Ct[s], ServerMsg[s][1].ct)]$
$\quad\quad \land\ State' =$
$\quad\quad\quad \text{LET } SubHbState \triangleq State[s]$
$\quad\quad\quad\quad hb \triangleq [SubHbState \text{ EXCEPT } ![ServerMsg[s][1].s] =$
$\quad\quad\quad\quad\quad ServerMsg[s][1].aot]$
$\quad\quad\quad \text{IN } [State \text{ EXCEPT } ![s] = hb]$
$\quad\quad \land\ CalState' = \text{LET } newcal \triangleq$
$\quad\quad\quad \text{IF } s \in Primary$ primary node: update $CalState$
$\quad\quad\quad\quad \text{THEN } AdvanceState(ServerMsg[s][1].aot,$
$\quad\quad\quad\quad\quad State[s][ServerMsg[s][1].s], CalState)$
$\quad\quad\quad\quad \text{ELSE } CalState$
$\quad\quad\quad \text{IN } newcal$
$\quad\quad \land\ Cp' = \text{LET } newcp \triangleq$

primary node: compute new mcp

$\quad\quad\quad \text{IF } s \in Primary \text{ THEN } CalState'[Cardinality(Server) \div 2 + 1]$

secondary node: update mcp

$\quad\quad\quad \text{ELSE IF } \neg HLCLt(ServerMsg[s][1].cp, Cp[s])$
$\quad\quad\quad\quad \land \neg HLCLt(Ot[s], ServerMsg[s][1].cp)$
$\quad\quad\quad\quad \text{THEN } ServerMsg[s][1].cp$

```
                    ELSE  Cp[s]
               IN   [Cp EXCEPT ![s] = newcp]
        ∧ ServerMsg' = [ServerMsg EXCEPT ![s] = Tail(@)]
        ∧ CurrentTerm' = [CurrentTerm EXCEPT ![s] = Max(CurrentTerm[s], ServerMsg[s][1].term)]
    ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ot, Pt,
                 ReadyToServe, SyncSource⟩

ServerTakeUpdatePosition ≜
    ∧ ReadyToServe > 0
    ∧ ∃ s ∈ Server :
        ∧ Len(ServerMsg[s]) ≠ 0   message channel is not empty
        ∧ ServerMsg[s].type  = "update_position"
        ∧ Ct' = [Ct EXCEPT ![s] = HLCMax(Ct[s], ServerMsg[s][1].ct)]  update ct accordingly
        ∧ State' =
            LET SubHbState ≜ State[s]
                hb ≜ [SubHbState EXCEPT ![ServerMsg[s][1].s] =
                      ServerMsg[s][1].aot]
            IN   [State EXCEPT ![s] = hb]
        ∧ CalState' = LET newcal ≜
                      IF s ∈ Primary  primary node: update CalState
                      THEN  AdvanceState(ServerMsg[s][1].aot,
                                         State[s][ServerMsg[s][1].s], CalState)
                      ELSE  CalStateIN   newcal
        ∧ Cp' = LET newcp ≜
                 primary node: compute new mcp
                IF s ∈ Primary THEN CalState'[Cardinality(Server) ÷ 2 + 1]
                 secondary node: update mcp
                 ELSE  IF ¬HLCLt(ServerMsg[s][1].cp, Cp[s])
                          ∧ ¬HLCLt(Ot[s], ServerMsg[s][1].cp)
                 THEN ServerMsg[s][1].cp
                 ELSE  Cp[s]
                IN   [Cp EXCEPT ![s] = newcp]
        ∧ ServerMsg' = LET newServerMsg ≜ [ServerMsg EXCEPT ![s] = Tail(@)]
                           appendMsg ≜ [type ↦ "update_position", s ↦ ServerMsg[s][1].s, aot ↦ ServerMs
                                        ct ↦ ServerMsg[s][1].ct, cp ↦ ServerMsg[s][1].cp, term ↦ Server
                           newMsg ≜ IF s ∈ Primary
                                    THEN newServerMsg  If s is primary, accept the msg, else forward it to its sy
                                    ELSE [newServerMsg EXCEPT ![SyncSource[s]] = Append(newServer
                       IN   newMsg
        ∧ CurrentTerm' = [CurrentTerm EXCEPT ![s] = Max(CurrentTerm[s], ServerMsg[s][1].term)]
    ∧ UNCHANGED ⟨Primary, Secondary, Oplog, Store, Ot,
                 Pt, ReadyToServe, SyncSource⟩

NTPSync ≜   simplify NTP protocal
    ∧ ReadyToServe > 0
```

$\land\ Pt' = [s \in Server \mapsto MaxPt]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Oplog,\ Store,\ Ct,\ Ot,$
$ServerMsg,\ Cp,$
$CalState,\ State,\ CurrentTerm,\ ReadyToServe,\ SyncSource\rangle$

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

Replicate oplog from node j to node i, and update related structures accordingly
$Replicate \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, i,\ j \in Server :$
$\land\ CanSyncFrom(i,\ j)$ i can sync from j only need not to rollback
$\land\ i \in Secondary$
$\land\ ReplicateOplog(i,\ j) \neq \langle\rangle$
$\land\ Oplog' = [Oplog$ EXCEPT $![i] = @ \circ ReplicateOplog(i,\ j)]$
$\land\ Store' = [Store$ EXCEPT $![i] = Store[j]]$
$\land\ Ct' = [Ct$ 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 $Curren$
$\land\ State' =$
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", $s \mapsto i,\ aot \mapsto Ot'[i],\ ct \mapsto Ct'[i],\ cp \mapsto Cp'[i],\ term \mapsto Curr$
IN $ServerMsg' = [ServerMsg$ EXCEPT $![j] = Append(ServerMsg[j],\ msg)]$
$\land\ SyncSource' = [SyncSource$ EXCEPT $![i] = j]$
$\land$ UNCHANGED $\langle Primary,\ Secondary,\ Pt,\ CalState,$
$ReadyToServe\rangle$

Rollback i's oplog and recover it to j's state
Recover to j's state immediately to prevent internal client request
$RollbackAndRecover \triangleq$
$\land\ ReadyToServe > 0$
$\land \exists\, i,\ j \in Server :$
$\land\ i \in Secondary$
$\land\ CanRollback(i,\ j)$
$\land$ LET $cmp \triangleq RollbackCommonPoint(i,\ j)$ 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 SubHbState ≜ State[i]
                   hb ≜ [SubHbState EXCEPT ![j] = Ot[j]]
               IN  [State EXCEPT ![i] = hb]  update j's state i knows
    ∧ LET msg ≜ [type ↦ "update_position", s ↦ i, aot ↦ Ot'[i], ct ↦ Ct'[i], cp ↦ Cp'[i]]
      IN  ServerMsg' = [ServerMsg EXCEPT ![j] = Append(ServerMsg[j], msg)]
    ∧ SyncSource' = [SyncSource EXCEPT ![i] = j]
    ∧ UNCHANGED ⟨Primary, Secondary, Pt, CalState,
                 ReadyToServe⟩

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

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

$\lor\ NTPSync$

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

Properties to check?

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

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

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

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

\ * Modification *History*
\ * Last modified *Tue Apr* 19 22:17:02 *CST* 2022 by *dh*
\ * Created *Mon Apr* 18 11:38:53 *CST* 2022 by *dh*