# Tiga TLA+ Specification

MODULE *Tiga*

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

## Bounds for Model Check [Configurable]

Time Range [Configurable]

$MaxTime \triangleq 3$

In *Tiga*, we assume client and coordinator are co-located
In this spec, we use "coordinator" to represent them
Each coordinator is only allowed to submit *MaxReqNum* requests [Configurable]
In the specification, we will only consider two roles, client and replicas
(*i.e.* it can be considered as co-locating one proxy with one client)
For the proxy-based design, we just need to replace client with proxy,
and then the specification describes the interaction between proxy and replicas

$MaxReqNum \triangleq 1$

The leader is only allowed to crash when the view $< MaxViews$ [Configurable]

$MaxViews \triangleq 3$

The set of replicas and an ordering of them [Can be configured in TLA+ *Toolbox*]

```
Replicas ≜ 0 . . 2
ReplicaOrder ≜ ⟨0, 1, 2⟩
Shards ≜ 0 . . 2
Coords ≜ 0 . . 1
LatencyBounds ≜ [c ∈ Coords ↦ 1]

ASSUME IsFiniteSet(Replicas)
ASSUME IsFiniteSet(Shards)
ASSUME ReplicaOrder ∈ Seq(Replicas)
Servers ≜ {
    [
        replicaId ↦ e[1],
        shardId ↦ e[2]
    ] : e ∈ Replicas × Shards
}
```

These variables are used to implment at-most-once primitives

## Constants

$F \triangleq (Cardinality(Replicas) - 1) \div 2$

$ceilHalfF \triangleq \text{IF } (F \div 2) * 2 = F \text{ THEN } F \div 2 \text{ ELSE } (F + 1) \div 2$
$floorHalfF \triangleq F \div 2$
$QuorumSize \triangleq F + 1$
$FastQuorumSize \triangleq F + ceilHalfF + 1$
$RecoveryQuorumSize \triangleq ceilHalfF + 1$
$FastQuorums \triangleq \{R \in \text{SUBSET } (Replicas) :$
$\quad Cardinality(R) \geq FastQuorumSize\}$
$Quorums \triangleq \{R \in \text{SUBSET } (Replicas) :$
$\quad Cardinality(R) * 2 > Cardinality(Replicas)\}$

## Server Status

$StNormal \triangleq 1$
$StViewChange \triangleq 2$
$StCrossShardSyncing \triangleq 3$
$StRecovering \triangleq 4$
$StFailing \triangleq 5$

## Message Types

$MTxn \triangleq 1$
$MLogEntry \triangleq 2$ *Log* entry, different from index, it includes command field, which can be large in practice
$MDeadlineNotification \triangleq 3$ Leaders send the message to other leaders for deadline agreement
$MInterReplicaSync \triangleq 4$ Synchronize within shard group (across replicas) to ensure strict serializability
$MFastReply \triangleq 5$ Fast Reply Message
$MSlowReply \triangleq 6$ Slow Reply Message

The following messages are mainly for view change within each sharding group
$MViewChangeReq \triangleq 7$ Sent by config manager when leader/sequencer failure detected
$MViewChange \triangleq 8$ Sent to *ACK* view change
$MStartView \triangleq 9$ Sent by new leader to start view

The following messages are mainly used for periodic sync
Just as described in *NOPaxos*, it is an optional optimization to enable fast recovery after failure
$MLocalSyncStatus \triangleq 10$ Sent by the leader to ensure *log* durability
$MLocalCommit \triangleq 11$ Sent by followers as *ACK*
The following messages are used for periodic sync across sharding groups
This is an optional optimization to enable fast recovery
$MPeerShardCommitStatus \triangleq 12$

The following messages are mainly used for server recovery
$MCrashVectorReq \triangleq 13$
$MCrashVectorRep \triangleq 14$
$MRecoveryReq \triangleq 15$
$MRecoveryRep \triangleq 16$
$MStartViewReq \triangleq 17$

$MCrossShardConfirm \triangleq 19$

Config Manager ($CM$)'s Operations. Since $CM$ is supported by typical viewstamped replication ($VR$), in this spec, we do not repeat the $VR$'s failure recovery spec for $CM$

$MCMPrepare \triangleq 20$
$MCMPrepareReply \triangleq 21$
$MCMCommit \triangleq 22$

## Message Schemas

Each server is identified by a combination of $< replicaId, shardId >$ $TxnID$ uniquely identifies one request on one server But across replicas, the same $TxnID$ may have different deadlines (the leader may modify the deadline to make the request eligible to enter the early-buffer) so $< deadline, txnId >$ uniquely identifes one request across replicas

```
TxnID = [
    coordId ↦ i in (1 . . ),
    rId     ↦ i in (1 . . )
]
```

```
Txn = [
    mtype ↦ MTxn
    txnId ↦ TxnID,
    shards ↦ Shards,
    command ↦ command,
    st      ↦ sendTime,
    bound ↦ latencyBound
]
```

```
LogEntry = [
    mtype ↦ MLogEntry
    txnId ↦ TxnID,
    shards ↦ Shards,
    command ↦ command,
    deadline ↦ deadline
]
```

After the request arrives at the *shards* and is placed into its early buffer (either with deadline modified or not), the server will broadcast *DeadlineNotification* to all the other servers in the same replica group to tell them the deadline of the request on its own server

```
DeadlineNotification = [
    mtype ↦ MDeadlineNotification,
    gView ↦ 0 . . . x
    lView ↦ 0 . . . y
    sender ↦ src ∈ Servers,
    dest ↦ dst ∈ Servers,
    entry ↦ LogEntry
]
```

After leader has released the *txn*, it synchornizes the *log* with its followers. If followers are inconsistent, they will rectify their logs to keey consistent with leader

$InterReplicaSync \triangleq [$

  $mtype \mapsto MInterReplicaSync,$

  $lView \mapsto 0 \ldots y$

  $sender \mapsto src \in Servers,$

  $dest \mapsto dst \in Servers,$

  $entries \mapsto [LogEntry \ldots]$

$]$

*logId* (*i.e.*, the position index of the *log* entry in the *log* list) is not necessary and it is not described in the paper. Here we include *logSlotNum* in *FastReply* and *SlowReply* messages to facilitate the check of *Linearizability* invariant

$FastReply \triangleq [$

  $mtype \mapsto MFastReply,$

  $sender \mapsto src \in Servers,$

  $dest \mapsto dst \in Coords,$

  $gView \mapsto 0 \ldots x$

  $lView \mapsto 0 \ldots x$

  $txnId \mapsto txnId$

  In real implementation, we use $SHA1 +$ Incremental Hash

  $hash \mapsto [\ entries \mapsto log$ entries so far $cv \mapsto crashVector\ ]$

  $deadline \mapsto i \in (1 \mathrel{..} MaxTime + MaxBound),$

  $logId \mapsto n \in (1 \mathrel{..})$

$]$

$SlowReply \triangleq [$

  $mtype \mapsto MSlowReply,$

  $sender \mapsto src \in Servers,$

  $dest \mapsto c \in Coords,$

  $gView \mapsto 0 \ldots x$

  $lView \mapsto 0 \ldots x$

  $txnId \mapsto txnId$

  $logId \mapsto n \in (1 \mathrel{..})$

$]$

$ViewChangeReq \triangleq [$

  $mtype \mapsto MViewChangeReq,$

  $sender \mapsto src \in Replicas,$ (by *configManager*)

  $dest \mapsto dst \in Servers,$

  $gView \mapsto 0 \mathrel{..} x$

  $gVec \mapsto$ the *lViews* for each shard

$]$

$ViewChange \triangleq [$

  $mtype \mapsto MViewChange,$

  $sender \mapsto src \in Servers,$

  $dest \mapsto dst \in Servers,$

  $gView \mapsto 0 \mathrel{..} x$

  $gVec \mapsto$ the *lViews* for each shard

  $lView \mapsto 0 \ldots x$

  $lastNormal \mapsto v \in ViewIDs,$

```
    $lSyncPoint \mapsto 0 \mathrel{..}$
    $entries \mapsto l \in vLogs[1 \mathrel{..} n]$,
    $cv \mapsto$ crash vector
]

$CrossShardConfirm =$ [
    $mtype \mapsto MCrossShardConfirm$,
    $sender \mapsto src \in Servers$,
    $dest \mapsto dst \in Servers$,
    $lView \mapsto 0 \ldots x$
    $gView \mapsto 0 \ldots$
    $entries \mapsto l \in vLogs[1 \mathrel{..} n]$
]

$StartView =$ [
    $mtype \mapsto MStartView$,
    $sender \mapsto src \in Servers$,
    $dest \mapsto dst \in Servers$,
    $lView \mapsto 0 \ldots x$
    $gView \mapsto 0 \ldots x$
    $gVec \mapsto$ the $lViews$ for each shard
    $entries \mapsto l \in vLogs[1 \mathrel{..} n]$,
    $cv \mapsto$ crash vector
]

$CrashVectorReq =$ [
    $mtype \mapsto MCrashVectorReq$,
    $sender \mapsto src \in Servers$,
    $dest \mapsto dst \in Servers$,
    $nonce \mapsto nonce$
]

$CrashVectorRep =$ [
    $mtype \mapsto MCrashVectorRep$,
    $sender \mapsto src \in Servers$,
    $dest \mapsto dst \in Servers$,
    $nonce \mapsto nonce$,
    $cv \mapsto$ vector of counters
]

$RecoveryReq =$ [
    $mtype \mapsto MRecoveryReq$,
    $sender \mapsto src \in Servers$,
    $dest \mapsto dst \in Servers$,
    $cv \mapsto$ vector of counters
]

$RecoveryRep =$ [
    $mtype \mapsto MRecoveryRep$,
    $sender \mapsto src \in Servers$,
    $dest \mapsto dst \in Servers$,
    $gView \mapsto 0 \mathrel{..} x$
```

$lView \mapsto 0 \mathrel{..} x$
$cv \mapsto vector\ of\ counters$
]

$StartViewReq = [$
$mtype \mapsto MStartViewReq,$
$sender \mapsto src \in Servers,$
$dest \mapsto dst \in Servers,$
$lView \mapsto 0 \mathrel{..} x$
$cv \mapsto vector\ of\ counters$
]

Follower reports to its leader

$LocalSyncStatus = [$
$mtype \mapsto MLocalSyncStatus,$
$sender \mapsto src \in Servers,$
$dest \mapsto dst \in Servers,$
$lView \mapsto 0 \ldots x$
$lSyncPoint \mapsto n \in (1 \mathrel{..} )$
$cv \mapsto$ vector of counters
]

Leader notifies its followers

$LocalCommit = [$
$mtype \mapsto MLocalCommit,$
$sender \mapsto src \in Servers,$
$dest \mapsto dst \in Servers,$
$lView \mapsto 0 \ldots x$
$entries \mapsto log$ entries
$lCommitPoint \mapsto n \in (1 \ldots)$
]

Each server tells its neighbors (the servers in the same region but belong to different *shards*) its local commit status. This is optional optimization (only for checkpoint and failure recovery acceleration)

$PeerShardCommitStatus = [$
$mtype \mapsto MPeerShardCommitStatus,$
$sender \mapsto src \in Servers,$
$dest \mapsto dst \in Servers,$
$gView \mapsto 0 \mathrel{..} x$
$deadline \mapsto the\ largest\ committed\ deadline$
]

Configuration Manager ($CM$)'s message to prepare global information (including $gView$ and $gVec$)

In our implementation, $CM$ is co-located on Shard $- 0$, but from design perspective, $CM$ is completed standalone and decoupled from $Tiga\ Servers$

$CMPrepare = [$

```
    mtype ↦ MCMPrepare,
    sender ↦ src ∈ Servers,
    dest ↦ dst ∈ Servers,
    cView ↦ 0 .. x
    gView ↦ 0 .. x
    gVec ↦ [shardId ↦ lView]
]
CMPrepareReply = [
    mtype ↦ MCMPrepareReply,
    sender ↦ src ∈ Servers,
    dest ↦ dst ∈ Servers,
    cView ↦ 0 .. x
    gView ↦ 0 .. x
]
CMCommit = [
    mtype ↦ MCMPrepareReply,
    sender ↦ src ∈ Servers,
    dest ↦ dst ∈ Servers,
    cView ↦ 0 .. x
    gView ↦ 0 .. x
]
```

---

**Network State**

VARIABLES *messages* Set of all messages sent

**Server State**

VARIABLES

Messages that have been processed by servers

*vServerProcessed*,

*Log* list of entries

*vLog*,

The sequencer to hold txns and release it after clock passes its deadline ($s + l$)

*vEarlyBuffer*,

The buffer to hold txns on followers because these txns come too late and cannot enter early-buffer

*vLateBuffer*,

Each leader server has a data structure of *DeadlineQuroum* to collect the deadlines from other servers for agreement

*vDeadlineQuorum*,

After servers have recovered their logs from the signle shard, they need confirmation from the other *shards* to ensure the recovered logs satisfy strict serializability

*vCrossShardConfirmQuorum*,

One of *StNormal*, *StViewChange*, *StFailing*, *StRecovering*

$vServerStatus$,

Global views of each server

$vGView$,

The g-vecs of each server

$vGVec$,

Local views of each server

$vLView$,

Current Time of the server

$vServerClock$,

Last *lView* in which this server had *StNormal* status

$vLastNormView$,

Used for collecting view change votes

$vViewChange$,

*vLSyncPoint* indicates to which the server state (*vLog*) is consistent with the leader.

$vLSyncPoint$,

*vLCommitPoint* indicates that the *log* entries before this point has been locally committed, *i.e.*, replicated to majority in this sharding groups. So followers can safely execute the logged txns

$vLCommitPoint$,

*vPeerCommitDeadline* records the peer's largest deadline that has been locally committed. This can be used to save data transfer during cross-shard confirmation

$vPeerCommitDeadline$,

*vLSyncQuorum* is used by each leader to collect the *LocalSyncStatus* messages from servers in the same sharding group

$vLSyncQuorum$,

Locally unique string (for *CrashVectorReq*)

$vUUIDCounter$,

*CrashVector*, initialized as all-zero vector

$vCrashVector$,
$vCrashVectorReps$,
$vRecoveryReps$

**Coordinator State**

VARIABLES Current Clock Time of the coordinator

$vCoordClock$,

The txns that have been sent by this coordinator. This variable makes it easy to derive the Invariants

$vCoordTxns$,

Messages that have been processed by coordinators

$vCoordProcessed$

**Configuration Manager (CM) State**

VARIABLES

Since *CM* is supported by traditional *VR*, here we do not want to repeat *VR*'s failure recovery in this spec, so we make *CMStatus* always *StNormal*

$vCMStatus$,
$vCMView$,

Config Manager: the latest global info the manager maintains (*gView* and *gVec*)

$vCMGInfo$,
$vCMPrepareGInfo$,

Config Manager: quorum of *CMPrepareReplies*

$vCMPrepareReps$,

$vCMProcessed$

VARIABLES $ActionName$

$networkVars \triangleq \langle messages \rangle$

$serverStateVars \triangleq$
$\langle vLog, vEarlyBuffer, vLateBuffer,$
$vDeadlineQuorum, vCrossShardConfirmQuorum, vServerStatus,$
$vGView, vGVec, vLView, vServerClock, vLastNormView,$
$vViewChange, vLSyncPoint, vLCommitPoint,$
$vPeerCommitDeadline, vLSyncQuorum,$
$vUUIDCounter, vCrashVector, vCrashVectorReps,$
$vRecoveryReps, vServerProcessed\rangle$

$coordStateVars \triangleq \langle vCoordClock, vCoordTxns, vCoordProcessed\rangle$

$configManagerStateVars \triangleq \langle vCMStatus, vCMView, vCMGInfo,$
$vCMPrepareGInfo, vCMPrepareReps,$
$vCMProcessed\rangle$

$InitNetworkState \triangleq messages = \{\}$

$InitServerState \triangleq$
$\land\ vServerProcessed = [serverId \in Servers \mapsto \{\}]$
$\land\ vLog = [serverId \in Servers \mapsto \langle\rangle]$
$\land\ vEarlyBuffer = [serverId \in Servers \mapsto \{\}]$
$\land\ vLateBuffer = [serverId \in Servers \mapsto \{\}]$
$\land\ vDeadlineQuorum = [serverId \in Servers \mapsto \{\}]$
$\land\ vCrossShardConfirmQuorum = [serverId \in Servers \mapsto \{\}]$
$\land\ vServerStatus = [serverId \in Servers \mapsto StNormal]$
$\land\ vGView = [serverId \in Servers \mapsto 0]$
$\land\ vGVec = [$

```
        $serverId \in Servers \mapsto [$
            $shardId \in Shards \mapsto 0$
        $]$
    $]$
  $\land$ $vLView = [serverId \in Servers \mapsto 0]$
  $\land$ $vServerClock = [serverId \in Servers \mapsto 1]$
  $\land$ $vLastNormView = [serverId \in Servers \mapsto 0]$
  $\land$ $vViewChange = [serverId \in Servers \mapsto \{\}]$
  $\land$ $vLSyncPoint = [serverId \in Servers \mapsto 0]$
  $\land$ $vLCommitPoint = [serverId \in Servers \mapsto 0]$
  $\land$ $vPeerCommitDeadline = [serverId \in Servers \mapsto$
        $[shardId \in Shards \mapsto 0]$
    $]$
  $\land$ $vLSyncQuorum = [serverId \in Servers \mapsto \{\}]$
  $\land$ $vUUIDCounter = [serverId \in Servers \mapsto 0]$
  $\land$ $vCrashVector = [$
        $serverId \in Servers \mapsto [$
            $rr \in Replicas \mapsto 0$
        $]$
    $]$
  $\land$ $vCrashVectorReps = [serverId \in Servers \mapsto \{\}]$
  $\land$ $vRecoveryReps = [serverId \in Servers \mapsto \{\}]$


$InitCoordState \triangleq$
  $\land$ $vCoordProcessed = [c \in Coords \mapsto \{\}]$
  $\land$ $vCoordClock = [c \in Coords \mapsto 1]$
  $\land$ $vCoordTxns = [c \in Coords \mapsto \{\}]$

$InitConfigManagerState \triangleq$
  $\land$ $vCMStatus = [$
        $replicaId \in Replicas \mapsto StNormal$
    $]$
  $\land$ $vCMView = [$
        $replicaId \in Replicas \mapsto 0$
    $]$
  $\land$ $vCMGInfo = [$
        $replicaId \in Replicas \mapsto [$
            $gView \mapsto 0,$
            $gVec \mapsto [shardId \in Shards \mapsto 0]$
        $]$
    $]$
  $\land$ $vCMPrepareGInfo = [$
        $replicaId \in Replicas \mapsto [$
            $gView \mapsto 0,$
            $gVec \quad\quad \mapsto [shardId \in Shards \mapsto 0]$
        ]
    ]
$\land$ $vCMPrepareReps = [$
        $replicaId \in Replicas \mapsto \{\}$
    ]
$\land$ $vCMProcessed = [$
        $replicaId \in Replicas \mapsto \{\}$
    ]
```

$PickMax(S) \triangleq \text{CHOOSE } x \in S : \forall y \in S : y \leq x$

$PickMin(S) \triangleq \text{CHOOSE } x \in S : \forall y \in S : y \geq x$

$Min(a, b) \triangleq \text{IF } a < b \text{ THEN } a \text{ ELSE } b$

$Max(a, b) \triangleq \text{IF } a < b \text{ THEN } b \text{ ELSE } a$

$Send(ms) \triangleq messages' = messages \cup ms$

$SeqToSet(s) \triangleq$
  $\{s[i] : i \in \text{DOMAIN } s\}$

$IsInjective(s) \triangleq$

TRUE iff the sequence $s$ contains no duplicates where two elements a, $b$ of $s$ are defined to be duplicates iff $a = b$. In other words,
$Cardinality(ToSet(s)) = Len(s)$

This definition is overridden by *TLC* in the *Java* class *SequencesExt*. The operator is overridden by the *Java* method with the same name.

Also see Functions!Injective operator.

$\forall i, j \in \text{DOMAIN } s : (s[i] = s[j]) \Rightarrow (i = j)$

$SetToSeq(S) \triangleq$

Convert a set to some sequence that contains all the elements of the set exactly once, and contains no other elements.

$\text{CHOOSE } f \in [1 \mathrel{..} Cardinality(S) \rightarrow S] : IsInjective(f)$

$Remove(s, e) \triangleq$

The sequence $s$ with $e$ removed or $s$ iff $e \notin Range(s)$

$SelectSeq(s, \text{LAMBDA } t : t \neq e)$

$SetToSortSeq(S, op(\_, \_)) \triangleq$

Convert a set to a sorted sequence that contains all the elements of the set exactly once, and contains no other elements. Not defined via CHOOSE like *SetToSeq* but with an additional conjunct, because this variant works efficiently without a dedicated *TLC* override.

```
  $SortSeq(SetToSeq(S), op)$
```

**View ID Helpers**

```
$LeaderID(viewId) \triangleq ReplicaOrder[(viewId \% Len(ReplicaOrder)) + 1]$    remember ⟨⟩ are 1-indexed

$isLeader(replicaId, viewId) \triangleq (replicaId = LeaderID(viewId))$

$PrintVal(id, exp) \triangleq Print(\langle id, exp \rangle, \text{TRUE})$

$ViewGreater(gv1, lv1, gv2, lv2) \triangleq$
    IF $gv1 > gv2$ THEN TRUE
     ELSE
        IF  $\land gv1 = gv2$
            $\land lv1 > lv2$
         THEN   TRUE
         ELSE   FALSE
```

Coordinator *c* submits a *txn*. We assume Coordinator can only send one *txn* in one tick of time.
If time has reached the bound, this client cannot send request any more

```
$LastAppendedDeadline(Log) \triangleq$ IF $Len(Log) = 0$ THEN 0
                                     ELSE $Tail(Log).deadline$

$CoordSubmitTxn(c) \triangleq$
    $\land vCoordClock[c] < MaxTime$
    $\land Cardinality(vCoordTxns[c]) < MaxReqNum$
    $\land$ LET
          $txnId \triangleq [$
              $coordId \mapsto c,$
              $rId \mapsto Cardinality(vCoordTxns[c]) + 1$
          $]$
       IN
       $\land Send(\{[mtype \mapsto MTxn,$
              $txnId \mapsto txnId,$
              $command \mapsto$ “”,
               Here we assume involves all shards
              $shards \mapsto Shards,$
              $st \mapsto vCoordClock[c],$
              $bound \mapsto LatencyBounds[c],$
              $sender \mapsto c,$
              $dest \mapsto serverId$
          $] : serverId \in Servers\})$
       $\land vCoordClock' = [vCoordClock$ EXCEPT $![c] = vCoordClock[c] + 1]$
       $\land vCoordTxns' = [vCoordTxns$ EXCEPT $![c] = vCoordTxns[c] \cup \{txnId\}]$
```

```
$HandleTxn(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
        $newLog \triangleq [$
            $mtype \mapsto MLogEntry,$
            $txnId \mapsto m.txnId,$
            $command \mapsto m.command,$
            $shards \mapsto m.shards,$
            $deadline \mapsto Max(LastAppendedDeadline(vLog[myServerId]), m.st + m.bound)$
        $]$
        $serversInOneReplica \triangleq \{s \in Servers : s.replicaId = myServerId.replicaId\}$
    IN
    $\lor$  $\land isLeader(myServerId.replicaId, vLView[myServerId])$
        $\land vEarlyBuffer' = [$
            $vEarlyBuffer$ EXCEPT $![myServerId]$
                $= vEarlyBuffer[myServerId] \cup \{newLog\}]$
         Broadcast deadline notifications to other shards
        $\land Send(\{[$
            $mtype \mapsto MDeadlineNotification,$
            $gView \mapsto vGView[myServerId],$
            $lView \mapsto vLView[myServerId],$
            $sender \mapsto myServerId,$
            $dest \mapsto dstServerId,$
            $entry \mapsto newLog$
            $] : dstServerId \in serversInOneReplica\})$
        $\land$ UNCHANGED $\langle vLateBuffer \rangle$
    $\lor$  $\land \neg isLeader(myServerId.replicaId, vLView[myServerId])$
        $\land$  $\lor$  $\land newLog.deadline = (m.st + m.bound)$
                $\land vEarlyBuffer' = [$
                        $vEarlyBuffer$ EXCEPT $![myServerId]$
                            $= vEarlyBuffer[myServerId] \cup \{newLog\}$
                    $]$
                $\land$ UNCHANGED $\langle vLateBuffer \rangle$
             $\lor$  $\land \neg(newLog.deadline = (m.st + m.bound))$
                $\land$  $vLateBuffer' = [$
                        $vLateBuffer$ EXCEPT $![myServerId]$
                            $= vLateBuffer[myServerId] \cup \{newLog\}$
                    $]$
                $\land$ UNCHANGED $\langle vEarlyBuffer \rangle$
        $\land$ UNCHANGED $\langle networkVars \rangle$


$HandleDeadlineNotification(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
        $quorum \triangleq \{$
            $msg \in vDeadlineQuorum[myServerId]$
                : $\land\ msg.entry.txnId = m.entry.txnId$
                  $\land\ msg.gView = m.gView$
                  $\land\ m.gView = vGView[myServerId]$
            $\} \cup \{m\}$
    IN
     Only leader does deadline agreement
    $\land\ vGView[myServerId] = m.gView$
    $\land\ vGVec[myServerId][m.sender.shardId] = m.lView$
    $\land\ isLeader(myServerId.replicaId,\ vLView[myServerId])$
    $\land\ vDeadlineQuorum' = [$
            $vDeadlineQuorum$ EXCEPT $![myServerId]$
                $= vDeadlineQuorum[myServerId] \cup \{m\}$
        $]$
    $\land$ IF $Cardinality(quorum) = Cardinality(m.entry.shards)$
          THEN
          Deadline quorum established : Update the deadline of the *txn* in Sequencer
            LET
                $maxDeadlineTxn \triangleq$
                    CHOOSE $x \in quorum$ :
                        $\forall y \in quorum$ :
                            $y.entry.deadline \leq x.entry.deadline$
                $sequencingTxn \triangleq$
                    CHOOSE $x \in vEarlyBuffer[myServerId]$ :
                        $x.txnId = m.entry.txnId$
            IN
            IF $maxDeadlineTxn.entry.deadline > sequencingTxn.deadline$
             THEN
                $vEarlyBuffer' = [vEarlyBuffer$ EXCEPT $![myServerId]$
                    $= (vEarlyBuffer[myServerId] \setminus \{sequencingTxn\}) \cup \{maxDeadlineTxn.entry\}]$
             ELSE UNCHANGED $\langle vEarlyBuffer \rangle$

          ELSE
          Deadline quorum not sufficient so far: do not take further actions
            UNCHANGED $\langle vEarlyBuffer \rangle$


$HandleInterReplicaSync(m) \triangleq$
    $\land\ m.lView = vLView[m.dest]$
    Even if $m$'s *crashVector* is newer (larger value), we do not accept *it.The* consistency of
    *crashVector* will finally be solved during viewchange
    $\land\ m.crashVector[m.sender] = vCrashVector[m.sender]$
    $\land\ \neg isLeader(m.dest.replicaId,\ vLView[m.dest])$
    $\land$ LET
```

```
$myServerId \triangleq m.dest$
$syncedTxnIds \triangleq \{m.entries[i].txnId : i \in 1 \mathrel{..} Len(m.entries)\}$
$currentSyncPoint \triangleq Len(vLSyncPoint[myServerId])$
IN
$\lor$  $\land$ $currentSyncPoint < Len(m.entries)$
        $\land$ $vLog' = [vLog$ EXCEPT $![myServerId] = m.entries]$
   Kick synced entries out of earlyBuffer
        $\land$ $vEarlyBuffer' = [$
              $vEarlyBuffer$ EXCEPT $![myServerId]$
                  $= \{msg \in vEarlyBuffer[myServerId] :$
                      $msg.txnId \notin syncedTxnIds\}$
           $]$
   Kick synced entries out of late buffer. In actual implementation, InterReplicaSync only
   carries log indices, and the entries are fetched from Late Buffer first, if still missing,
   then it will go to ask leader. Such a design can save much unncessary transmission in
   practice.
        $\land$ $vLateBuffer' = [$
              $vLateBuffer$ EXCEPT $![myServerId]$
                  $= \{msg \in vLateBuffer[myServerId] :$
                      $msg.txnId \notin syncedTxnIds\}$
           $]$
   Kick synced entries out of deadline quorum. These txns have been synced, no need to
   record in DeadlineQuorum
        $\land$ $vDeadlineQuorum' = [$
              $vDeadlineQuorum$ EXCEPT $![myServerId]$
                  $= \{msg \in vDeadlineQuorum[myServerId] :$
                      $msg.txnId \notin syncedTxnIds\}$
           $]$
        $\land$ $vLSyncPoint' = [$
              $vLSyncPoint$ EXCEPT $![myServerId] = Len(m.entries)]$
   Send slow-replies to coordinators
        $\land$ $Send(\{[$
                $mtype \mapsto MSlowReply,$
                $sender \mapsto myServerId,$
                $dest \mapsto m.entries[i].txnId.coordId,$
                $gView \mapsto vGView[myServerId],$
                $lView \mapsto vLView[myServerId],$
                $txnId \mapsto m.entries[i].txnId,$
                $logId \mapsto i$
           $] : i \in (currentSyncPoint + 1) \mathrel{..} Len(m.entries)\})$
$\lor$  $\land$ $currentSyncPoint \geq Len(m.entries)$
      Noting new to sync
        $\land$ UNCHANGED $\langle networkVars, vLog, vEarlyBuffer,$
                      $vLateBuffer, vDeadlineQuorum, vLSyncPoint\rangle$
```

```
$StartLeaderFail(serverId) \triangleq$
    This leader fails
  LET
      $serversInOneShard \triangleq \{$
          $s \in Servers : s.shardId = serverId.shardId$
      $\}$
      $aliveReplicas \triangleq \{$
          $s \in serversInOneShard :$      $\land\ vServerStatus[s] = StNormal$
                                           $\land\ s \neq serverId$
      $\}$
  IN
    if the current alive replicas are less than QuorumSize
    Then no more replicas in this sharding group can fail (by assumption of consensus)
  IF $Cardinality(aliveReplicas) > QuorumSize$ THEN
      $vServerStatus' = [vServerStatus$ EXCEPT $![serverId] = StFailing]$
   ELSE     UNCHANGED $\langle vServerStatus \rangle$

$DetectLeaderFail(cmReplicaId) \triangleq$
  $\exists\ shardId \in Shards :$
    LET
        $lView \triangleq vCMGInfo[cmReplicaId].gVec[shardId]$
        $leaderId \triangleq LeaderID(lView)$
        $serverId \triangleq [$
            $replicaId \mapsto leaderId,$
            $shardId \mapsto shardId$
        $]$
    IN
    $vServerStatus[serverId] = StFailing$

$SelectProperLView(currentView, shardId) \triangleq$
  LET
      $aliveReplicaId \triangleq$ CHOOSE $replicaId \in Replicas :$
                      $vServerStatus[shardId][replicaId] = StNormal$
  IN
    Ensure 1 the new view is larger than currentView
   * (2) its corresponding leader happens to be the selected aliveReplicaId
      $(currentView \div Cardinality(Replicas) + 1) * Cardinality(Replicas) + aliveReplicaId$

$PrepareViewChange(cmReplicaId) \triangleq$
  LET
      $newGVec \triangleq [$
          $shardId \in Shards \mapsto$
              $SelectProperLView(vCMGInfo[cmReplicaId].gVec[shardId], shardId)$
      $]$
  IN
```

```
    ∧ $vCMPrepareGInfo' = [vCMPrepareGInfo$ EXCEPT $![cmReplicaId] =$
          [
              $gView \mapsto vCMGInfo[cmReplicaId].gView + 1,$
              $gVec \mapsto newGVec$
          ]
      ]
    ∧ $Send(\{[$
            $mtype \mapsto MCMPrepare,$
            $sender \mapsto cmReplicaId,$
            $dest \mapsto dstRid,$
            $cView \mapsto vCMView[cmReplicaId],$
            $gView \mapsto vCMPrepareGInfo'[cmReplicaId].gView,$
            $gVec \mapsto newGVec$

      $] : dstRid \in Replicas\})$

$LaunchViewChange(cmReplicaId) \triangleq$
    IF  ∧ $isLeader(cmReplicaId, vCMView[cmReplicaId])$
        ∧ $DetectLeaderFail(cmReplicaId)$
    THEN
        $PrepareViewChange(cmReplicaId)$
    ELSE
        UNCHANGED $\langle networkVars \rangle$


$HandleCMPrepare(m) \triangleq$
    ∧ $m.cView = vCMView[m.dest]$
    ∧ $m.gView > vCMGInfo[m.dest].gView$
    ∧ $vCMPrepareGInfo' = [vCMPrepareGInfo$ EXCEPT $![m.dest] =$
          [
              $gView \mapsto m.gView,$
              $gVec \mapsto m.gVec$
          ]
      ]
    ∧ $Send(\{[$
            $mtype \mapsto MCMPrepareReply,$
            $sender \mapsto m.dest,$
            $dest \mapsto m.src,$
            $cView \mapsto m.cView,$
            $gView \mapsto m.gView$
      $]\})$

$HandleCMPrepareReply(m) \triangleq$
    ∧ $m.cView = vCMView[m.dest]$
    ∧ $isLeader(m.dest, vCMView[m.dest])$
    ∧ $m.gView = vCMPrepareGInfo[m.dest].gView$
    ∧ $vCMPrepareReps' = [vCMPrepareReps$ EXCEPT $![m.dest] =$
          $vCMPrepareReps[m.dest] \cup \{m\}$
      ]
    ∧ LET
          $quorum \triangleq \{mm \in vCMPrepareReps[m.dest] : mm.gView = m.gView\}$
      IN
      IF $Cardinality(quorum) = QuorumSize$ THEN
            Quorum sufficient, the prepared $GInfo$ is persisted and can be safely used
          ∧ $vCMGInfo' = [vCMGInfo$ EXCEPT $![m.dest] =$
                $vCMPrepareGInfo[m.dest]$
            ]
            notify other follower $CM$, so that they can catch up with the leader
          ∧ $Send(\{[$
                  $mtype \mapsto MCMCommit,$
                  $sender \mapsto m.dest,$
                  $dest \mapsto rid,$
                  $cView \mapsto m.cView,$
                  $gView \mapsto m.gView$
            $] : rid \in \{r \in Replicas : r \neq m.dest\}\})$
            start view change, broadcast view change request to every server
          ∧ $Send(\{[$
                  $mtype \mapsto MViewChangeReq,$
                  $sender \mapsto m.dest,$
                  $dest \mapsto serverId,$
                  $gView \mapsto vCMGInfo'[m.dest].gView,$
                  $gVec \mapsto vCMGInfo'[m.dest].gVec$
            $] : serverId \in Servers\})$
       ELSE
          UNCHANGED $\langle networkVars, vCMGInfo\rangle$

$HandleCMCommit(m) \triangleq$
    ∧ $m.cView = vCMView[m.dest]$
    ∧ $\neg isLeader(m.dest, vCMView[m.dest])$
    ∧ $m.gView = vCMPrepareGInfo[m.dest].gView$
    ∧ $vCMGInfo' = [vCMGInfo$ EXCEPT $![m.dest] =$
                      $vCMPrepareGInfo[m.dest]$
      ]


$HandleViewChangeReq(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
        $myLeader \triangleq$ CHOOSE $s \in Servers :$
```

```
                    $\land\ s.replicaId = LeaderID(m.gVec[myServerId.shardId])$
                    $\land\ s.shardId = myServerId.shardId$
IN
 If the $msg$'s view is lower, ignore
$\land\ vGView[myServerId] < m.gView$
$\land$ IF $vServerStatus[myServerId] = StNormal$ THEN
        $\land\ vServerStatus' \quad = [vServerStatus$ EXCEPT $![myServerId] \quad = StViewChange]$
        $\land\ vLastNormView' = [vLastNormView$ EXCEPT $![myServerId] = vLView[myServerId]]$
    ELSE    UNCHANGED $\langle vServerStatus, vLastNormView\rangle$
$\land\ vGView' = [$
        $vGView$ EXCEPT $![myServerId] = m.vGView$
    $]$
$\land\ vGVec' = [$
        $vGVec$ EXCEPT $![myServerId] = m.gVec$
    $]$
$\land\ vLView' = [$
        $vLView$ EXCEPT $![myServerId] = m.gVec[myServerId.shardId]$
    $]$
 Clear ealry buffer,
$\land\ vEarlyBuffer' = [$
        $vEarlyBuffer$ EXCEPT $![myServerId] = \{\}$
    $]$
 Clear late buffer
$\land\ vLateBuffer' = [$
        $vLateBuffer$ EXCEPT $![myServerId] = \{\}$
    $]$
 Clear deadline quorum
$\land\ vDeadlineQuorum' = [$
        $vDeadlineQuorum$ EXCEPT $![myServerId] = \{\}$
    $]$
 Clear $vCrossShardConfirmQuorum$
$\land\ vCrossShardConfirmQuorum' = [$
        $serverId \in Servers \mapsto \{\}$
    $]$
 Send $ViewChange$ to the $myLeader$
$\land\ Send(\{[$
            $mtype \quad\ \ \mapsto MViewChange,$
            $sender \quad\ \mapsto myServerId,$
            $dest \qquad\ \mapsto myLeader,$
            $gView \quad\ \ \mapsto m.vGView,$
            $gVec \qquad \mapsto m.gVec,$
            $lView \quad\ \ \mapsto vLView'[myServerId],$
            $lastNormal \mapsto vLastNormView'[myServerId],$
            $lSyncPoint \mapsto vLSyncPoint[myServerId],$
            $entries \quad\ \mapsto vLog[myServerId],$
            $cv \qquad\qquad \mapsto vCrashVector[myServerId]$
        $]\})$
```

Define a comparison function based on the key

```
$Compare(a, b) \triangleq$
    $\lor \quad a.deadline < b.deadline$
    $\lor \quad \land a.deadline = b.deadline$
          $\land a.txnId.coordId < b.txnId.coordId$
    $\lor \quad \land a.deadline = b.deadline$
          $\land a.txnId.coordId = b.txnId.coordId$
          $\land a.txnId.rId < b.txnId.rId$


$isCrashVectorValid(m) \triangleq$
    $\land \ \forall rr \in Replicas : vCrashVector[m.dest][rr] \leq m.cv[rr]$
    $\land \ vCrashVector' = [$
            $vCrashVector$ EXCEPT $![m.dest] = [$
                $rr \in Replicas \mapsto Max(m.cv[rr], vCrashVector[m.dest][rr])$
            $]$
        $]$

$CountVotes(entry, logSets) \triangleq$
    LET
        $validCandidates \triangleq \{s \in logSets : \exists e \in s :$
                                    $\land e.deadline = entry.deadline$
                                    $\land e.txnId = entry.txnId\}$
    IN
        $Cardinality(validCandidates)$


$ReBuildLogs(vcQuorum) \triangleq$
    LET
        $refinedQuorum \triangleq \{m \in vcQuorum :$
                            $\forall msg \in vcQuorum : msg.lastNormal \leq m.lastNormal\}$
        $lSyncPoints \triangleq \{m.lSyncPoint : m \in refinedQuorum\}$
        $largestLSyncPointVC \triangleq$ CHOOSE $vc \in refinedQuorum :$
                            $\forall sp \in lSyncPoints : sp \leq vc.lSyncPoint$
        $syncedLogSeq \triangleq SubSeq(largestLSyncPointVC.entries, 1, largestLSyncPointVC.lSyncPoint)$
        $deadlineBoundary \triangleq$ IF $largestLSyncPointVC.lSyncPoint = 0$ THEN $0$
                            ELSE $syncedLogSeq[largestLSyncPointVC.lSyncPoint].deadline$
        $logSets \triangleq \{SeqToSet(m.entries) : m \in refinedQuorum\}$
        $allLogs \triangleq$ UNION $logSets$
        $allUnSyncedLogs \triangleq \{entry \in allLogs : entry.deadline > deadlineBoundary\}$
        $unSyncedLogs \triangleq \{entry \in allUnSyncedLogs :$
            $CountVotes(entry, logSets) \geq RecoveryQuorumSize\}$
        $unSyncedLogSeq \triangleq SetToSortSeq(unSyncedLogs, Compare)$
    IN
    $syncedLogSeq \circ unSyncedLogSeq$

$SelectEntriesBeyondCommitPoint(entries,\ deadline) \triangleq$
    LET
        $validLogIndices \triangleq \{$
            $i \in 1\ ..\ Len(entries) : entries[i].deadline > deadline$
        $\}$
        $startIndex \triangleq PickMin(validLogIndices)$
    IN
    $SubSeq(entries,\ startIndex,\ Len(entries))$

$HandleViewChange(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
        $serversInOneShard \triangleq \{s \in Servers : s.shardId = myServerId.shardId\}$
        $leadersInAllShard \triangleq \{$
                $s \in Servers : s.replicaId = isLeader(s.replicaId,\ m.gVec[s.shardId])$
        $\}$
    IN
    $\land$ $\lor$ $ViewGreater(m.gView,\ m.lView,\ vGView[myServerId],\ vLView[myServerId])$
      $\lor$ $\land$ $m.gView = vGView[myServerId]$
         $\land$ $m.lView = vLView[myServerId]$
         $\land$ $vServerStatus[myServerId] = StViewChange$
    $\land$ $isLeader(myServerId.replicaId,\ m.lView)$
    $\land$ $vGView' = [vGView$ EXCEPT $![myServerId] = m.gView]$
    $\land$ $vLView' = [vLView$ EXCEPT $![myServerId] = m.lView]$
    $\land$ $vGVec' = [vGVec$ EXCEPT $![myServerId] = m.gVec]$
    $\land$ $vViewChange' = [$
            $vViewChange$ EXCEPT $![myServerId] = \{$
                $vc \in vViewChange[myServerId] :$
                    $vc.lView = m.lView$
            $\} \cup \{m\}$
        $]$
    $\land$ IF $Cardinality(vViewChange'[myServerId]) = QuorumSize$ THEN
            $\land$ $vLog' = [vLog$ EXCEPT $![myServerId] = ReBuildLogs(vViewChange'[myServerId])]$
            $\land$ $vServerStatus' = [vServerStatus$ EXCEPT $![myServerId] = StCrossShardSyncing]$
            $\land$ $vLastNormView' = [vLastNormView$ EXCEPT $![myServerId] = vLView[myServerId]]$
         Even after the $log$ is recovered within one shard,
          * The newly elected leader cannot $StartView$
          * It needs to sync with other $shards'$ leaders to ensure strict serializability
            $\land$ $vViewChange' = [vViewChange$ EXCEPT $![myServerId] = \{\}]$
            $\land$ $Send(\{[$
                    $mtype \mapsto MCrossShardConfirm,$
                    $sender \mapsto myServerId,$
                    $dest$      $\mapsto dst,$
                    $lView$     $\mapsto vLView'[myServerId],$
                    $gView$     $\mapsto vGView'[myServerId],$
                    $entries$   $\mapsto SelectEntriesBeyondCommitPoint($
                                    $vLog'[myServerId],\ vPeerCommitDeadline[dst.shardId])$
                $] : dst \in leadersInAllShard\})$
        ELSE
            $\land\ vServerStatus' = [vServerStatus$ EXCEPT $![myServerId] = StViewChange]$
            $\land$ UNCHANGED $\langle networkVars,\ vLog,\ vServerStatus,\ vViewChange\rangle$

$BuildGlobalConsistentLog(serverId,\ entries) \triangleq$
    LET
        $myEntries \triangleq \{$
            $entry \in entries : \land\ serverId \in entry.shards$
                                 $\land \forall\ e \in entries :$
                                     IF $e.txnId = entry.txnId$ THEN
                                         $e.deadline \leq entry.deadline$
                                     ELSE TRUE
        $\}$
    IN
    $SetToSortSeq(myEntries,\ Compare)$

$HandleCrossShardConfirm(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
    IN
    $\land\ vServerStatus[myServerId] = StCrossShardSyncing$
    $\land\ m.gView = vGView[myServerId]$
    $\land\ m.lView = vGVec[myServerId][m.sender.shardId]$
    $\land\ vCrossShardConfirmQuorum' = [$
            $vCrossShardConfirmQuorum$ EXCEPT $![myServerId] = \{$
                $mm \in vCrossShardConfirmQuorum[myServerId] :$
                        $\land\ mm.gView = vGView[myServerId]$
                        $\land\ mm.lView = vGVec[myServerId][mm.sender.shardId]$
            $\} \cup \{m\}$
        $]$
    $\land$ IF $Cardinality(vCrossShardConfirmQuorum'[myServerId]) = Cardinality(Shards)$
        THEN
            Check $Txns'$ Deadlines to ensure strict serializability is not violated
            In implementation, we should not pass all txns, instead, we should only pass dealines and $txn$ indices
            As an optimization, we should also use checkpoint in implementation
            Here for conciseness, we pass all $log$ entries
          LET
              $allLogs \triangleq$ UNION $\{SeqToSet(mm.entries) :$
                                  $mm \in vCrossShardConfirmQuorum'[myServerId]\}$
            $serversInOneShard \triangleq \{s \in Servers : s.shardId = myServerId.shardId\}$
        IN
         $\land$ $vLog' = [$
                 $vLog$ EXCEPT $![myServerId] =$
                     $BuildGlobalConsistentLog(m.sender, allLogs)$
             $]$
         $\land$ $Send(\{[$
                 $mtype \mapsto MStartView,$
                 $sender \mapsto myServerId,$
                 $dest \mapsto dst,$
                 $lView \mapsto vLView[myServerId],$
                 $gView \mapsto vGView[myServerId],$
                 $gVec \mapsto vGVec[myServerId],$
                 $entries \mapsto vLog'[myServerId],$
                 $cv \mapsto vCrashVector[myServerId]$
             $] : dst \in serversInOneShard\})$
     ELSE
        UNCHANGED $\langle vLog, networkVars\rangle$

$HandleStartView(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
    IN
    $\land$ $\lor$ $ViewGreater(m.gView, m.lView, vGView[myServerId], vLView[myServerId])$
       $\lor$ $\land$ $m.gView = vGView[myServerId]$
          $\land$ $m.lView = vLView[myServerId]$
          $\land$ $\lor$ $vServerStatus[myServerId] = StViewChange$
             $\lor$ $vServerStatus[myServerId] = StRecovering$
    $\land$ $vGView' = [vGView$ EXCEPT $![myServerId] = m.gView]$
    $\land$ $vLView' = [vLView$ EXCEPT $![myServerId] = m.gLView]$
    $\land$ $vGVec' = [vGVec$ EXCEPT $![myServerId] = m.vGVec]$
    $\land$ $vServerStatus' = [vServerStatus$ EXCEPT $![myServerId] = StNormal]$
    $\land$ $vLog' = [vLog$ EXCEPT $![myServerId] = m.entries]$
    $\land$ $vEarlyBuffer' = [vEarlyBuffer$ EXCEPT $![myServerId] = \{\}]$
    $\land$ $vLateBuffer' = [vLateBuffer$ EXCEPT $![myServerId] = \{\}]$
    $\land$ $vDeadlineQuorum' = [vDeadlineQuorum$ EXCEPT $![myServerId] = \{\}]$
    $\land$ $vCrossShardConfirmQuorum' = [$
            $vCrossShardConfirmQuorum$ EXCEPT $![myServerId] = \{\}$
        $]$
    $\land$ $vLSyncPoint' = [vLSyncPoint$ EXCEPT $![myServerId] = Len(vLog'[myServerId])]$
    $\land$ $vLastNormView' = [vLastNormView$ EXCEPT $![myServerId] = m.lView]$
    $\land$ $vViewChange' = [vViewChange$ EXCEPT $![myServerId] = \{\}]$
    $\land$ $vLSyncQuorum' = [vLSyncQuorum$ EXCEPT $![myServerId] = \{\}]$
    $\land$ $vCrashVectorReps' = [vCrashVectorReps$ EXCEPT $![myServerId] = \{\}]$
    $\land$ $vRecoveryReps' = [vRecoveryReps$ EXCEPT $![myServerId] = \{\}]$
```

```
$ResetServerState(serverId) \triangleq$
    $\land$ $vLog' = [vLog$ EXCEPT $![serverId] = \langle\rangle]$
    $\land$ $vEarlyBuffer' = [vEarlyBuffer$ EXCEPT $![serverId] = \{\}]$
    $\land$ $vLateBuffer' = [vLateBuffer$ EXCEPT $![serverId] = \{\}]$
    $\land$ $vDeadlineQuorum' = [vDeadlineQuorum$ EXCEPT $![serverId] = \{\}]$
    $\land$ $vCrossShardConfirmQuorum' = [$
            $vCrossShardConfirmQuorum$ EXCEPT $![serverId] = \{\}$
        $]$
    $\land$ $vGView' = [vGView$ EXCEPT $![serverId] = 0]$
    $\land$ $vGVec' = [vGVec$ EXCEPT $![serverId] = [s \in Shards \mapsto 0]]$
    $\land$ $vLView' = [vLView$ EXCEPT $![serverId] = 0]$
    $\land$ $vLastNormView' = [vLastNormView$ EXCEPT $![serverId] = 0]$
    $\land$ $vViewChange' = [vViewChange$ EXCEPT $![serverId] = \{\}]$
    $\land$ $vLSyncPoint' = [vLSyncPoint$ EXCEPT $![serverId] = 0]$
    $\land$ $vLCommitPoint' = [vLCommitPoint$ EXCEPT $![serverId] = 0]$
    $\land$ $vPeerCommitDeadline' = [vPeerCommitDeadline$ EXCEPT $![serverId] = 0]$
    $\land$ $vLSyncQuorum' = [vLSyncQuorum$ EXCEPT $![serverId] = \{\}]$
    $\land$ $vCrashVector' = [vCrashVector$ EXCEPT $![serverId] = [$
            $rr \in Replicas \mapsto 0$
        $]]$
    $\land$ $vCrashVectorReps' = [vCrashVectorReps$ EXCEPT $![serverId] = \{\}]$
    $\land$ $vRecoveryReps' = [vRecoveryReps$ EXCEPT $![serverId] = \{\}]$
    $\land$ $vServerProcessed' = [vServerProcessed$ EXCEPT $![serverId] = \{\}]$

$StartServerRecovery(serverId) \triangleq$
    LET
        $serversInOneShard \triangleq \{$
            $s \in Servers : s.shardId = serverId.shardId$
        $\}$
        $nonce \triangleq vUUIDCounter[serverId] + 1$
    IN
    $\land$ $vServerStatus' = [vServerStatus$ EXCEPT $![serverId] = StRecovering]$
    $\land$ $vUUIDCounter' = [vUUIDCounter$ EXCEPT $![serverId] = vUUIDCounter[serverId] + 1]$
    $\land$ $ResetServerState(serverId)$
    $\land$ $Send(\{[$
             $mtype$      $\mapsto MCrashVectorReq,$
             $sender$     $\mapsto serverId,$
             $dest$       $\mapsto dst,$
             $nonce$      $\mapsto nonce$
        $] : dst \in serversInOneShard\})$

$HandleCrashVectorReq(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
    IN
    ∧ vServerStatus[myServerId] = StNormal
    ∧ Send({[
            mtype      ↦ MCrashVectorRep,
            sender     ↦ myServerId,
            dest       ↦ m.sender,
            nonce      ↦ m.nonce,
            cv         ↦ vCrashVector[myServerId]
        ]})

AggregateCV(serverId) ≜
    LET
        cvQuorum ≜ {m.cv : m ∈ vCrashVectorReps[serverId]}
        cvValQuorum ≜ [rr ∈ Replicas ↦ {cv[rr] : cv ∈ cvQuorum} ]
    IN
        [rr ∈ Replicas ↦ PickMax(cvValQuorum[rr])]

HandleCrashVectorRep(m) ≜
    LET
        myServerId ≜ m.dest
        serversInOneShard ≜ {s ∈ Servers : s.shardId = myServerId.shardId}
    IN
    ∧ vServerStatus[myServerId] = StRecovering
    ∧ vUUIDCounter[myServerId] = m.nonce
    ∧ vCrashVectorReps' = [
          vCrashVectorReps EXCEPT ![myServerId] = vCrashVectorReps ∪ {m}
      ]
    ∧ IF Cardinality(vCrashVectorReps'[myServerId]) = QuorumSize THEN
          LET
              acv ≜ AggregateCV(myServerId)
              myCV ≜ [acv EXCEPT ![myServerId] = acv[myServerId] + 1]
          IN
          ∧ vCrashVector' = [
                vCrashVector EXCEPT ![myServerId] = myCV
            ]
          ∧ Send({[
                  mtype      ↦ MRecoveryReq,
                  sender     ↦ myServerId,
                  dest       ↦ dst,
                  cv         ↦ myCV
              ] : dst ∈ serversInOneShard})
      ELSE    UNCHANGED ⟨networkVars, vCrashVector⟩


HandleRecoveryReq(m) ≜
```

```
    LET
        $myServerId \triangleq m.dest$
    IN
    $\land$ $vServerStatus[myServerId] = StNormal$
    $\land$ $Send(\{[$
            $mtype \mapsto MRecoveryRep,$
            $sender \mapsto myServerId,$
            $dest \mapsto m.sender,$
            $gView \mapsto vGView[myServerId],$
            $lView \mapsto vLView[myServerId],$
            $cv \mapsto vCrashVector'[myServerId]$
        $]\})$

$HandleRecoveryRep(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
    IN
    $\land$ $vServerStatus[myServerId] = StRecovering$
    $\land$ $vRecoveryReps' = [$
            $vRecoveryReps$ EXCEPT $![myServerId]$
                $= vRecoveryReps[myServerId] \cup \{m\}$
        $]$
    $\land$ IF $Cardinality(vRecoveryReps[myServerId]) = QuorumSize$ THEN
            LET
                $lViewQuorum \triangleq \{mm.lView : mm \in vRecoveryReps[myServerId]\}$
                $gViewQuorum \triangleq \{mm.gView : mm \in vRecoveryReps[myServerId]\}$
            IN
            $\land$ $vLView' = [vLView$ EXCEPT $![myServerId] = PickMax(lViewQuorum)]$
            $\land$ $vGView' = [vLView$ EXCEPT $![myServerId] = PickMax(gViewQuorum)]$
            $\land$ $Send(\{[$
                    $mtype \mapsto MStartViewReq,$
                    $sender \mapsto myServerId,$
                    $dest \mapsto [$
                            $replicaId \mapsto LeaderID(vLView[myServerId]),$
                            $shardId \mapsto myServerId.shardId$
                        $],$
                    $lView \mapsto vLView'[myServerId],$
                    $cv \mapsto vCrashVector'[myServerId]$
                $]\})$
        ELSE UNCHANGED $\langle networkVars, vLView, vGView \rangle$

$HandleStartViewReq(m) \triangleq$
    LET
```

```
        $myServerId \triangleq m.dest$
    IN
    $\land$ $vServerStatus[myServerId] = StNormal$
    $\land$ $vLView[myServerId] = m.lView$
    $\land$ $isLeader(myServerId.replicaId,\ vLView[myServerId])$
    $\land$ $Send(\{[$
            $mtype \mapsto MStartView,$
            $sender \mapsto myServerId,$
            $dest \mapsto m.sender,$
            $lView \mapsto vLView[myServerId],$
            $gView \mapsto vGView[myServerId],$
            $gVec \mapsto vGVec[myServerId],$
            $entries \mapsto vLog[myServerId],$
            $cv \mapsto vCrashVector[myServerId]$
        $]\})$


$StartLocalSync(serverId) \triangleq$
    LET
        $leaderServerId \triangleq [$
            $replicaId \mapsto LeaderID(vLView[serverId]),$
            $shardId \mapsto serverId.shardId$
        $]$
    IN
    $\land$ $vServerStatus[serverId] = StNormal$
    $\land$ $Send(\{[$
            $mtype \mapsto MLocalSyncStatus,$
            $sender \mapsto serverId,$
            $dest \mapsto leaderServerId,$
            $lView \mapsto vLView[serverId],$
            $lSyncPoint \mapsto vLSyncPoint[serverId],$
            $cv \mapsto vCrashVector[serverId]$
        $]\})$

$HandleLocalSyncStatus(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
        $lSyncQuorum \triangleq vLSyncQuorum[myServerId]$
    IN
    $\land$ $vServerStatus[myServerId] = StNormal$
    $\land$ $vLView[myServerId] = m.lView$
    $\land$ $isLeader(myServerId.replicaId,\ vLView[myServerId])$
    $\land$ $\forall\, mm \in lSyncQuorum :$
        $\lor$ $mm.sender \neq m.sender$
```

```
            ∨  $mm.lSyncPoint < m.lSyncPoint$
      ∧  $vLSyncQuorum' = [$
               $vLSyncQuorum$ EXCEPT $![mySeverId] =$
                    $\{mm \in lSyncQuorum : mm.sender \neq m.sender\} \cup \{m\}$
         $]$
      ∧  IF  $Cardinality(vLSyncQuorum'[myServerId]) \geq QuorumSize$ THEN
               LET
                    $candidateQuorum \triangleq \{$
                         $R \in$ SUBSET $(vLSyncQuorum'[myServerId]) :$
                              $Cardinality(R) = QuorumSize$
                    $\}$
                    $quorumSyncPoints \triangleq \{$
                         $\{x.lSyncPoint : x \in R\} : R \in candidateQuorum$
                    $\}$
                    $validCommitPoints \triangleq \{PickMax(Q) : Q \in quorumSyncPoints\}$
                    $maxCommitPoint \triangleq PickMax(validCommitPoints)$
               IN
                ∧  $vLCommitPoint' = [vLCommitPoint$ EXCEPT $![myServerId] = maxCommitPoint]$
                ∧  $Send(\{[$
                         $mtype$        $\mapsto MLocalCommit,$
                         $sender$       $\mapsto myServerId,$
                         $dest$         $\mapsto m.sender,$
                         $lView$        $\mapsto vLView[myServerId],$
                         $lCommitPoint$ $\mapsto vLCommitPoint'[myServerId],$
                         $cv$           $\mapsto vCrashVector'[myServerId]$
                    $]\})$
           ELSE     UNCHANGED $\langle vLCommitPoint, networkVars\rangle$


$HandleLocalCommit(m) \triangleq$
   LET
      $myServerId \triangleq m.dest$
   IN
   ∧  $vServerStatus[myServerId] = StNormal$
   ∧  $vLView[myServerId] = m.lView$
   ∧  $\neg isLeader(myServerId.replicaId, vLView[myServerId])$
    Make sure the *syncPoint* is large enough before updating *CommitPoint*
   ∧  IF   ∧  $vLSyncPoint[myServerId] \geq m.lCommitPoint$
           ∧  $vLCommitPoint[myServerId] < m.lCommitPoint$
       THEN
          $vLCommitPoint' = [$
               $vLCommitPoint$ EXCEPT $![myServerId] = m.lCommitPoint$
          $]$
       ELSE  UNCHANGED $\langle vLCommitPoint\rangle$
```

```
$BroadcastCommitStatusToPeers(serverId) \triangleq$
    LET
        $serversInOneReplica \triangleq \{s \in Servers : s.replicaId = serverId.replicaId\}$
        $commitPoint \triangleq vLCommitPoint[serverId]$
        $commitDeadline \triangleq$
            IF $commitPoint = 0$ THEN $0$
             ELSE $vLog[commitPoint].deadline$
    IN
    $\land$ $vServerStatus[serverId] = StNormal$
    $\land$ $Send(\{[$
            $mtype$        $\mapsto MPeerShardCommitStatus,$
            $sender$       $\mapsto serverId,$
            $dest$         $\mapsto dst,$
            $gView$        $\mapsto vGView[serverId],$
            $lView$        $\mapsto vLView[serverId],$
            $deadline$     $\mapsto commitDeadline$
        $] : dst \in serversInOneReplica\})$


$HandlePeerShardCommitStatus(m) \triangleq$
    LET
        $myServerId \triangleq m.dest$
    IN
    $\land$ $vServerStatus[myServerId] = StNormal$
    $\land$ $vGView[myServerId] = m.gView$
    $\land$ $vGVec[myServerId][m.sender.shardId] = m.lView$
    $\land$ IF $m.deadline > vPeerCommitDeadline[myServerId][m.sender.shardId]$ THEN
            $\land$ $vPeerCommitDeadline[myServerId]' = [$
                  $vPeerCommitDeadline[myServerId]$
                      EXCEPT $![m.sender.shardId] = m.deadline$
                $]$
        ELSE UNCHANGED $\langle vPeerCommitDeadline \rangle$


$isCommitting(txn, deadlineQ) \triangleq$
    LET $quorum \triangleq \{msg \in deadlineQ : msg.entry.txnId = txn.txnId\}$
    IN  $Cardinality(quorum) = Cardinality(txn.shards)$


$ReleaseSeqeuncer(serverId, currentTime) \triangleq$
    LET
        $serversInOneShard \triangleq \{s \in Servers : s.shardId = serverId.shardId\}$
        $expireTxns \triangleq$
            $\{msg \in vEarlyBuffer[serverId] :$
                $\land msg.deadline \leq currentTime\}$
    $sortedTxnList \triangleq SetToSortSeq(expireTxns, Compare)$
    $committingStatus \triangleq$
        $[i \in 1 \,..\, Len(sortedTxnList)$
            $\mapsto isCommitting(sortedTxnList[i], vDeadlineQuorum[serverId])$
        $]$
    $canReleaseTxnIndices \triangleq \{$
        $i \in 1 \,..\, Len(sortedTxnList) :$
            $\forall j \in 1 \,..\, i : committingStatus[j] = \text{TRUE}\}$
IN
IF  $Cardinality(canReleaseTxnIndices) = 0$   Nothing to release
 THEN    UNCHANGED $\langle networkVars,$
              $vLog, vEarlyBuffer, vLateBuffer, vDeadlineQuorum\rangle$
 ELSE
    LET
        $releaseUpTo \triangleq$ CHOOSE $i \in canReleaseTxnIndices :$
                        $\forall j \in canReleaseTxnIndices : j \leq i$
        $releaseSeq \triangleq SubSeq(sortedTxnList, 1, releaseUpTo)$
        $releaseTxns \triangleq \{releaseSeq[i] : i \in 1 \,..\, Len(releaseSeq)\}$
    IN
    $\land vEarlyBuffer' = [$
        $vEarlyBuffer$ EXCEPT $![serverId]$
            $= vEarlyBuffer[serverId] \setminus releaseTxns]$
    $\land vDeadlineQuorum' = [$
        $vDeadlineQuorum$ EXCEPT $![serverId]$
            $= \{msg \in vDeadlineQuorum[serverId] :$
                $\forall txn \in releaseTxns : txn.txnId \neq msg.entry.txnId\}$
        $]$
     Append to $log$
    $\land vLog' = [vLog$ EXCEPT $![serverId] = vLog[serverId] \circ releaseSeq]$
    $\land$ IF $isLeader(serverId.replicaId, vLView[serverId])$ THEN
            $\land vLSyncPoint' = [vLSyncPoint$ EXCEPT $![serverId] = Len(vLog'[serverId])]$
        ELSE    UNCHANGED $\langle vLSyncPoint\rangle$
     Send fast-replies to coordinators
    $\land Send(\{[$
        $mtype \mapsto MFastReply,$
        $sender \mapsto serverId,$
        $dest \mapsto sortedTxnList[i].txnId.coordId,$
        $gView \mapsto vGView[serverId],$
        $lView \mapsto vLView[serverId],$
        $txnId \mapsto sortedTxnList[i].txnId,$
        $hash \mapsto [$
                $log \mapsto vLog'[serverId],$
                $cv \mapsto vCrashVector$
            $],$
        $logId \mapsto i$
        ] : $i \in (1 + Len(vLog[serverId]))\ ..\ Len(vLog'[serverId])\})$
      Send InterReplicaSync to the other servers in the same sharding group
      In real implementation, we send the log indices incrementally (i.e., consider it as an optimization)
      Here for clarity and simplicity, we always send the whole log list
    $\land$ $Send(\{[$
            $mtype \mapsto MInterReplicaSync,$
            $lView \mapsto vLView[serverId],$
            $sender \mapsto serverId,$
            $dest \mapsto dstServerId,$
            $entries \mapsto vLog'[serverId]$
        $] : dstServerId \in serversInOneShard\})$


$ServerClockMove(serverId) \triangleq$
    IF $vServerClock[serverId] \geq MaxTime$ THEN
        UNCHANGED $\langle networkVars, serverStateVars \rangle$
    ELSE
        $\land$ $vServerClock' = [$
              $vServerClock$ EXCEPT $![serverId] = vServerClock[serverId] + 1]$
        $\land$ IF $vServerStatus[serverId] = StNormal$ THEN
              $\land$ $ReleaseSeqeuncer(serverId, vServerClock[serverId] + 1)$
            ELSE
              UNCHANGED $\langle networkVars, vLog, vEarlyBuffer,$
                  $vLateBuffer, vDeadlineQuorum \rangle$
        $\land$ UNCHANGED $\langle vCrossShardConfirmQuorum,$
              $vServerStatus, vGView, vGVec, vLView, vLastNormView,$
              $vViewChange, vLSyncPoint, vLCommitPoint,$
              $vPeerCommitDeadline, vLSyncQuorum,$
              $vUUIDCounter, vCrashVector, vCrashVectorReps,$
              $vRecoveryReps, vServerProcessed \rangle$


$CoordClockMove(coordId) \triangleq$
    $\lor$ $\land$ $vCoordClock[coordId] \geq MaxTime$
       $\land$ UNCHANGED $\langle vCoordClock \rangle$
    $\lor$ $\land$ $vCoordClock[coordId] < MaxTime$
       $\land$ $vCoordClock' = [$
           $vCoordClock$ EXCEPT $![coordId] = vCoordClock[coordId] + 1]$

$Init \triangleq$
    $\land$ $InitNetworkState$
    $\land$ $InitServerState$
    $\land$ $InitCoordState$
    $\land$ $InitConfigManagerState$
    $\land$ $ActionName = \langle$ "Init" $\rangle$
```

```
$Next \triangleq$
    $\lor$  $\land$ $ActionName' = \langle$"Next"$\rangle$
        $\land$ UNCHANGED $\langle networkVars, serverStateVars,$
                          $coordStateVars, configManagerStateVars\rangle$
    $\lor$  $\exists c \in Coords$ :
        $\land$ $Cardinality(vCoordTxns[c]) < MaxReqNum$
        $\land$ $CoordSubmitTxn(c)$
        $\land$ UNCHANGED $\langle serverStateVars, configManagerStateVars,$
                    $vCoordProcessed\rangle$
        $\land$ $ActionName' = \langle$"CoordSubmitTxn"$\rangle$

    $\lor$ $\exists m \in messages$ :
        $\land$ $m.mtype = MTxn$
        $\land$ $vServerStatus[m.dest] = StNormal$
        $\land$ $m \notin vServerProcessed[m.dest]$
        $\land$ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
           $vServerProcessed[m.dest] \cup \{m\}]$
        $\land$ $HandleTxn(m)$
        $\land$ UNCHANGED  $\langle coordStateVars, configManagerStateVars,$
           $vLog, vDeadlineQuorum, vCrossShardConfirmQuorum,$
           $vServerStatus, vGView, vGVec,$
           $vLView, vServerClock, vLastNormView,$
           $vViewChange, vLSyncPoint, vLCommitPoint,$
           $vPeerCommitDeadline, vLSyncQuorum,$
           $vUUIDCounter, vCrashVector,$
           $vCrashVectorReps, vRecoveryReps\rangle$
        $\land$ $ActionName' = \langle$"HandleTxn"$\rangle$

    $\lor$ $\exists m \in messages$ :
        $\land$ $m.mtype = MDeadlineNotification$
        $\land$ $vServerStatus[m.dest] = StNormal$
        $\land$ $m \notin vServerProcessed[m.dest]$
        $\land$ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
           $vServerProcessed[m.dest] \cup \{m\}]$
        $\land$ $HandleDeadlineNotification(m)$
        $\land$ UNCHANGED  $\langle networkVars, coordStateVars, configManagerStateVars,$
              $vLog, vCrossShardConfirmQuorum, vLateBuffer,$
              $vServerStatus, vGView, vGVec,$
              $vLView, vServerClock, vLastNormView,$
              $vViewChange, vLSyncPoint, vLCommitPoint,$
              $vPeerCommitDeadline, vLSyncQuorum,$
              $vUUIDCounter, vCrashVector, vCrashVectorReps,$
              $vRecoveryReps\rangle$
        $\land$ $ActionName' = \langle$"HandleDeadlineNotification"$\rangle$

    $\lor$ $\exists m \in messages$ :
```

```
    $\land\ m.mtype = MInterReplicaSync$
    $\land\ vServerStatus[m.dest] = StNormal$
    $\land\ m \notin vServerProcessed[m.dest]$
    $\land\ vServerProcessed' = [vServerProcessed\ \text{EXCEPT}\ ![m.dest] =$
        $vServerProcessed[m.dest] \cup \{m\}]$
    $\land\ HandleInterReplicaSync(m)$
    $\land\ \text{UNCHANGED}\ \langle coordStateVars,\ configManagerStateVars,$
            $vLog,\ vCrossShardConfirmQuorum,\ vLateBuffer,$
            $vServerStatus,\ vGView,\ vGVec,$
            $vLView,\ vServerClock,\ vLastNormView,$
            $vViewChange,\ vLCommitPoint,\ vPeerCommitDeadline,$
            $vLSyncQuorum,\ vUUIDCounter,\ vCrashVector,$
            $vCrashVectorReps,\ vRecoveryReps\rangle$
    $\land\ ActionName' = \langle$ "HandleInterReplicaSync" $\rangle$

 Some $Leader(s)$ fail
$\lor\ \exists\, serverId \in Servers :$
    $\land\ vLView[serverId] < MaxViews$
    $\land\ isLeader(serverId.replicaId,\ vLView[serverId])$
    $\land\ StartLeaderFail(serverId)$
    $\land\ \text{UNCHANGED}\ \langle networkVars,\ coordStateVars,\ configManagerStateVars,$
        $vLog,\ vEarlyBuffer,\ vLateBuffer,$
        $vDeadlineQuorum,\ vCrossShardConfirmQuorum,\ vGView,\ vGVec,$
        $vLView,\ vServerClock,\ vLastNormView,$
        $vViewChange,\ vLSyncPoint,\ vLCommitPoint,$
        $vPeerCommitDeadline,\ vLSyncQuorum,$
        $vUUIDCounter,\ vCrashVector,\ vCrashVectorReps,$
        $vRecoveryReps,\ vServerProcessed\rangle$
    $\land\ ActionName' = \langle$ "StartLeaderFail" $\rangle$

 Config Manager notices some $leader(s)$ fail and launch view change
$\lor\ \ \exists\, cmReplicaId \in Replicas :$
    $\land\ \ LaunchViewChange(cmReplicaId)$
    $\land\ \ \text{UNCHANGED}\ \ \langle coordStateVars,\ serverStateVars,\ configManagerStateVars\rangle$
    $\land\ \ ActionName' = \langle$ "LaunchViewChange" $\rangle$

$\lor\ \ \exists\, m \in messages :$
    $\land\ \ m.mtype = MCMPrepare$
    $\land\ \ m \notin vCMProcessed[m.dest]$
    $\land\ \ vCMProcessed' = [vCMProcessed\ \text{EXCEPT}\ ![m.dest] =$
            $vCMProcessed[m.dest] \cup \{m\}]$
    $\land\ \ HandleCMPrepare(m)$
    $\land\ \ \text{UNCHANGED}\ \ \langle coordStateVars,\ serverStateVars\rangle$
    $\land\ ActionName' = \langle$ "HandleCMPrepare" $\rangle$
```

```
∨ $\exists\, m \in messages$ :
    ∧ $m.mtype = MCMPrepareReply$
    ∧ $m \notin vCMProcessed[m.dest]$
    ∧ $vCMProcessed' = [vCMProcessed$ EXCEPT $![m.dest] =$
            $vCMProcessed[m.dest] \cup \{m\}]$
    ∧ $HandleCMPrepareReply(m)$
    ∧ UNCHANGED $\langle coordStateVars, serverStateVars,$
                        $vCMStatus, vCMView, vCMPrepareGInfo\rangle$
    ∧ $ActionName' = \langle$"HandleCMPrepareReply"$\rangle$

∨ $\exists\, m \in messages$ :
    ∧ $m.mtype = MCMCommit$
    ∧ $m \notin vCMProcessed[m.dest]$
    ∧ $vCMProcessed' = [vCMProcessed$ EXCEPT $![m.dest] =$
            $vCMProcessed[m.dest] \cup \{m\}]$
    ∧ $HandleCMCommit(m)$
    ∧ UNCHANGED $\langle networkVars, coordStateVars, serverStateVars,$
                        $vCMStatus, vCMView, vCMPrepareGInfo, vCMPrepareReps\rangle$
    ∧ $ActionName' = \langle$"HandleCMCommit"$\rangle$

∨ $\exists\, m \in messages$ :
    ∧ $m.mtype = MViewChangeReq$
    ∧ $m \notin vServerProcessed[m.dest]$
    ∧ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
            $vServerProcessed[m.dest] \cup \{m\}]$
    ∧ $vServerStatus[m.dest] \neq StFailing$
    ∧ $HandleViewChangeReq(m)$
    ∧ UNCHANGED $\langle coordStateVars, configManagerStateVars,$
            $vLog, vServerClock, vViewChange, vLSyncPoint,$
            $vLCommitPoint, vLSyncQuorum, vPeerCommitDeadline,$
            $vUUIDCounter, vCrashVector, vCrashVectorReps, vRecoveryReps\rangle$
    ∧ $ActionName' = \langle$"HandleViewChangeReq"$\rangle$

∨ $\exists\, m \in messages$ :
    ∧ $m.mtype = MViewChange$
    ∧ $isCrashVectorValid(m)$
    ∧ $m \notin vServerProcessed[m.dest]$
    ∧ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
            $vServerProcessed[m.dest] \cup \{m\}]$
    ∧ $vServerStatus[m.dest] \neq StFailing$
    ∧ $HandleViewChange(m)$
    ∧ UNCHANGED $\langle coordStateVars, configManagerStateVars,$
            $vGVec, vServerClock, vLSyncPoint, vLastNormView,$
            $vLCommitPoint, vPeerCommitDeadline, vLSyncQuorum,$
            vUUIDCounter, vCrashVector, vCrashVectorReps,
            vRecoveryReps⟩
    ∧ ActionName′ = ⟨"HandleViewChange"⟩

∨  ∃ m ∈ messages :
    ∧  m.mtype = MCrossShardConfirm
    ∧  m ∉ vServerProcessed[m.dest]
    ∧  vServerProcessed′ = [vServerProcessed EXCEPT ![m.dest] =
            vServerProcessed[m.dest] ∪ {m}]
    ∧  vServerStatus[m.dest] = StViewChange
    ∧  HandleCrossShardConfirm(m)
    ∧  UNCHANGED  ⟨coordStateVars, configManagerStateVars,
            vGVec, vServerClock, vLSyncPoint, vLastNormView,
            vLCommitPoint, vPeerCommitDeadline, vLSyncQuorum,
            vUUIDCounter, vCrashVector, vCrashVectorReps, vRecoveryReps⟩
    ∧ ActionName′ = ⟨"HandleCrossShardConfirm"⟩

∨  ∃ m ∈ messages :
    ∧  m.mtype = MStartView
    ∧  isCrashVectorValid(m)
    ∧  m ∉ vServerProcessed[m.dest]
    ∧  vServerProcessed′ = [vServerProcessed EXCEPT ![m.dest] =
            vServerProcessed[m.dest] ∪ {m}]
    ∧  HandleStartView(m)
    ∧  UNCHANGED  ⟨coordStateVars, configManagerStateVars,
                vServerClock, vLCommitPoint, vPeerCommitDeadline,
                vUUIDCounter, vCrashVector⟩
    ∧ ActionName′ = ⟨"HandleStartView"⟩

 Failed server rejoin
∨ ∃ serverId ∈ Servers :
    ∧ vServerStatus[serverId] = StFailing
    ∧ vServerStatus′ = [vServerStatus EXCEPT ![serverId] = StRecovering]
    ∧ ResetServerState(serverId)
    ∧ StartServerRecovery(serverId)
    ∧ UNCHANGED ⟨networkVars, coordStateVars, coordStateVars⟩
    ∧ ActionName′ = ⟨"StartReplicaRecovery"⟩

∨  ∃ m ∈ messages :
    ∧  m.mtype = MCrashVectorReq
    ∧  m ∉ vServerProcessed[m.dest]
    ∧  vServerProcessed′ = [vServerProcessed EXCEPT ![m.dest] =
            vServerProcessed[m.dest] ∪ {m}]
    ∧  HandleCrashVectorReq(m)
    ∧  UNCHANGED  ⟨coordStateVars, configManagerStateVars,
            vLog, vEarlyBuffer, vLateBuffer, vDeadlineQuorum,
            vCrossShardConfirmQuorum, vServerStatus,
            vGView, vGVec, vLView, vServerClock, vLastNormView,
            vViewChange, vLSyncPoint, vLCommitPoint,
            vPeerCommitDeadline, vLSyncQuorum, vUUIDCounter,
            vCrashVector, vCrashVectorReps, vRecoveryReps⟩
    ∧ ActionName' = ⟨"HandleCrashVectorReq"⟩

∨  ∃ m ∈ messages :
    ∧  m.mtype = MCrashVectorRep
    ∧  m ∉ vServerProcessed[m.dest]
    ∧  vServerProcessed' = [vServerProcessed EXCEPT ![m.dest] =
            vServerProcessed[m.dest] ∪ {m}]
    ∧  HandleCrashVectorRep(m)
    ∧  UNCHANGED  ⟨coordStateVars, configManagerStateVars,
            vLog, vEarlyBuffer, vLateBuffer,
            vDeadlineQuorum, vCrossShardConfirmQuorum, vServerStatus,
            vGView, vGVec, vLView, vServerClock, vLastNormView,
            vViewChange, vLSyncPoint, vLCommitPoint,
            vPeerCommitDeadline, vLSyncQuorum,
            vUUIDCounter, vCrashVectorReps, vRecoveryReps⟩
    ∧ ActionName' = ⟨"HandleCrashVectorRep"⟩

∨  ∃ m ∈ messages :
    ∧  m.mtype = MRecoveryReq
    ∧  m ∉ vServerProcessed[m.dest]
    ∧  vServerProcessed' = [vServerProcessed EXCEPT ![m.dest] =
            vServerProcessed[m.dest] ∪ {m}]
    ∧  isCrashVectorValid(m)
    ∧  HandleRecoveryReq(m)
    ∧  UNCHANGED  ⟨coordStateVars, configManagerStateVars,
            vLog, vEarlyBuffer, vLateBuffer,
            vDeadlineQuorum, vCrossShardConfirmQuorum, vServerStatus,
            vGView, vGVec, vLView, vServerClock, vLastNormView,
            vViewChange, vLSyncPoint, vLCommitPoint,
            vPeerCommitDeadline, vLSyncQuorum,
            vUUIDCounter, vCrashVectorReps, vRecoveryReps⟩
    ∧ ActionName' = ⟨"HandleRecoveryReq"⟩

∨  ∃ m ∈ messages :
    ∧  m.mtype = MRecoveryRep
    ∧  m ∉ vServerProcessed[m.dest]
    ∧  vServerProcessed' = [vServerProcessed EXCEPT ![m.dest] =
            vServerProcessed[m.dest] ∪ {m}]
    ∧  isCrashVectorValid(m)
    ∧  HandleRecoveryRep(m)
    ∧  UNCHANGED  ⟨coordStateVars, configManagerStateVars,
            $vLog$, $vEarlyBuffer$, $vLateBuffer$,
            $vDeadlineQuorum$, $vCrossShardConfirmQuorum$, $vServerStatus$,
            $vGVec$, $vServerClock$, $vLastNormView$,
            $vViewChange$, $vLSyncPoint$, $vLCommitPoint$,
            $vPeerCommitDeadline$, $vLSyncQuorum$,
            $vUUIDCounter$, $vCrashVectorReps$, $vRecoveryReps\rangle$
    $\land$ $ActionName' = \langle$ "HandleRecoveryRep" $\rangle$

$\lor$ $\exists\, m \in messages$ :
    $\land$ $m.mtype = MStartViewReq$
    $\land$ $m \notin vServerProcessed[m.dest]$
    $\land$ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
            $vServerProcessed[m.dest] \cup \{m\}]$
    $\land$ $isCrashVectorValid(m)$
    $\land$ $HandleStartViewReq(m)$
    $\land$ UNCHANGED $\langle coordStateVars$, $configManagerStateVars$,
            $vLog$, $vEarlyBuffer$, $vLateBuffer$, $vDeadlineQuorum$,
            $vCrossShardConfirmQuorum$, $vServerStatus$,
            $vGView$, $vGVec$, $vLView$, $vServerClock$,
            $vLastNormView$, $vViewChange$, $vLSyncPoint$,
            $vLCommitPoint$, $vPeerCommitDeadline$, $vLSyncQuorum$,
            $vUUIDCounter$, $vCrashVector$,
            $vCrashVectorReps$, $vRecoveryReps\rangle$
    $\land$ $ActionName' = \langle$ "HandleStartViewReq" $\rangle$

 Periodic Sync
$\lor$ $\exists\, serverId \in Servers$ :
    $\land$ $StartLocalSync(serverId)$
    $\land$ UNCHANGED $\langle coordStateVars$,
            $serverStateVars$, $configManagerStateVars\rangle$
    $\land$ $ActionName' = \langle$ "StartLocalSync" $\rangle$

$\lor$ $\exists\, m \in messages$ :
    $\land$ $m.mtype = MLocalSyncStatus$
    $\land$ $m \notin vServerProcessed[m.dest]$
    $\land$ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
            $vServerProcessed[m.dest] \cup \{m\}]$
    $\land$ $isCrashVectorValid(m)$
    $\land$ $HandleLocalSyncStatus(m)$
    $\land$ UNCHANGED $\langle coordStateVars$, $configManagerStateVars$,
            $vLog$, $vEarlyBuffer$, $vLateBuffer$,
            $vDeadlineQuorum$, $vCrossShardConfirmQuorum$,
            $vServerClock$, $vViewChange$, $vGVec$, $vGView$,
            $vLSyncPoint$, $vLView$, $vLastNormView$,
            $vServerStatus$, $vPeerCommitDeadline$,
            $vUUIDCounter$, $vCrashVectorReps$, $vRecoveryReps\rangle$
    $\land$ $ActionName' = \langle$"HandleLocalSyncStatus"$\rangle$

$\lor$ $\exists\, m \in messages$ :
    $\land$ $m.mtype = MLocalCommit$
    $\land$ $m \notin vServerProcessed[m.dest]$
    $\land$ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
          $vServerProcessed[m.dest] \cup \{m\}]$
    $\land$ $isCrashVectorValid(m)$
    $\land$ $HandleLocalCommit(m)$
    $\land$ UNCHANGED $\langle coordStateVars, configManagerStateVars,$
          $networkVars, vLog, vEarlyBuffer, vLateBuffer,$
          $vDeadlineQuorum, vCrossShardConfirmQuorum,$
          $vServerStatus, vServerClock,$
          $vGView, vGVec, vLView, vLastNormView,$
          $vViewChange, vLSyncPoint, vPeerCommitDeadline,$
          $vLSyncQuorum, vUUIDCounter,$
          $vCrashVectorReps, vRecoveryReps\rangle$
    $\land$ $ActionName' = \langle$"HandleLocalCommit"$\rangle$

$\lor$ $\exists\, serverId \in Servers$ :
    $\land$ $BroadcastCommitStatusToPeers(serverId)$
    $\land$ UNCHANGED $\langle coordStateVars, serverStateVars,$
          $configManagerStateVars\rangle$
    $\land$ $ActionName' = \langle$"BroadcastCommitStatusToPeers"$\rangle$

$\lor$ $\exists\, m \in messages$ :
    $\land$ $m.mtype = MPeerShardCommitStatus$
    $\land$ $m \notin vServerProcessed[m.dest]$
    $\land$ $vServerProcessed' = [vServerProcessed$ EXCEPT $![m.dest] =$
          $vServerProcessed[m.dest] \cup \{m\}]$
    $\land$ $HandlePeerShardCommitStatus(m)$
    $\land$ UNCHANGED $\langle networkVars, coordStateVars, configManagerStateVars,$
          $vLog, vEarlyBuffer, vLateBuffer, vServerStatus,$
          $vDeadlineQuorum, vCrossShardConfirmQuorum,$
          $vGView, vGVec, vLView, vServerClock, vLastNormView,$
          $vViewChange, vLSyncPoint, vLCommitPoint,$
          $vPeerCommitDeadline, vLSyncQuorum, vUUIDCounter,$
          $vCrashVector, vCrashVectorReps, vRecoveryReps\rangle$
    $\land$ $ActionName' = \langle$"HandlePeerShardCommitStatus"$\rangle$

 Clock Move
$\lor$ $\exists\, serverId \in Servers$ :
    $\land$ $ServerClockMove(serverId)$
    $\land$ UNCHANGED $\langle coordStateVars, configManagerStateVars\rangle$
    $\land$ $ActionName' = \langle$"ServerClockMove"$\rangle$
    ∨ $\exists\, coordId \in Coords$ :
        ∧ $CoordClockMove(coordId)$
        ∧ UNCHANGED $\langle networkVars, serverStateVars, configManagerStateVars,$
            $vCoordTxns, vCoordProcessed\rangle$
        ∧ $ActionName' = \langle$“CoordClockMove”$\rangle$

$Spec \triangleq Init \land \Box[Next]_{\langle networkVars,$
                        $serverStateVars, coordStateVars, configManagerStateVars,$
                        $ActionName\rangle}$


$ShardRecovered(shardId, lViewID) \triangleq$
    LET
        $serversInOneShard \triangleq \{s \in Servers : s.shardId = shardId\}$
        $leaderServer \triangleq [$
            $replicaId \mapsto LeaderID(lViewID),$
            $shardId \mapsto shardId$
        $]$
    IN
    ∧ $\exists\, RM \in$ SUBSET $(serversInOneShard)$ :
        ∧ $Cardinality(RM) \geq QuorumSize$
        ∧ $leaderServer \in RM$
        ∧ $\forall\ r \in RM : vServerStatus[r] = StNormal$
        ∧ $\forall\ r \in RM : vLastNormView[r] \geq lViewID$


$CommittedInView(v, shardId, txnId) \triangleq$
    LET
        $serversInOneShard \triangleq \{s \in Servers : s.shardId = shardId\}$
        $leaderServer \triangleq [$
            $replicaId \mapsto LeaderID(v),$
            $shardId \mapsto shardId$
        $]$
        $replySet \triangleq \{$
            $m \in messages$ : ∧ ∨ $m.mtype = MFastReply$
                               ∨ $m.mtype = MSlowReply$
                             ∧ $m.txnId = txnId$
                             ∧ $m.sender \in serversInOneShard$
                             ∧ $m.lView = v$
        $\}$
    IN
    IF $\forall\, reply \in replySet$ :
        ∨ $reply.mtype \neq MFastReply$
        ∨ $reply.sender \neq leaderServer$
     THEN   No leader’s fast *reply* → This *txn* is not committed
        FALSE
```

```
ELSE
  LET
     $leaderReply \triangleq$ CHOOSE $reply \in replySet$ :
                                 $\land\ reply.mtype = MFastReply$
                                 $\land\ reply.sender = leaderServer$
  IN
   Committed in Fast Path
  $\lor\ \exists\, fastQuorum \in$ SUBSET $replySet$ :
       $\land\ leaderReply \in fastQuorum$
       $\land\ Cardinality(fastQuorum) = FastQuorumSize$
        All replies have the same hash (or it is a slow reply)
       $\land\ \forall\, reply \in fastQuorum$ :
           $\lor\ \land\ reply.mtype = MFastReply$
              $\land\ reply.hash = leaderReply.hash$
            Slow Reply can be used as fast reply
           $\lor\ reply.mtype = MSlowReply$
   Committed in Slow Path
  $\lor\ \exists\, slowQuorum \in$ SUBSET $replySet$ :
       $\land\ leaderReply \in slowQuorum$
       $\land\ Cardinality(slowQuorum) = QuorumSize$
       $\land\ \forall\, reply \in slowQuorum \setminus \{leaderReply\}$ :
            $reply.mtype = MSlowReply$
```

## Invariants

Durability [In-Shard-Property]: Committed txns always survive failure *i.e.* If a *txn* is committed (to be more precise, locally committed) in one view, then it will remain committed in the higher views.

One thing to note, the check of "committed" only happens when the system is still "normal". While the system is under recovery (*i.e.* less than $f + 1$ replicas are normal), the check of committed does not make sense

```
$Durability \triangleq$
  $\forall\, shardId \in Shards$ :
    $\forall\, v1,\ v2 \in 0\,..\,MaxViews$ :
       If a txn is committed in lower view ($v1$,),
       it is impossible to make this request uncommited in higher vie
     $\neg($ $\land\ v1 < v2$
         $\land\ ShardRecovered(shardId,\ v2)$
         $\land\ \exists\, c \in Coords$ :
              $\exists\, txnId \in vCoordTxns[c]$ :
                  $\land\ CommittedInView(v1,\ shardId,\ txnId)$
                  $\land\ \neg CommittedInView(v2,\ shardId,\ txnId)$
     $)$
```

Consistency [In-Shard-Property]: Committed txns have the same history even after view changes, *i.e.* If a request is committed in a lower view ($v1$), then (based on *Durability* Property), then it remains committed in higher view ($v2$)

Consistency requires the history of the txns (*i.e.* all the txs before this *txn*) remain the same

```
$Consistency \triangleq$
    $\forall shardId \in Shards :$
        $\forall v1, v2 \in 1 \, .. \, MaxViews :$
            $\neg($  $\land v1 < v2$
                  To check Consistency of txns in higher views,
                  the shard should have entered the higher views
                 $\land ShardRecovered(shardId, v2)$
                 $\land \exists c \in Coords :$
                     $\exists txnId \in vCoordTxns[c] :$
                         Durability has been checked in another invariant
                       IF  $\land CommittedInView(v1, shardId, txnId)$
                           $\land CommittedInView(v2, shardId, txnId)$
                        THEN
                           LET
                               $v1LeaderReply \triangleq$ CHOOSE $m \in messages :$
                                                     $\land$  $m.mtype = MFastReply$
                                                     $\land$  $m.txnId = txnId$
                                                     $\land$  $m.lView = v1$
                                                     $\land$  $m.sender.shardId = shardId$
                                                     $\land$  $m.sender.replicaId = LeaderID(v1)$
                               $v2LeaderReply \triangleq$ CHOOSE $m \in messages :$
                                                     $\land$  $m.mtype = MFastReply$
                                                     $\land$  $m.txnId = txnId$
                                                     $\land$  $m.lView = v2$
                                                     $\land$  $m.sender.shardId = shardId$
                                                     $\land$  $m.sender.replicaId = LeaderID(v2)$
                           IN
                               $v1LeaderReply.hash \neq v2LeaderReply.hash$
                        ELSE  FALSE
            $)$
```

*Linearizability* [In-Shard-Property]: Only one *txn* can be committed for a given position, *i.e.* If one *txn* has committed at position $i$, then no contrary observation can be made

*i.e.* there cannot be a second *txn* committed at the same position

```
$Linearizability \triangleq$
    LET
        $allTxns \triangleq$ UNION $\{vCoordTxns[c] : c \in Coords\}$
    IN
    $\forall shardId \in Shards :$
        $\forall txnId1, txnId2 \in allTxns :$
```

```
IF $txnId1 = txnId2$ THEN TRUE
 ELSE
   $\forall v1, v2 \in 1 \,..\, MaxViews :$
     IF  $\land$ $CommittedInView(v1, shardId, txnId1)$
         $\land$ $CommittedInView(v1, shardId, txnId2)$
      THEN
        LET
          $v1LeaderReply \triangleq$ CHOOSE $m \in messages :$
                                 $\land$  $m.mtype = MFastReply$
                                 $\land$  $m.txnId = txnId1$
                                 $\land$  $m.lView = v1$
                                 $\land$  $m.sender.shardId = shardId$
                                 $\land$  $m.sender.replicaId = LeaderID(v1)$
          $v2LeaderReply \triangleq$ CHOOSE $m \in messages :$
                                 $\land$  $m.mtype = MFastReply$
                                 $\land$  $m.txnId = txnId2$
                                 $\land$  $m.lView = v2$
                                 $\land$  $m.sender.shardId = shardId$
                                 $\land$  $m.sender.replicaId = LeaderID(v2)$
        IN
           They cannot be committed in the same log position, regardless of the view
          $v1LeaderReply.logId \neq v2LeaderReply.logId$
      ELSE  Not both are committed, so no need to check
        TRUE
```

Serializability [Cross-Shard-Property]: Given two txns and two *shards*: If they are both committed in both *shards*, then they should be committed in the same order, *i.e.*, if *txn* − 1 committed before *txn* − 2 on Shard − 1, then *txn* − 1 is also committed before *txn* − 2 on Shard − 2

```
$Serializability \triangleq$
  LET
    $allTxns \triangleq$ UNION $\{vCoordTxns[c] : c \in Coords\}$
  IN
  $\forall txnId1, txnId2 \in allTxns :$
    IF $txnId1 = txnId2$ THEN TRUE
     ELSE
      $\forall v \in 1 \,..\, MaxViews :$
        $\forall shardId1, shardId2 \in Shards :$
          IF $shardId1 = shardId2$ THEN TRUE
           ELSE
             IF  $\land$ $CommittedInView(v, shardId1, txnId1)$
                 $\land$ $CommittedInView(v, shardId1, txnId2)$
                 $\land$ $CommittedInView(v, shardId2, txnId1)$
                 $\land$ $CommittedInView(v, shardId2, txnId2)$
              THEN
                LET
```

```
                  txn1_LeaderReplyOnShard1 ≜ CHOOSE m ∈ messages :
                                               ∧ m.mtype = MFastReply
                                               ∧ m.txnId  = txnId1
                                               ∧ m.lView = v
                                               ∧ m.sender.shardId = shardId1
                                               ∧ m.sender.replicaId = LeaderID(v)
                  txn2_LeaderReplyOnShard1 ≜ CHOOSE m ∈ messages :
                                               ∧ m.mtype = MFastReply
                                               ∧ m.txnId  = txnId2
                                               ∧ m.lView = v
                                               ∧ m.sender.shardId = shardId1
                                               ∧ m.sender.replicaId = LeaderID(v)
                  txn1_LeaderReplyOnShard2 ≜ CHOOSE m ∈ messages :
                                               ∧ m.mtype = MFastReply
                                               ∧ m.txnId  = txnId1
                                               ∧ m.lView = v
                                               ∧ m.sender.shardId = shardId2
                                               ∧ m.sender.replicaId = LeaderID(v)
                  txn2_LeaderReplyOnShard2 ≜ CHOOSE m ∈ messages :
                                               ∧ m.mtype = MFastReply
                                               ∧ m.txnId  = txnId2
                                               ∧ m.lView = v
                                               ∧ m.sender.shardId = shardId2
                                               ∧ m.sender.replicaId = LeaderID(v)
              IN
              ∨ ∧ txn1_LeaderReplyOnShard1.logId > txn2_LeaderReplyOnShard1.logId
                ∧ txn1_LeaderReplyOnShard2.logId > txn2_LeaderReplyOnShard2.logId
              ∨ ∧ txn1_LeaderReplyOnShard1.logId < txn2_LeaderReplyOnShard1.logId
                ∧ txn1_LeaderReplyOnShard2.logId < txn2_LeaderReplyOnShard2.logId

          ELSE TRUE
```