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*]
$Replicas \triangleq 0 \,..\, 2$
$ReplicaOrder \triangleq \langle 0, 1, 2 \rangle$
$Shards \triangleq 0 \,..\, 2$
$Coords \triangleq 0 \,..\, 1$
$LatencyBounds \triangleq [c \in Coords \mapsto 1]$

ASSUME $IsFiniteSet(Replicas)$
ASSUME $IsFiniteSet(Shards)$
ASSUME $ReplicaOrder \in Seq(Replicas)$
$Servers \triangleq \{$
$\quad[$
$\quad\quad replicaId \mapsto e[1],$
$\quad\quad shardId \mapsto e[2]$
$\quad] : e \in Replicas \times Shards$
$\}$

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

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

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

Server Statuses (Within Shard)
$StNormal \triangleq 1$
$StViewChange \triangleq 2$
$StCrossShardSyncing \triangleq 3$
$StRecovering \triangleq 4$
$StFailing \triangleq 5$

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

$TxnID = [$
$\quad coordId \mapsto i \text{ in } (1 \ldots),$
$\quad rId \quad \mapsto i \text{ in } (1 \ldots)$
$]$

$Txn = [$
$\quad mtype \mapsto MTxn$
$\quad txnId \mapsto TxnID,$
$\quad shards \mapsto Shards,$
$\quad command \mapsto command,$
$\quad st \quad \mapsto sendTime,$
$\quad bound \mapsto latencyBound$
$]$

$LogEntry = [$
$\quad mtype \mapsto MLogEntry$
$\quad txnId \mapsto TxnID,$
$\quad shards \mapsto Shards,$
$\quad command \mapsto command,$
$\quad deadline \mapsto deadline$
$]$

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

$DeadlineNotification = [$
$\quad mtype \mapsto MDeadlineNotification,$
$\quad gView \mapsto 0 \ldots x$
$\quad lView \mapsto 0 \ldots y$
$\quad sender \mapsto src \in Servers,$
$\quad dest \mapsto dst \in Servers,$
$\quad entry \mapsto LogEntry$
$]$

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

$InterReplicaSync = [$
$mtype \mapsto MInterReplicaSync,$
$lView \mapsto 0 \ldots y$
$sender \mapsto src \in Servers,$
$dest \mapsto dst \in Servers,$
$entries \mapsto [LogEntry \ldots]$
]

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

$FastReply = [$
$mtype \mapsto MFastReply,$
$sender \mapsto src \in Servers,$
$dest \mapsto dst \in Coords,$
$gView \mapsto 0 \ldots x$
$lView \mapsto 0 \ldots x$
$txnId \mapsto txnId$

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

$hash \mapsto$ [$entries \mapsto log$ entries so far $cv \mapsto crashVector$]
$deadline \mapsto i \in (1 \,..\, MaxTime + MaxBound),$
$logId \mapsto n \in (1 \,..\,)$
]

$SlowReply = [$
$mtype \mapsto MSlowReply,$
$sender \mapsto src \in Servers,$
$dest \mapsto c \in Coords,$
$gView \mapsto 0 \ldots x$
$lView \mapsto 0 \ldots x$
$txnId \mapsto txnId$
$logId \mapsto n \in (1 \,..\,)$
]

$ViewChangeReq = [$
$mtype \mapsto MViewChangeReq,$
$sender \mapsto src \in Replicas,$ (by *configManager*)
$dest \mapsto dst \in Servers,$
$gView \mapsto 0 \,..\, x$
$gVec \mapsto$ the *lViews* for each shard
]

$ViewChange = [$
$mtype \mapsto MViewChange,$
$sender \mapsto src \in Servers,$
$dest \mapsto dst \in Servers,$
$gView \mapsto 0 \,..\, x$
$gVec \mapsto$ the *lViews* for each shard
$lView \mapsto 0 \ldots x$
$lastNormal \mapsto v \in ViewIDs,$

$lSyncPoint \mapsto 0\,..$
$entries \quad \mapsto l \in vLogs[1\,..\,n],$
$cv \quad \mapsto$ crash vector
]

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

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

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

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

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

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

$lView \mapsto 0 \,..\, x$
$cv \quad \mapsto vector\ of\ counters$
]

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

Follower reports to its leader

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

Leader notifies its followers

$LocalCommit = [$
$mtype \quad \mapsto MLocalCommit,$
$sender \quad \mapsto src \in Servers,$
$dest \quad \mapsto dst \in Servers,$
$lView \quad \mapsto 0 \ldots x$
$entries \quad \mapsto log$ 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)

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

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*

$CMPrepare = [$

$mtype \mapsto MCMPrepare$,
$sender \mapsto src \in Servers$,
$dest \mapsto dst \in Servers$,
$cView \mapsto 0 \,..\, x$
$gView \mapsto 0 \,..\, x$
$gVec \mapsto [shardId \mapsto lView]$
]

$CMPrepareReply = [$
$mtype \mapsto MCMPrepareReply$,
$sender \mapsto src \in Servers$,
$dest \mapsto dst \in Servers$,
$cView \mapsto 0 \,..\, x$
$gView \mapsto 0 \,..\, x$
]

$CMCommit = [$
$mtype \mapsto MCMPrepareReply$,
$sender \mapsto src \in Servers$,
$dest \mapsto dst \in Servers$,
$cView \mapsto 0 \,..\, x$
$gView \mapsto 0 \,..\, x$
]

Network State

The variables record the messages processed by *Replicas*/Clients, so that the *Replicas*/Clients will not process twice

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

vCMStatus,
vCMView,

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

vCMGInfo,
vCMPrepareGInfo,

Config Manager: quorum of *CMPrepareReplies*

vCMPrepareReps,

vCMProcessed

VARIABLES *ActionName*

$networkVars \triangleq \langle messages \rangle$

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

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

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

$InitNetworkState \triangleq messages = \{\}$

$InitServerState \triangleq$
$\land\ vServerProcessed = [serverId \in Servers \mapsto \{\}]$
$\land\ vLog = [serverId \in Servers \mapsto \langle\rangle]$

$\land$ $vEarlyBuffer = [serverId \in Servers \mapsto \{\}]$
$\land$ $vLateBuffer = [serverId \in Servers \mapsto \{\}]$
$\land$ $vDeadlineQuorum = [serverId \in Servers \mapsto \{\}]$
$\land$ $vCrossShardConfirmQuorum = [serverId \in Servers \mapsto \{\}]$
$\land$ $vServerStatus = [serverId \in Servers \mapsto StNormal]$
$\land$ $vGView = [serverId \in Servers \mapsto 0]$
$\land$ $vGVec = [$
$serverId \in Servers \mapsto [$
$shardId \in Shards \mapsto 0$
$]$
$]$
$\land$ $vLView = [serverId \in Servers \mapsto 0]$
$\land$ $vServerClock = [serverId \in Servers \mapsto 1]$
$\land$ $vLastNormView = [serverId \in Servers \mapsto 0]$
$\land$ $vViewChange = [serverId \in Servers \mapsto \{\}]$
$\land$ $vLSyncPoint = [serverId \in Servers \mapsto 0]$
$\land$ $vLCommitPoint = [serverId \in Servers \mapsto 0]$
$\land$ $vPeerCommitDeadline = [serverId \in Servers \mapsto$
$[shardId \in Shards \mapsto 0]$
$]$
$\land$ $vLSyncQuorum = [serverId \in Servers \mapsto \{\}]$
$\land$ $vUUIDCounter = [serverId \in Servers \mapsto 0]$
$\land$ $vCrashVector = [$
$serverId \in Servers \mapsto [$
$rr \in Replicas \mapsto 0$
$]$
$]$
$\land$ $vCrashVectorReps = [serverId \in Servers \mapsto \{\}]$
$\land$ $vRecoveryReps = [serverId \in Servers \mapsto \{\}]$

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

$InitConfigManagerState \triangleq$
$\land$ $vCMStatus = [$
$replicaId \in Replicas \mapsto StNormal$
$]$
$\land$ $vCMView = [$
$replicaId \in Replicas \mapsto 0$
$]$
$\land$ $vCMGInfo = [$
$replicaId \in Replicas \mapsto [$

$$
\begin{array}{l}
\quad\quad\quad gView \quad \mapsto 0, \\
\quad\quad\quad gVec \quad\;\; \mapsto [shardId \in Shards \mapsto 0] \\
\quad\quad] \\
\quad] \\
\land \;\; vCMPrepareGInfo = [\\
\quad\quad replicaId \;\in Replicas \mapsto [\\
\quad\quad\quad gView \quad \mapsto 0, \\
\quad\quad\quad gVec \quad\;\; \mapsto [shardId \in Shards \mapsto 0] \\
\quad\quad] \\
\quad] \\
\land \;\; vCMPrepareReps = [\\
\quad\quad replicaId \in Replicas \mapsto \;\{\} \\
\quad] \\
\land \;\; vCMProcessed = [\\
\quad\quad replicaId \in Replicas \mapsto \{\} \\
\quad]
\end{array}
$$

$PickMax(S) \triangleq$ CHOOSE $x \in S : \forall\, y \in S : y \leq x$

$PickMin(S) \triangleq$ CHOOSE $x \in S : \forall\, y \in S : y \geq x$

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

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

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

$SeqToSet(s) \triangleq$
$\{s[i] : i \in$ 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.

$\forall\, i,\, j \in$ 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.

CHOOSE $f \in [1 \,..\, Cardinality(S) \to S] : IsInjective(f)$

$Remove(s,\, e) \triangleq$

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 $\langle\rangle$ 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   ∧ gv1 = gv2
        ∧ lv1 > lv2
    THEN    TRUE
    ELSE    FALSE
```

Client action

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

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

$CoordSubmitTxn(c) \triangleq$

$\wedge\ vCoordClock[c] < MaxTime$

$\wedge\ Cardinality(vCoordTxns[c]) < MaxReqNum$

$\wedge$ LET

$txnId \triangleq [$

$coordId \mapsto c,$

$rId \mapsto Cardinality(vCoordTxns[c]) + 1$

$]$

IN

$\wedge\ Send(\{[mtype \mapsto MTxn,$

$txnId \mapsto txnId,$

$command \mapsto$ "",

Here we assume involves all *shards*

```
                shards   ↦ Shards,
                st       ↦ vCoordClock[c],
                bound    ↦ LatencyBounds[c],
                sender   ↦ c,
                dest     ↦ serverId
            ] : serverId   ∈ Servers})
        ∧ vCoordClock' = [vCoordClock EXCEPT ![c] = vCoordClock[c] + 1]
        ∧ vCoordTxns' = [vCoordTxns EXCEPT ![c] = vCoordTxns[c] ∪ {txnId}]

HandleTxn(m) ≜
    LET
        myServerId ≜ m.dest
        newLog ≜ [
            mtype     ↦ MLogEntry,
            txnId     ↦ m.txnId,
            command ↦ m.command,
            shards    ↦ m.shards,
            deadline  ↦ Max(LastAppendedDeadline(vLog[myServerId]), m.st + m.bound)
        ]
        serversInOneReplica ≜ {s ∈ Servers : s.replicaId = myServerId.replicaId}
    IN
    ∨ ∧ isLeader(myServerId.replicaId, vLVView[myServerId])
      ∧ vEarlyBuffer' = [
          vEarlyBuffer EXCEPT ![myServerId]
              = vEarlyBuffer[myServerId] ∪ {newLog}]
       Broadcast deadline notifications to other shards
      ∧ Send({[
          mtype   ↦ MDeadlineNotification,
          gView   ↦ vGView[myServerId],
          lView   ↦ vLView[myServerId],
          sender  ↦ myServerId,
          dest    ↦ dstServerId,
          entry   ↦ newLog
          ] : dstServerId ∈ serversInOneReplica})
      ∧ UNCHANGED ⟨vLateBuffer⟩
    ∨ ∧ ¬isLeader(myServerId.replicaId, vLVView[myServerId])
      ∧ ∨ ∧ newLog.deadline = (m.st + m.bound)
          ∧ vEarlyBuffer' = [
                vEarlyBuffer EXCEPT ![myServerId]
                    = vEarlyBuffer[myServerId] ∪ {newLog}
            ]
          ∧ UNCHANGED ⟨vLateBuffer⟩
        ∨ ∧ ¬(newLog.deadline = (m.st + m.bound))
          ∧   vLateBuffer' = [
```

```
                    vLateBuffer EXCEPT ![myServerId]
                        = vLateBuffer[myServerId] ∪ {newLog}
                ]
            ∧ UNCHANGED ⟨vEarlyBuffer⟩
        ∧ UNCHANGED ⟨networkVars⟩


HandleDeadlineNotification(m) ≜
    LET
        myServerId ≜ m.dest
        quorum ≜ {
            msg ∈ vDeadlineQuorum[myServerId]
                :   ∧ msg.entry.txnId = m.entry.txnId
                    ∧ msg.gView = m.gView
                    ∧ m.gView = vGView[myServerId]
            } ∪ {m}
    IN
     Only leader does deadline agreement
    ∧ vGView[myServerId] = m.gView
    ∧ vGVec[myServerId][m.sender.shardId] = m.lView
    ∧ isLeader(myServerId.replicaId, vLView[myServerId])
    ∧ vDeadlineQuorum' = [
            vDeadlineQuorum EXCEPT ![myServerId]
                = vDeadlineQuorum[myServerId] ∪ {m}
       ]
    ∧ IF Cardinality(quorum) = Cardinality(m.entry.shards)
       THEN
       Deadline quorum established : Update the deadline of the txn in Sequencer
          LET
              maxDeadlineTxn ≜
                  CHOOSE x ∈ quorum :
                      ∀ y     ∈ quorum :
                          y.entry.deadline ≤ x.entry.deadline
              sequencingTxn ≜
                  CHOOSE x ∈ vEarlyBuffer[myServerId] :
                      x.txnId = m.entry.txnId
          IN
          IF maxDeadlineTxn.entry.deadline > sequencingTxn.deadline
           THEN
              vEarlyBuffer' = [vEarlyBuffer EXCEPT ![myServerId]
                  = (vEarlyBuffer[myServerId] \ {sequencingTxn}) ∪ {maxDeadlineTxn.entry}]
           ELSE UNCHANGED ⟨vEarlyBuffer⟩

       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
$myServerId \triangleq m.dest$
$syncedTxnIds \triangleq \{m.entries[i].txnId : i \in 1\ ..\ Len(m.entries)\}$
$currentSyncPoint \triangleq Len(vLSyncPoint[myServerId])$
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      ↦ m.entries[i].txnId.coordId,
                    gView     ↦ vGView[myServerId],
                    lView     ↦ vLView[myServerId],
                    txnId     ↦ m.entries[i].txnId,
                    logId     ↦ i
                ] : i ∈ (currentSyncPoint + 1) .. Len(m.entries)})
        ∨  ∧ currentSyncPoint ≥ Len(m.entries)
           Noting new to sync
           ∧ UNCHANGED ⟨networkVars, vLog, vEarlyBuffer,
                        vLateBuffer, vDeadlineQuorum, vLSyncPoint⟩


StartLeaderFail(serverId) ≜
   This leader fails
  LET
     serversInOneShard ≜ {
        s ∈ Servers : s.shardId = serverId.shardId
     }
     aliveReplicas ≜ {
        s ∈ serversInOneShard :    ∧ vServerStatus[s] = StNormal
                                   ∧ s ≠ 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 ⟨vServerStatus⟩


DetectLeaderFail(cmReplicaId) ≜
  ∃ shardId ∈ Shards :
    LET
       lView ≜ vCMGInfo[cmReplicaId].gVec[shardId]
       leaderId ≜ LeaderID(lView)
       serverId ≜ [
          replicaId   ↦ leaderId,
          shardId     ↦ shardId
       ]
    IN
     vServerStatus[serverId] = StFailing

SelectProperLView(currentView, shardId) ≜
  LET
     aliveReplicaId ≜ CHOOSE replicaId ∈ 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 ÷ Cardinality(Replicas) + 1) * Cardinality(Replicas) + aliveReplicaId

PrepareViewChange(cmReplicaId) ≜
    LET
        newGVec ≜ [
            shardId ∈ Shards ↦
                SelectProperLView(vCMGInfo[cmReplicaId].gVec[shardId], shardId)
        ]
    IN
    ∧ vCMPrepareGInfo' = [vCMPrepareGInfo EXCEPT ![cmReplicaId] =
            [
                gView   ↦ vCMGInfo[cmReplicaId].gView + 1,
                gVec    ↦  newGVec
            ]
        ]
    ∧ Send({[
            mtype   ↦ MCMPrepare,
            sender  ↦ cmReplicaId,
            dest    ↦ dstRid,
            cView   ↦ vCMView[cmReplicaId],
            gView   ↦ vCMPrepareGInfo'[cmReplicaId].gView,
            gVec    ↦  newGVec

        ] : dstRid ∈ Replicas})

LaunchViewChange(cmReplicaId) ≜
    IF  ∧ isLeader(cmReplicaId, vCMView[cmReplicaId])
        ∧ DetectLeaderFail(cmReplicaId)
     THEN
        PrepareViewChange(cmReplicaId)
     ELSE
        UNCHANGED ⟨networkVars⟩


HandleCMPrepare(m) ≜
    ∧ m.cView = vCMView[m.dest]
    ∧ m.gView > vCMGInfo[m.dest].gView
    ∧ vCMPrepareGInfo' = [vCMPrepareGInfo EXCEPT ![m.dest] =
            [
                gView   ↦ m.gView,
                gVec    ↦ m.gVec
```

```
            ]
        ]
    ∧ Send({[
            mtype  ↦ MCMPrepareReply,
            sender ↦ m.dest,
            dest   ↦ m.src,
            cView  ↦ m.cView,
            gView  ↦ m.gView
        ]})

HandleCMPrepareReply(m) ≜
    ∧ 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] ∪ {m}
        ]
    ∧ LET
            quorum ≜ {mm ∈ 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  ↦ MCMCommit,
                    sender ↦ m.dest,
                    dest   ↦ rid,
                    cView  ↦ m.cView,
                    gView  ↦ m.gView
                ] : rid ∈ {r ∈ Replicas : r ≠ m.dest}})
            start view change, broadcast view change request to every server
            ∧ Send({[
                    mtype  ↦ MViewChangeReq,
                    sender ↦ m.dest,
                    dest   ↦ serverId,
                    gView  ↦ vCMGInfo'[m.dest].gView,
                    gVec   ↦ vCMGInfo'[m.dest].gVec
                ] : serverId ∈ Servers})
        ELSE
            UNCHANGED ⟨networkVars, vCMGInfo⟩

HandleCMCommit(m) ≜
```

```
    ∧ m.cView = vCMView[m.dest]
    ∧ ¬isLeader(m.dest, vCMView[m.dest])
    ∧ m.gView = vCMPrepareGInfo[m.dest].gView
    ∧ vCMGInfo' = [vCMGInfo EXCEPT ![m.dest] =
                      vCMPrepareGInfo[m.dest]
      ]


HandleViewChangeReq(m) ≜
  LET
      myServerId ≜ m.dest
      myLeader ≜ CHOOSE s ∈ Servers :
                    ∧ s.replicaId = LeaderID(m.gVec[myServerId.shardId])
                    ∧ s.shardId = myServerId.shardId
  IN
   If the msg's view is lower, ignore
    ∧ vGView[myServerId] < m.gView
    ∧ IF vServerStatus[myServerId] = StNormal THEN
          ∧ vServerStatus'    = [vServerStatus EXCEPT ![myServerId]    = StViewChange]
          ∧ vLastNormView' = [vLastNormView EXCEPT ![myServerId] = vLView[myServerId]]
      ELSE     UNCHANGED ⟨vServerStatus, vLastNormView⟩
    ∧ vGView' = [
          vGView EXCEPT ![myServerId] = m.vGView
      ]
    ∧ vGVec' = [
          vGVec EXCEPT ![myServerId] = m.gVec
      ]
    ∧ vLView' = [
          vLView EXCEPT ![myServerId] = m.gVec[myServerId.shardId]
      ]
   Clear ealry buffer,
    ∧ vEarlyBuffer' = [
          vEarlyBuffer EXCEPT ![myServerId] = {}
      ]
   Clear late buffer
    ∧ vLateBuffer' = [
          vLateBuffer EXCEPT ![myServerId] = {}
      ]
   Clear deadline quorum
    ∧ vDeadlineQuorum' = [
          vDeadlineQuorum EXCEPT ![myServerId] = {}
      ]
   Clear vCrossShardConfirmQuorum
    ∧ vCrossShardConfirmQuorum' = [
          serverId ∈ Servers ↦ {}
```

```
    ]
  Send ViewChange to the myLeader
  ∧ Send({[
        mtype      ↦ MViewChange,
        sender     ↦ myServerId,
        dest       ↦ myLeader,
        gView      ↦ m.vGView,
        gVec       ↦ m.gVec,
        lView      ↦ vLView'[myServerId],
        lastNormal ↦ vLastNormView'[myServerId],
        lSyncPoint ↦ vLSyncPoint[myServerId],
        entries    ↦ vLog[myServerId],
        cv         ↦ vCrashVector[myServerId]
    ]})

  Define a comparison function based on the key
Compare(a, b) ≜
  ∨   a.deadline < b.deadline
  ∨ ∧ a.deadline = b.deadline
    ∧ a.txnId.coordId < b.txnId.coordId
  ∨ ∧ a.deadline = b.deadline
    ∧ a.txnId.coordId = b.txnId.coordId
    ∧ a.txnId.rId < b.txnId.rId

isCrashVectorValid(m) ≜
  ∧ ∀ rr ∈ Replicas : vCrashVector[m.dest][rr] ≤ m.cv[rr]
  ∧ vCrashVector' = [
      vCrashVector EXCEPT ![m.dest] = [
        rr ∈ Replicas ↦ Max(m.cv[rr], vCrashVector[m.dest][rr])
      ]
    ]

CountVotes(entry, logSets) ≜
  LET
    validCandidates ≜ {s ∈ logSets : ∃ e ∈ s :
                          ∧ e.deadline = entry.deadline
                          ∧ e.txnId = entry.txnId}
  IN
    Cardinality(validCandidates)

ReBuildLogs(vcQuorum) ≜
  LET
    refinedQuorum ≜ {m ∈ vcQuorum :
                       ∀ msg ∈ vcQuorum : msg.lastNormal ≤ m.lastNormal}
```

```
        lSyncPoints ≜ {m.lSyncPoint : m ∈ refinedQuorum}
        largestLSyncPointVC ≜ CHOOSE vc ∈ refinedQuorum :
                                ∀ sp ∈ lSyncPoints : sp ≤ vc.lSyncPoint
        syncedLogSeq ≜ SubSeq(largestLSyncPointVC.entries, 1, largestLSyncPointVC.lSyncPoint)
        deadlineBoundary ≜ IF largestLSyncPointVC.lSyncPoint = 0 THEN 0
                           ELSE syncedLogSeq[largestLSyncPointVC.lSyncPoint].deadline
        logSets ≜ {SeqToSet(m.entries) : m ∈ refinedQuorum}
        allLogs ≜ UNION logSets
        allUnSyncedLogs ≜ {entry ∈ allLogs : entry.deadline > deadlineBoundary}
        unSyncedLogs ≜ {entry ∈ allUnSyncedLogs :
            CountVotes(entry, logSets) ≥ RecoveryQuorumSize}
        unSyncedLogSeq ≜ SetToSortSeq(unSyncedLogs, Compare)
    IN
    syncedLogSeq ∘ unSyncedLogSeq

SelectEntriesBeyondCommitPoint(entries, deadline) ≜
    LET
        validLogIndices ≜ {
            i ∈ 1 .. Len(entries) : entries[i].deadline > deadline
        }
        startIndex ≜ PickMin(validLogIndices)
    IN
    SubSeq(entries, startIndex, Len(entries))

HandleViewChange(m) ≜
    LET
        myServerId ≜ m.dest
        serversInOneShard ≜ {s ∈ Servers : s.shardId = myServerId.shardId}
        leadersInAllShard ≜ {
                s ∈ Servers : s.replicaId = isLeader(s.replicaId, m.gVec[s.shardId])
        }
    IN
    ∧ ∨ ViewGreater(m.gView, m.lView, vGView[myServerId], vLView[myServerId])
      ∨ ∧ m.gView = vGView[myServerId]
        ∧ m.lView = vLView[myServerId]
        ∧ vServerStatus[myServerId] = StViewChange
    ∧ isLeader(myServerId.replicaId, m.lView)
    ∧ vGView' = [vGView EXCEPT ![myServerId] = m.gView]
    ∧ vLView' = [vLView EXCEPT ![myServerId] = m.lView]
    ∧ vGVec' = [vGVec EXCEPT ![myServerId] = m.gVec]
    ∧ vViewChange' = [
        vViewChange EXCEPT ![myServerId] = {
            vc ∈ vViewChange[myServerId] :
                vc.lView = m.lView
        } ∪ {m}
```

```
    ]
  ∧ IF Cardinality(vViewChange'[myServerId]) = QuorumSize THEN
      ∧ vLog' = [vLog EXCEPT ![myServerId] = ReBuildLogs(vViewChange'[myServerId])]
      ∧ vServerStatus'    = [vServerStatus EXCEPT ![myServerId]    = StCrossShardSyncing]
      ∧ 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
      ∧ vViewChange' = [vViewChange EXCEPT ![myServerId] = {}]
      ∧ Send({[
              mtype      ↦ MCrossShardConfirm,
              sender     ↦ myServerId,
              dest       ↦ dst,
              lView      ↦ vLView'[myServerId],
              gView      ↦ vGView'[myServerId],
              entries    ↦ SelectEntriesBeyondCommitPoint(
                               vLog'[myServerId], vPeerCommitDeadline[dst.shardId])
          ] : dst ∈ leadersInAllShard})
    ELSE
      ∧ vServerStatus' = [vServerStatus EXCEPT ![myServerId] = StViewChange]
      ∧ UNCHANGED ⟨networkVars, vLog, vServerStatus, vViewChange⟩

BuildGlobalConsistentLog(serverId, entries) ≜
  LET
    myEntries ≜ {
      entry ∈ entries : ∧ serverId ∈ entry.shards
                        ∧ ∀ e ∈ entries :
                            IF e.txnId = entry.txnId THEN
                                e.deadline ≤ entry.deadline
                             ELSE TRUE
    }
  IN
  SetToSortSeq(myEntries, Compare)

HandleCrossShardConfirm(m) ≜
  LET
    myServerId ≜ m.dest
  IN
  ∧ vServerStatus[myServerId] = StCrossShardSyncing
  ∧ m.gView = vGView[myServerId]
  ∧ m.lView  = vGVec[myServerId][m.sender.shardId]
  ∧ vCrossShardConfirmQuorum' = [
      vCrossShardConfirmQuorum EXCEPT ![myServerId] = {
        mm ∈ vCrossShardConfirmQuorum[myServerId] :
          ∧ mm.gView = vGView[myServerId]
```

```
                    ∧ mm.lView = vGVec[myServerId][mm.sender.shardId]
            } ∪ {m}
        ]
    ∧ 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 ≜ UNION {SeqToSet(mm.entries) :
                                  mm ∈ vCrossShardConfirmQuorum'[myServerId]}
              serversInOneShard ≜ {s ∈ Servers : s.shardId = myServerId.shardId}
          IN
          ∧ vLog' = [
                vLog EXCEPT ![myServerId] =
                    BuildGlobalConsistentLog(m.sender, allLogs)
            ]
          ∧ Send({[
                mtype      ↦ MStartView,
                sender     ↦ myServerId,
                dest       ↦ dst,
                lView      ↦ vLView[myServerId],
                gView      ↦ vGView[myServerId],
                gVec       ↦ vGVec[myServerId],
                entries    ↦ vLog'[myServerId],
                cv         ↦ vCrashVector[myServerId]
            ] : dst ∈ serversInOneShard})
      ELSE
          UNCHANGED ⟨vLog, networkVars⟩

HandleStartView(m) ≜
    LET
        myServerId ≜ m.dest
    IN
    ∧ ∨ ViewGreater(m.gView, m.lView, vGView[myServerId], vLView[myServerId])
      ∨ ∧ m.gView = vGView[myServerId]
        ∧ m.lView = vLView[myServerId]
        ∧ ∨ vServerStatus[myServerId] = StViewChange
          ∨ vServerStatus[myServerId] = StRecovering
    ∧ vGView' = [vGView EXCEPT ![myServerId] = m.gView]
    ∧ vLView' = [vLView EXCEPT ![myServerId] = m.gLView]
    ∧ vGVec' = [vGVec EXCEPT ![myServerId] = m.vGVec]
    ∧ vServerStatus' = [vServerStatus EXCEPT ![myServerId] = StNormal]
    ∧ vLog' = [vLog EXCEPT ![myServerId] = m.entries]
```

```
    ∧ vEarlyBuffer' = [vEarlyBuffer EXCEPT ![myServerId] = {}]
    ∧ vLateBuffer' = [vLateBuffer EXCEPT ![myServerId] = {}]
    ∧ vDeadlineQuorum' = [vDeadlineQuorum EXCEPT ![myServerId] = {}]
    ∧ vCrossShardConfirmQuorum' = [
          vCrossShardConfirmQuorum EXCEPT ![myServerId] = {}
      ]
    ∧ vLSyncPoint' = [vLSyncPoint EXCEPT ![myServerId] = Len(vLog'[myServerId])]
    ∧ vLastNormView' = [vLastNormView EXCEPT ![myServerId] = m.lView]
    ∧ vViewChange' = [vViewChange EXCEPT ![myServerId] = {}]
    ∧ vLSyncQuorum' = [vLSyncQuorum EXCEPT ![myServerId] = {}]
    ∧ vCrashVectorReps' = [vCrashVectorReps EXCEPT ![myServerId] = {}]
    ∧ vRecoveryReps' = [vRecoveryReps EXCEPT ![myServerId] = {}]

ResetServerState(serverId) ≜
    ∧ vLog' = [vLog EXCEPT ![serverId] = ⟨⟩]
    ∧ vEarlyBuffer' = [vEarlyBuffer EXCEPT ![serverId] = {}]
    ∧ vLateBuffer' = [vLateBuffer EXCEPT ![serverId] = {}]
    ∧ vDeadlineQuorum' = [vDeadlineQuorum EXCEPT ![serverId] = {}]
    ∧ vCrossShardConfirmQuorum' = [
          vCrossShardConfirmQuorum EXCEPT ![serverId] = {}
      ]
    ∧ vGView' = [vGView EXCEPT ![serverId] = 0]
    ∧ vGVec' = [vGVec EXCEPT ![serverId] = [s ∈ Shards ↦ 0]]
    ∧ vLView' = [vLView EXCEPT ![serverId] = 0]
    ∧ vLastNormView' = [vLastNormView EXCEPT ![serverId] = 0]
    ∧ vViewChange' = [vViewChange EXCEPT ![serverId] = {}]
    ∧ vLSyncPoint' = [vLSyncPoint EXCEPT ![serverId] = 0]
    ∧ vLCommitPoint' = [vLCommitPoint EXCEPT ![serverId] = 0]
    ∧ vPeerCommitDeadline' = [vPeerCommitDeadline EXCEPT ![serverId] = 0]
    ∧ vLSyncQuorum' = [vLSyncQuorum EXCEPT ![serverId] = {}]
    ∧ vCrashVector' = [vCrashVector EXCEPT ![serverId] = [
          rr ∈ Replicas ↦ 0
      ]]
    ∧ vCrashVectorReps' = [vCrashVectorReps EXCEPT ![serverId] = {}]
    ∧ vRecoveryReps' = [vRecoveryReps EXCEPT ![serverId] = {}]
    ∧ vServerProcessed' = [vServerProcessed EXCEPT ![serverId] = {}]

StartServerRecovery(serverId) ≜
  LET
      serversInOneShard ≜ {
          s ∈ Servers : s.shardId = serverId.shardId
      }
      nonce ≜ vUUIDCounter[serverId] + 1
  IN
    ∧ vServerStatus' = [vServerStatus EXCEPT ![serverId] = StRecovering]
```

```
    ∧ vUUIDCounter' = [vUUIDCounter EXCEPT ![serverId] = vUUIDCounter[serverId] + 1]
    ∧ ResetServerState(serverId)
    ∧ Send({[
            mtype      ↦ MCrashVectorReq,
            sender     ↦ serverId,
            dest       ↦ dst,
            nonce      ↦ nonce
      ] : dst ∈ serversInOneShard})

HandleCrashVectorReq(m) ≜
    LET
        myServerId ≜ 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 ≜ m.dest
    IN
    ∧ vServerStatus[myServerId] = StNormal
    ∧ Send({[
            mtype   ↦ MRecoveryRep,
            sender  ↦ myServerId,
            dest    ↦ m.sender,
            gView   ↦ vGView[myServerId],
            lView   ↦ vLView[myServerId],
            cv      ↦ vCrashVector'[myServerId]
        ]})

HandleRecoveryRep(m) ≜
    LET
        myServerId ≜ m.dest
    IN
    ∧ vServerStatus[myServerId] = StRecovering
    ∧ vRecoveryReps' = [
        vRecoveryReps EXCEPT ![myServerId]
            = vRecoveryReps[myServerId] ∪ {m}
      ]
    ∧ IF Cardinality(vRecoveryReps[myServerId]) = QuorumSize THEN
        LET
            lViewQuorum ≜ {mm.lView : mm ∈ vRecoveryReps[myServerId]}
            gViewQuorum ≜ {mm.gView : mm ∈ vRecoveryReps[myServerId]}
        IN
        ∧ vLView' = [vLView EXCEPT ![myServerId] = PickMax(lViewQuorum)]
        ∧ vGView' = [vLView EXCEPT ![myServerId] = PickMax(gViewQuorum)]
        ∧ Send({[
                mtype       ↦ MStartViewReq,
                sender      ↦ myServerId,
```

```
                    dest        ↦ [
                                    replicaId ↦ LeaderID(vLView[myServerId]),
                                    shardId   ↦ myServerId.shardId
                                ],
                    lView       ↦ vLView'[myServerId],
                    cv          ↦ vCrashVector'[myServerId]
                ]})
        ELSE UNCHANGED ⟨networkVars, vLView, vGView⟩


HandleStartViewReq(m) ≜
    LET
        myServerId ≜ m.dest
    IN
    ∧ vServerStatus[myServerId] = StNormal
    ∧ vLView[myServerId] = m.lView
    ∧ isLeader(myServerId.replicaId, vLView[myServerId])
    ∧ Send({[
            mtype       ↦ MStartView,
            sender      ↦ myServerId,
            dest        ↦ m.sender,
            lView       ↦ vLView[myServerId],
            gView       ↦ vGView[myServerId],
            gVec        ↦ vGVec[myServerId],
            entries     ↦ vLog[myServerId],
            cv          ↦ vCrashVector[myServerId]
        ]})


StartLocalSync(serverId) ≜
    LET
        leaderServerId ≜ [
            replicaId ↦ LeaderID(vLView[serverId]),
            shardId   ↦ serverId.shardId
        ]
    IN
    ∧ vServerStatus[serverId] = StNormal
    ∧ Send({[
            mtype       ↦ MLocalSyncStatus,
            sender      ↦ serverId,
            dest        ↦ leaderServerId,
            lView       ↦ vLView[serverId],
            lSyncPoint  ↦ vLSyncPoint[serverId],
            cv          ↦ vCrashVector[serverId]
        ]})
```

```
HandleLocalSyncStatus(m) ≜
    LET
        myServerId ≜ m.dest
        lSyncQuorum ≜ vLSyncQuorum[myServerId]
    IN
     ∧ vServerStatus[myServerId] = StNormal
     ∧ vLView[myServerId] = m.lView
     ∧ isLeader(myServerId.replicaId, vLView[myServerId])
     ∧ ∀ mm ∈ lSyncQuorum :
        ∨ mm.sender ≠ m.sender
        ∨ mm.lSyncPoint < m.lSyncPoint
     ∧ vLSyncQuorum' = [
            vLSyncQuorum EXCEPT ![myServerId] =
                {mm ∈ lSyncQuorum : mm.sender ≠ m.sender} ∪ {m}
       ]
     ∧ IF Cardinality(vLSyncQuorum'[myServerId]) ≥ QuorumSize THEN
            LET
                candidateQuorum ≜ {
                    R ∈ SUBSET (vLSyncQuorum'[myServerId]) :
                        Cardinality(R) = QuorumSize
                }
                quorumSyncPoints ≜ {
                    {x.lSyncPoint : x ∈ R} : R ∈ candidateQuorum
                }
                validCommitPoints ≜ {PickMax(Q) : Q ∈ quorumSyncPoints}
                maxCommitPoint ≜ PickMax(validCommitPoints)
            IN
             ∧ vLCommitPoint' = [vLCommitPoint EXCEPT ![myServerId] = maxCommitPoint]
             ∧ Send({[
                    mtype       ↦ MLocalCommit,
                    sender      ↦ myServerId,
                    dest        ↦ m.sender,
                    lView       ↦ vLView[myServerId],
                    lCommitPoint ↦ vLCommitPoint'[myServerId],
                    cv          ↦ vCrashVector'[myServerId]
                ]})
        ELSE UNCHANGED ⟨vLCommitPoint, networkVars⟩


HandleLocalCommit(m) ≜
    LET
        myServerId ≜ m.dest
    IN
     ∧ vServerStatus[myServerId] = StNormal
     ∧ vLView[myServerId] = m.lView
```

```
    ∧ ¬isLeader(myServerId.replicaId, vLView[myServerId])
     Make sure the syncPoint is large enough before updating CommitPoint
    ∧ IF   ∧ vLSyncPoint[myServerId] ≥ m.lCommitPoint
           ∧ vLCommitPoint[myServerId] < m.lCommitPoint
      THEN
         vLCommitPoint' = [
             vLCommitPoint EXCEPT ![myServerId] = m.lCommitPoint
         ]
      ELSE UNCHANGED ⟨vLCommitPoint⟩


BroadcastCommitStatusToPeers(serverId) ≜
    LET
        serversInOneReplica ≜ {s ∈ Servers : s.replicaId = serverId.replicaId}
        commitPoint ≜ vLCommitPoint[serverId]
        commitDeadline ≜
            IF commitPoint = 0 THEN 0
             ELSE vLog[commitPoint].deadline
    IN
    ∧ vServerStatus[serverId] = StNormal
    ∧ Send({[
            mtype      ↦ MPeerShardCommitStatus,
            sender     ↦ serverId,
            dest       ↦ dst,
            gView      ↦ vGView[serverId],
            lView      ↦ vLView[serverId],
            deadline   ↦ commitDeadline
       ] : dst ∈ serversInOneReplica})


HandlePeerShardCommitStatus(m) ≜
    LET
        myServerId ≜ m.dest
    IN
    ∧ vServerStatus[myServerId] = StNormal
    ∧ vGView[myServerId] = m.gView
    ∧ vGVec[myServerId][m.sender.shardId] = m.lView
    ∧ IF m.deadline > vPeerCommitDeadline[myServerId][m.sender.shardId] THEN
           ∧ vPeerCommitDeadline[myServerId]' = [
                 vPeerCommitDeadline[myServerId]
                     EXCEPT ![m.sender.shardId] = m.deadline
             ]
      ELSE UNCHANGED ⟨vPeerCommitDeadline⟩
```

```
isCommitting(txn, deadlineQ) ≜
    LET quorum ≜ {msg ∈ deadlineQ : msg.entry.txnId = txn.txnId}
    IN   Cardinality(quorum) = Cardinality(txn.shards)


ReleaseSeqeuncer(serverId, currentTime) ≜
    LET
        serversInOneShard ≜ {s ∈ Servers : s.shardId = serverId.shardId}
        expireTxns ≜
            {msg ∈ vEarlyBuffer[serverId] :
                ∧ msg.deadline ≤ currentTime}
        sortedTxnList ≜ SetToSortSeq(expireTxns, Compare)
        committingStatus ≜
            [i ∈ 1 .. Len(sortedTxnList)
              ↦ isCommitting(sortedTxnList[i], vDeadlineQuorum[serverId])
            ]
        canReleaseTxnIndices ≜ {
            i ∈ 1 .. Len(sortedTxnList) :
               ∀ j ∈ 1 .. i : committingStatus[j] = TRUE}
    IN
    IF  Cardinality(canReleaseTxnIndices) = 0   Nothing to release
     THEN    UNCHANGED ⟨networkVars,
                  vLog, vEarlyBuffer, vLateBuffer, vDeadlineQuorum⟩
     ELSE
        LET
            releaseUpTo ≜ CHOOSE i ∈ canReleaseTxnIndices :
                              ∀ j ∈ canReleaseTxnIndices : j ≤ i
            releaseSeq ≜ SubSeq(sortedTxnList, 1, releaseUpTo)
            releaseTxns ≜  {releaseSeq[i] : i ∈ 1 .. Len(releaseSeq)}
        IN
        ∧ vEarlyBuffer' = [
            vEarlyBuffer EXCEPT ![serverId]
                = vEarlyBuffer[serverId] \ releaseTxns]
        ∧ vDeadlineQuorum' = [
            vDeadlineQuorum EXCEPT ![serverId]
                = {msg ∈ vDeadlineQuorum[serverId] :
                    ∀ txn ∈ releaseTxns : txn.txnId ≠ msg.txnId}
           ]
         Append to log
        ∧ vLog' = [vLog EXCEPT ![serverId] = vLog[serverId] ∘ releaseSeq]
        ∧  IF isLeader(serverId.replicaId, vLView[serverId]) THEN
                ∧ vLSyncPoint' = [vLSyncPoint EXCEPT  ![serverId] = Len(vLog'[serverId])]
            ELSE     UNCHANGED ⟨vLSyncPoint⟩
         Send fast-replies to coordinators
        ∧ Send({[
```

```
            mtype    ↦ MFastReply,
            sender   ↦ serverId,
            dest     ↦ sortedTxnList[i].txnId.coordId,
            gView    ↦ vGView[serverId],
            lView    ↦ vLView[serverId],
            txnId    ↦ sortedTxnList[i].txnId,
            hash     ↦ [
                           log ↦ vLog'[serverId],
                           cv  ↦ vCrashVector
                       ],
            logId    ↦ i
            ] : i ∈ (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
      ∧ Send({[
            mtype    ↦ MInterReplicaSync,
            lView    ↦ vLView[serverId],
            sender   ↦ serverId,
            dest     ↦ dstServerId,
            entries  ↦ vLog'[serverId]
        ] : dstServerId ∈ serversInOneShard})

ServerClockMove(serverId) ≜
  IF vServerClock[serverId] ≥ MaxTime THEN
     UNCHANGED ⟨networkVars, serverStateVars⟩
  ELSE
     ∧ vServerClock' = [
           vServerClock EXCEPT ![serverId] = vServerClock[serverId] + 1]
     ∧ IF vServerStatus[serverId] = StNormal THEN
           ∧ ReleaseSeqeuncer(serverId, vServerClock[serverId] + 1)
       ELSE
           UNCHANGED ⟨networkVars, vLog, vEarlyBuffer,
               vLateBuffer, vDeadlineQuorum⟩
     ∧ UNCHANGED ⟨vCrossShardConfirmQuorum,
           vServerStatus, vGView, vGVec, vLView, vLastNormView,
           vViewChange, vLSyncPoint, vLCommitPoint,
           vPeerCommitDeadline, vLSyncQuorum,
           vUUIDCounter, vCrashVector, vCrashVectorReps,
           vRecoveryReps, vServerProcessed⟩

CoordClockMove(coordId) ≜
  ∨ ∧ vCoordClock[coordId] ≥ MaxTime
    ∧ UNCHANGED ⟨vCoordClock⟩
```

$\lor \ \land \ vCoordClock[coordId] < MaxTime$
$\land \ vCoordClock' = [$
$vCoordClock \ \text{EXCEPT} \ ![coordId] = vCoordClock[coordId] + 1]$

$Init \triangleq$
$\land \ InitNetworkState$
$\land \ InitServerState$
$\land \ InitCoordState$
$\land \ InitConfigManagerState$
$\land \ ActionName = \langle \text{“Init”} \rangle$

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

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

$\lor \exists \, m \in messages :$
$\land \ m.mtype = MDeadlineNotification$
$\land \ vServerStatus[m.dest] = StNormal$
$\land \ m \notin vServerProcessed[m.dest]$
$\land \ vServerProcessed' = [vServerProcessed \ \text{EXCEPT} \ ![m.dest] =$
$vServerProcessed[m.dest] \cup \{m\}]$

$\wedge$ *HandleDeadlineNotification*(m)
$\wedge$ UNCHANGED $\langle$*networkVars*, *coordStateVars*, *configManagerStateVars*,
vLog, *vCrossShardConfirmQuorum*, *vLateBuffer*,
vServerStatus, *vGView*, *vGVec*,
vLView, *vServerClock*, *vLastNormView*,
vViewChange, *vLSyncPoint*, *vLCommitPoint*,
vPeerCommitDeadline, *vLSyncQuorum*,
vUUIDCounter, *vCrashVector*, *vCrashVectorReps*,
vRecoveryReps$\rangle$
$\wedge$ $ActionName' = \langle$ "HandleDeadlineNotification" $\rangle$

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

Some *Leader*(s) fail
$\vee \exists\, serverId \in Servers$:
$\wedge$ $vLView[serverId] < MaxViews$
$\wedge$ *isLeader*$(serverId.replicaId, vLView[serverId])$
$\wedge$ *StartLeaderFail*$(serverId)$
$\wedge$ 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*$\rangle$
$\wedge$ $ActionName' = \langle$ "StartLeaderFail" $\rangle$

configManager notices some *leader*(s) fail and launch view change
$\vee$ $\exists\, cmReplicaId \in Replicas$:
$\wedge$ *LaunchViewChange*$(cmReplicaId)$

$\land$ UNCHANGED $\langle coordStateVars,\ serverStateVars,\ configManagerStateVars\rangle$
$\land$ $ActionName' = \langle$"LaunchViewChange"$\rangle$

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

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

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

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

$\lor\ \exists\, m \in messages :$
$\quad \land\ m.mtype = MViewChange$
$\quad \land\ isCrashVectorValid(m)$
$\quad \land\ m \notin vServerProcessed[m.dest]$
$\quad \land\ vServerProcessed' = [vServerProcessed \text{ EXCEPT } ![m.dest] =$
$\qquad vServerProcessed[m.dest] \cup \{m\}]$
$\quad \land\ vServerStatus[m.dest] \neq StFailing$
$\quad \land\ HandleViewChange(m)$
$\quad \land\ \text{UNCHANGED } \langle coordStateVars,\ configManagerStateVars,$
$\qquad vGVec,\ vServerClock,\ vLSyncPoint,\ vLastNormView,$
$\qquad vLCommitPoint,\ vPeerCommitDeadline,\ vLSyncQuorum,$
$\qquad vUUIDCounter,\ vCrashVector,\ vCrashVectorReps,$
$\qquad vRecoveryReps\rangle$
$\quad \land\ ActionName' = \langle \text{“HandleViewChange”} \rangle$

$\lor\ \exists\, m \in messages :$
$\quad \land\ m.mtype = MCrossShardConfirm$
$\quad \land\ m \notin vServerProcessed[m.dest]$
$\quad \land\ vServerProcessed' = [vServerProcessed \text{ EXCEPT } ![m.dest] =$
$\qquad vServerProcessed[m.dest] \cup \{m\}]$
$\quad \land\ vServerStatus[m.dest] = StViewChange$
$\quad \land\ HandleCrossShardConfirm(m)$
$\quad \land\ \text{UNCHANGED } \langle coordStateVars,\ configManagerStateVars,$
$\qquad vGVec,\ vServerClock,\ vLSyncPoint,\ vLastNormView,$
$\qquad vLCommitPoint,\ vPeerCommitDeadline,\ vLSyncQuorum,$
$\qquad vUUIDCounter,\ vCrashVector,\ vCrashVectorReps,\ vRecoveryReps\rangle$
$\quad \land\ ActionName' = \langle \text{“HandleCrossShardConfirm”} \rangle$

$\lor\ \exists\, m \in messages :$
$\quad \land\ m.mtype = MStartView$
$\quad \land\ isCrashVectorValid(m)$
$\quad \land\ m \notin vServerProcessed[m.dest]$
$\quad \land\ vServerProcessed' = [vServerProcessed \text{ EXCEPT } ![m.dest] =$
$\qquad vServerProcessed[m.dest] \cup \{m\}]$
$\quad \land\ HandleStartView(m)$
$\quad \land\ \text{UNCHANGED } \langle coordStateVars,\ configManagerStateVars,$
$\qquad vServerClock,\ vLCommitPoint,\ vPeerCommitDeadline,$
$\qquad vUUIDCounter,\ vCrashVector\rangle$
$\quad \land\ ActionName' = \langle \text{“HandleStartView”} \rangle$

Failed server rejoin

$\lor \exists\, serverId \in Servers :$
$\quad \land\ vServerStatus[serverId] = StFailing$
$\quad \land\ vServerStatus' = [vServerStatus \text{ EXCEPT } ![serverId] = StRecovering]$
$\quad \land\ ResetServerState(serverId)$
$\quad \land\ StartServerRecovery(serverId)$

$\land$ UNCHANGED $\langle networkVars,\ coordStateVars,\ coordStateVars\rangle$
$\land$ $ActionName' = \langle$ "StartReplicaRecovery" $\rangle$

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

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

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

$vUUIDCounter$, $vCrashVectorReps$, $vRecoveryReps\rangle$
$\land\ ActionName' = \langle$"HandleRecoveryReq"$\rangle$

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

    Clock Move
    ∨ ∃ serverId ∈ Servers :
        ∧ ServerClockMove(serverId)
        ∧ UNCHANGED ⟨coordStateVars, configManagerStateVars⟩
        ∧ ActionName' = ⟨"ServerClockMove"⟩

    ∨ ∃ coordId ∈ Coords :
        ∧ CoordClockMove(coordId)
        ∧ UNCHANGED ⟨networkVars, serverStateVars, configManagerStateVars,
            vCoordTxns, vCoordProcessed⟩
        ∧ ActionName' = ⟨"CoordClockMove"⟩

Spec ≜ Init ∧ □[Next]⟨networkVars,
                      serverStateVars, coordStateVars, configManagerStateVars,
                      ActionName⟩


ShardRecovered(shardId, lViewID) ≜
    LET
        serversInOneShard ≜ {s ∈ Servers : s.shardId = shardId}
        leaderServer ≜ [
            replicaId ↦ LeaderID(lViewID),
            shardId ↦ shardId
        ]
    IN
    ∧ ∃ RM ∈ SUBSET (serversInOneShard) :
        ∧ Cardinality(RM) ≥ QuorumSize
        ∧ leaderServer ∈ RM
        ∧ ∀ r ∈ RM : vServerStatus[r]    = StNormal
        ∧ ∀ r ∈ RM : vLastNormView[r] ≥ lViewID


CommittedInView(v, shardId, txnId) ≜
    LET
        serversInOneShard ≜ {s ∈ Servers : s.shardId = shardId}
        leaderServer ≜ [
            replicaId ↦ LeaderID(v),
            shardId ↦ shardId
        ]
```

$$
\begin{array}{l}
\quad replySet \triangleq \{ \\
\quad\quad m \in messages : \land \ \lor \ m.mtype = MFastReply \\
\quad\quad\quad\quad\quad\quad\quad\quad\ \lor \ m.mtype = MSlowReply \\
\quad\quad\quad\quad\quad\quad\quad\ \land \ m.txnId = txnId \\
\quad\quad\quad\quad\quad\quad\quad\ \land \ m.sender \in serversInOneShard \\
\quad\quad\quad\quad\quad\quad\quad\ \land \ m.lView = v \\
\quad \} \\
\text{IN} \\
\text{IF } \forall\, reply \in replySet : \\
\quad \lor \ reply.mtype \neq MFastReply \\
\quad \lor \ reply.sender \neq leaderServer \\
\text{THEN} \quad \text{No leader's fast } reply \rightarrow \text{ This } txn \text{ is not committed} \\
\quad \text{FALSE} \\
\text{ELSE} \\
\quad \text{LET} \\
\quad\quad leaderReply \triangleq \text{CHOOSE } reply \in replySet : \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \land \ reply.mtype = MFastReply \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \land \ reply.sender = leaderServer \\
\quad \text{IN} \\
\quad\quad \text{Committed in Fast Path} \\
\quad\quad \lor \ \exists\, fastQuorum \in \text{SUBSET } replySet : \\
\quad\quad\quad\quad \land \ leaderReply \in fastQuorum \\
\quad\quad\quad\quad \land \ Cardinality(fastQuorum) = FastQuorumSize \\
\quad\quad\quad\quad\quad \text{All replies have the same hash (or it is a slow reply)} \\
\quad\quad\quad\quad \land \ \forall\, reply \in fastQuorum : \\
\quad\quad\quad\quad\quad\quad \lor \ \land \ reply.mtype = MFastReply \\
\quad\quad\quad\quad\quad\quad\quad\quad \land \ reply.hash = leaderReply.hash \\
\quad\quad\quad\quad\quad\quad\quad \text{Slow Reply can be used as fast reply} \\
\quad\quad\quad\quad\quad\quad \lor \ reply.mtype = MSlowReply \\
\quad\quad \text{Committed in Slow Path} \\
\quad\quad \lor \ \exists\, slowQuorum \in \text{SUBSET } \ replySet : \\
\quad\quad\quad\quad \land \ leaderReply \in slowQuorum \\
\quad\quad\quad\quad \land \ Cardinality(slowQuorum) = QuorumSize \\
\quad\quad\quad\quad \land \ \forall\, reply \in slowQuorum \setminus \{leaderReply\} : \\
\quad\quad\quad\quad\quad\quad reply.mtype = MSlowReply
\end{array}
$$

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
∀ txnId1, txnId2 ∈ allTxns :
  IF txnId1 = txnId2 THEN TRUE
   ELSE
     ∀ v ∈ 1 .. MaxViews :
       ∀ shardId1, shardId2 ∈ Shards :
         IF shardId1 = shardId2 THEN TRUE
          ELSE
            IF   ∧ CommittedInView(v, shardId1, txnId1)
                 ∧ CommittedInView(v, shardId1, txnId2)
                 ∧ CommittedInView(v, shardId2, txnId1)
                 ∧ 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
```