

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*]

$Replicas \triangleq 0 \dots 2$

$ReplicaOrder \triangleq \langle 0, 1, 2 \rangle$

$Shards \triangleq 0 \dots 2$

$Coords \triangleq 0 \dots 1$

$LatencyBounds \triangleq [c \in Coords \mapsto 1]$

ASSUME $IsFiniteSet(Replicas)$

ASSUME $IsFiniteSet(Shards)$

ASSUME $ReplicaOrder \in Seq(Replicas)$

$Servers \triangleq \{$

$[$

$replicaId \mapsto e[1],$

$shardId \mapsto e[2]$

$] : e \in Replicas \times Shards$

$\}$

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

Constants

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

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

Server Status

$$\begin{aligned}
\text{StNormal} &\triangleq 1 \\
\text{StViewChange} &\triangleq 2 \\
\text{StCrossShardSyncing} &\triangleq 3 \\
\text{StRecovering} &\triangleq 4 \\
\text{StFailing} &\triangleq 5
\end{aligned}$$

Message Types

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

The following messages are mainly for view change within each sharding group

$$\begin{aligned}
\text{MViewChangeReq} &\triangleq 7 \quad \text{Sent by config manager when leader/sequencer failure detected} \\
\text{MViewChange} &\triangleq 8 \quad \text{Sent to ACK view change} \\
\text{MStartView} &\triangleq 9 \quad \text{Sent by new leader to start view}
\end{aligned}$$

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

$$\begin{aligned}
\text{MLocalSyncStatus} &\triangleq 10 \quad \text{Sent by the leader to ensure log durability} \\
\text{MLocalCommit} &\triangleq 11 \quad \text{Sent by followers as ACK}
\end{aligned}$$

The following messages are used for periodic sync across sharding groups

This is an optional optimization to enable fast recovery

$$\text{MPeerShardCommitStatus} \triangleq 12$$

The following messages are mainly used for server recovery

$$\begin{aligned}
\text{MCrashVectorReq} &\triangleq 13 \\
\text{MCrashVectorRep} &\triangleq 14 \\
\text{MRecoveryReq} &\triangleq 15 \\
\text{MRecoveryRep} &\triangleq 16 \\
\text{MStartViewReq} &\triangleq 17
\end{aligned}$$

$MCrossShardVerifyReq \triangleq 18$
 $MCrossShardVerifyRep \triangleq 19$

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 $\langle replicaId, shardId \rangle$. $TxnID$ uniquely identifies one request on one server. But across replicas, the same $TxnID$ may have different timestamps (the leader may modify the timestamp to make the request eligible to enter the early-buffer) so $\langle timestamp, txnId \rangle$ uniquely identifies 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,
  timestamp ↦ timestamp
]

```

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

```

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

```

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

```

InterReplicaSync = [
  mtype      ↦ MInterReplicaSync,
  lView      ↦ 0...y
  sender     ↦ src ∈ Servers,
  dest       ↦ dst ∈ Servers,
  entries    ↦ [LogEntry...]
]

```

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      ↦ MFastReply,
  sender     ↦ src ∈ Servers,
  dest       ↦ dst ∈ Coords,
  gView      ↦ 0...x
  lView      ↦ 0...x
  txnId      ↦ txnId

```

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

```

hash        ↦ [ entries ↦ log entries so far cv ↦ crashVector ]
timestamp   ↦ i ∈ (1..MaxTime + MaxBound), logId      ↦ n ∈ (1..)
]

```

```

SlowReply = [
  mtype      ↦ MSlowReply,
  sender     ↦ src ∈ Servers,
  dest       ↦ c ∈ Coords,
  gView      ↦ 0...x
  lView      ↦ 0...x
  txnId      ↦ txnId
  logId      ↦ n ∈ (1..)
]

```

```

ViewChangeReq = [
  mtype ↦ MViewChangeReq,
  sender ↦ src ∈ Replicas, (by configManager)
  dest ↦ dst ∈ Servers,
  gView ↦ 0..x
  gVec ↦ the lViews for each shard
]

```

```

ViewChange = [
  mtype      ↦ MViewChange,
  sender     ↦ src ∈ Servers,
  dest       ↦ dst ∈ Servers,
  gView      ↦ 0..x
  gVec       ↦ the lViews for each shard
  lView      ↦ 0..x
  lastNormal ↦ v ∈ ViewIDs,
  lSyncPoint ↦ 0..

```

```

    entries    ↦  $l \in vLogs[1 \dots n]$ ,
    cv         ↦ crash vector
]

```

Tell the leaders in other sharding groups my *syncPoint*

```

MCrossShardVerifyReq = [
    mtype      ↦ MCrossShardVerifyReq,
    sender     ↦  $src \in Servers$ ,
    dest       ↦  $dst \in Servers$ ,
    lView      ↦  $0 \dots x$ 
    gView      ↦  $0 \dots x$ 
    syncedDdl  ↦ The largest timestamp of the synced entries
]

```

Reply the entries to the other leaders. These *entries'* log positions are beyond the *syncPoint* of the receiving leader, so that the receiving leader can verify whether it needs timestamp agreement for the *txn*, or even misses the *txn*

```

MCrossShardVerifyRep = [
    mtype      ↦ MCrossShardVerifyRep,
    sender     ↦  $src \in Servers$ ,
    dest       ↦  $dst \in Servers$ ,
    lView      ↦  $0 \dots x$ 
    gView      ↦  $0 \dots x$ 
    entries    ↦  $l \in vLogs[1 \dots n]$ 
]

```

```

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

```

```

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

```

```

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

```

```

RecoveryReq = [
  mtype      ↦ MRecoveryReq,
  sender     ↦ src ∈ Servers,
  dest       ↦ dst ∈ Servers,
  cv         ↦ vector of counters
]

```

```

RecoveryRep = [
  mtype ↦ MRecoveryRep,
  sender ↦ src ∈ Servers,
  dest ↦ dst ∈ Servers,
  gView ↦ 0 .. x
  lView ↦ 0 .. x
  cv    ↦ vector of counters
]

```

```

StartViewReq = [
  mtype      ↦ MStartViewReq,
  sender     ↦ src ∈ Servers,
  dest       ↦ dst ∈ Servers,
  lView      ↦ 0 .. x
  cv         ↦ vector of counters
]

```

Follower reports to its leader

```

LocalSyncStatus = [
  mtype      ↦ MLocalSyncStatus,
  sender     ↦ src ∈ Servers,
  dest       ↦ dst ∈ Servers,
  lView      ↦ 0 ... x
  lSyncPoint ↦ 1 ..
  cv         ↦ vector of counters
]

```

Leader notifies its followers

```

LocalCommit = [
  mtype      ↦ MLocalCommit,
  sender     ↦ src ∈ Servers,
  dest       ↦ dst ∈ Servers,
  lView      ↦ 0 ... x
  entries     ↦ log entries
  lCommitPoint ↦ n ∈ (1...)
]

```

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)

\ * maybe obsolete

```

PeerShardCommitStatus = [
  mtype      ↦ MPeerShardCommitStatus,

```

```

    sender     $\mapsto src \in Servers,$ 
    dest       $\mapsto dst \in Servers,$ 
    gView      $\mapsto 0 \dots x$ 
    timestamp  $\mapsto$  the largest committed timestamp
]

```

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  $\mapsto 0 \dots x$ 
    gView  $\mapsto 0 \dots x$ 
    gVec  $\mapsto [shardId \mapsto lView]$ 
]

```

```

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

```

```

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

```

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 timestamp (*s* + *l*)

| | |
|---|--|
| <i>vEarlyBuffer</i> , | |
| The buffer to hold txns on followers because these txns come too late and cannot enter early-buffer | |
| <i>vLateBuffer</i> , | |
| Each leader server has a data structure of <i>TimestampQuorum</i> to collect the timestamps from other servers for agreement | |
| <i>vTimestampQuorum</i> , | |
| One of <i>StNormal</i> , <i>StViewChange</i> , <i>StFailing</i> , <i>StCrossShardSyncing</i> , <i>StRecovering</i> | |
| <i>vServerStatus</i> , | |
| Global views of each server | |
| <i>vGView</i> , | |
| The g-vecs of each server | |
| <i>vGVec</i> , | |
| Local views of each server | |
| <i>vLView</i> , | |
| Current Time of the server | |
| <i>vServerClock</i> , | |
| Last <i>lView</i> in which this server had <i>StNormal</i> status | |
| <i>vLastNormView</i> , | |
| Used for collecting view change votes | |
| <i>vViewChange</i> , | |
| Used for collecting <i>CrossShardVerify</i> replies. After the leader have recovered their logs for its own shard, they need verify from the other <i>shards</i> to ensure the recovered logs satisfy strict serializability, <i>i.e.</i> , every <i>log</i> has commonly-agreed timestamps across sharding groups. | |
| <i>vCrossShardVerifyReps</i> , | |
| <i>vLSyncPoint</i> indicates to which the server state (<i>vLog</i>) is consistent with the leader. | |
| <i>vLSyncPoint</i> , | |
| <i>vLCommitPoint</i> indicates that the <i>log</i> entries before this point has been locally committed, <i>i.e.</i> , replicated to majority in this sharding groups. So followers can safely execute the logged txns | |
| <i>vLCommitPoint</i> , | |
| <i>vLSyncQuorum</i> is used by each leader to collect the <i>LocalSyncStatus</i> messages from servers in the same sharding group | |
| <i>vLSyncQuorum</i> , | |
| Locally unique string (for <i>CrashVectorReq</i>) | |
| <i>vUUIDCounter</i> , | |
| <i>CrashVector</i> , initialized as all-zero vector | |
| <i>vCrashVector</i> , | |
| <i>vCrashVectorReps</i> , | |
| <i>vRecoveryReps</i> | |

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 $gVec$)
 $vCMGInfo$,
 $vCMPPrepareGInfo$,
 Config Manager: quorum of $CMPrepareReplies$
 $vCMPPrepareReps$,
 $vCMPProcessed$

VARIABLES $ActionName$

$networkVars \triangleq \langle messages \rangle$

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

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

$configManagerStateVars \triangleq \langle vCMStatus, vCMView, vCMGInfo,$
 $vCMPPrepareGInfo, vCMPPrepareReps,$
 $vCMPProcessed \rangle$

$InitNetworkState \triangleq messages = \{\}$

$InitServerState \triangleq$

$$\begin{aligned}
& \wedge vServerProcessed = [serverId \in Servers \mapsto \{\}] \\
& \wedge vLog = [serverId \in Servers \mapsto \langle \rangle] \\
& \wedge vEarlyBuffer = [serverId \in Servers \mapsto \{\}] \\
& \wedge vLateBuffer = [serverId \in Servers \mapsto \{\}] \\
& \wedge vTimestampQuorum = [serverId \in Servers \mapsto \{\}] \\
& \wedge vCrossShardVerifyReps = [serverId \in Servers \mapsto \{\}] \\
& \wedge vServerStatus = [serverId \in Servers \mapsto StNormal] \\
& \wedge vGView = [serverId \in Servers \mapsto 0] \\
& \wedge vGVec = [\\
& \quad serverId \in Servers \mapsto [\\
& \quad \quad shardId \in Shards \mapsto 0 \\
& \quad] \\
&] \\
& \wedge vLView = [serverId \in Servers \mapsto 0] \\
& \wedge vServerClock = [serverId \in Servers \mapsto 1] \\
& \wedge vLastNormView = [serverId \in Servers \mapsto 0] \\
& \wedge vViewChange = [serverId \in Servers \mapsto \{\}] \\
& \wedge vLSyncPoint = [serverId \in Servers \mapsto 0] \\
& \wedge vLCommitPoint = [serverId \in Servers \mapsto 0] \\
& \wedge vLSyncQuorum = [serverId \in Servers \mapsto \{\}] \\
& \wedge vUUIDCounter = [serverId \in Servers \mapsto 0] \\
& \wedge vCrashVector = [\\
& \quad serverId \in Servers \mapsto [\\
& \quad \quad rr \in Replicas \mapsto 0 \\
& \quad] \\
&] \\
& \wedge vCrashVectorReps = [serverId \in Servers \mapsto \{\}] \\
& \wedge vRecoveryReps = [serverId \in Servers \mapsto \{\}]
\end{aligned}$$

$$\begin{aligned}
InitCoordState & \triangleq \\
& \wedge vCoordProcessed = [c \in Coords \mapsto \{\}] \\
& \wedge vCoordClock = [c \in Coords \mapsto 1] \\
& \wedge vCoordTrns = [c \in Coords \mapsto \{\}]
\end{aligned}$$

$$\begin{aligned}
InitConfigManagerState & \triangleq \\
& \wedge vCMStatus = [\\
& \quad replicaId \in Replicas \mapsto StNormal \\
&] \\
& \wedge vCMView = [\\
& \quad replicaId \in Replicas \mapsto 0 \\
&] \\
& \wedge vCMGInfo = [\\
& \quad replicaId \in Replicas \mapsto [\\
& \quad \quad gView \mapsto 0, \\
& \quad] \\
&]
\end{aligned}$$

```

      gVec    ↦ [shardId ∈ Shards ↦ 0]
    ]
  ]
  ∧ vCMPPrepareGInfo = [
    replicaId ∈ Replicas ↦ [
      gView    ↦ 0,
      gVec     ↦ [shardId ∈ Shards ↦ 0]
    ]
  ]
  ∧ vCMPPrepareReps = [
    replicaId ∈ Replicas ↦ {}
  ]
  ∧ vCMPProcessed = [
    replicaId ∈ Replicas ↦ {}
  ]
]

```

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

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

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

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

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

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

$IsInjective(s) \triangleq$

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.

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

$Remove(s, e) \triangleq$

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 $\langle \rangle$ 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 $\wedge gv1 = gv2$
 $\wedge lv1 > lv2$
 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

$LastAppendedTimestamp(Log) \triangleq$ IF $Len(Log) = 0$ THEN 0
 ELSE $Tail(Log).timestamp$

$CoordSubmitTxn(c) \triangleq$
 $\wedge vCoordClock[c] < MaxTime$
 $\wedge Cardinality(vCoordTxns[c]) < MaxReqNum$
 \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*
 $shards \mapsto Shards,$
 $st \mapsto vCoordClock[c],$
 $bound \mapsto LatencyBounds[c],$

$$\begin{aligned}
& sender \mapsto c, \\
& dest \mapsto serverId \\
&] : serverId \in Servers\} \\
& \wedge vCoordClock' = [vCoordClock \text{ EXCEPT } ![c] = vCoordClock[c] + 1] \\
& \wedge vCoordTxns' = [vCoordTxns \text{ EXCEPT } ![c] = vCoordTxns[c] \cup \{txnId\}]
\end{aligned}$$

$$HandleTxn(m) \triangleq$$

LET

$$myServerId \triangleq m.dest$$

$$newLog \triangleq [$$

$$mtype \mapsto MLogEntry,$$

$$txnId \mapsto m.txnId,$$

$$command \mapsto m.command,$$

$$shards \mapsto m.shards,$$

$$timestamp \mapsto Max(LastAppendedTimestamp(vLog[myServerId]), m.st + m.bound)$$

$$]$$

$$serversInOneReplica \triangleq \{s \in Servers : s.replicaId = myServerId.replicaId\}$$

IN

$$\vee \wedge isLeader(myServerId.replicaId, vLView[myServerId])$$

$$\wedge vEarlyBuffer' = [$$

$$vEarlyBuffer \text{ EXCEPT } ![myServerId]$$

$$= vEarlyBuffer[myServerId] \cup \{newLog\}]$$

Broadcast timestamp notifications to other shards

$$\wedge Send(\{[$$

$$mtype \mapsto MTimestampNotification,$$

$$gView \mapsto vGView[myServerId],$$

$$lView \mapsto vLView[myServerId],$$

$$sender \mapsto myServerId,$$

$$dest \mapsto dstServerId,$$

$$entry \mapsto newLog$$

$$] : dstServerId \in serversInOneReplica\})$$

$$\wedge \text{UNCHANGED } \langle vLateBuffer \rangle$$

$$\vee \wedge \neg isLeader(myServerId.replicaId, vLView[myServerId])$$

$$\wedge \vee \wedge newLog.timestamp = (m.st + m.bound)$$

$$\wedge vEarlyBuffer' = [$$

$$vEarlyBuffer \text{ EXCEPT } ![myServerId]$$

$$= vEarlyBuffer[myServerId] \cup \{newLog\}$$

$$]$$

$$\wedge \text{UNCHANGED } \langle vLateBuffer \rangle$$

$$\vee \wedge \neg (newLog.timestamp = (m.st + m.bound))$$

$$\wedge vLateBuffer' = [$$

$$vLateBuffer \text{ EXCEPT } ![myServerId]$$

$$= vLateBuffer[myServerId] \cup \{newLog\}$$

$$]$$

$\wedge \text{ UNCHANGED } \langle vEarlyBuffer \rangle$
 $\wedge \text{ UNCHANGED } \langle networkVars \rangle$

$HandleTimestampNotification(m) \triangleq$

LET

$myServerId \triangleq m.dest$
 $quorum \triangleq \{$
 $msg \in vTimestampQuorum[myServerId]$
 $: \wedge msg.entry.txnId = m.entry.txnId$
 $\wedge msg.gView = m.gView$
 $\wedge m.gView = vGView[myServerId]$
 $\} \cup \{m\}$

IN

Only leader does timestamp agreement

$\wedge vGView[myServerId] = m.gView$
 $\wedge vGVec[myServerId][m.sender.shardId] = m.lView$
 $\wedge isLeader(myServerId.replicaId, vLView[myServerId])$
 $\wedge vTimestampQuorum' = [$
 $vTimestampQuorum \text{ EXCEPT } ![myServerId]$
 $= vTimestampQuorum[myServerId] \cup \{m\}$
 $]$

$\wedge \text{ IF } Cardinality(quorum) = Cardinality(m.entry.shards)$

THEN

Timestamp quorum established : Update the timestamp of the txn in Sequencer

LET

$maxTimestampTxn \triangleq$
 $\text{CHOOSE } x \in quorum :$
 $\forall y \in quorum :$
 $y.entry.timestamp \leq x.entry.timestamp$
 $sequencingTxn \triangleq$
 $\text{CHOOSE } x \in vEarlyBuffer[myServerId] :$
 $x.txnId = m.entry.txnId$

IN

$\text{IF } maxTimestampTxn.entry.timestamp > sequencingTxn.timestamp$

THEN

$vEarlyBuffer' = [vEarlyBuffer \text{ EXCEPT } ![myServerId]$
 $= (vEarlyBuffer[myServerId] \setminus \{sequencingTxn\}) \cup \{maxTimestampTxn.entry\}]$

ELSE UNCHANGED $\langle vEarlyBuffer \rangle$

ELSE

Timestamp quorum not sufficient so far: do not take further actions

UNCHANGED $\langle vEarlyBuffer \rangle$

$HandleInterReplicaSync(m) \triangleq$
 $\wedge m.lView = vLView[m.dest]$
 Even if m 's $crashVector$ is newer (larger value), we do not accept it. The consistency of $crashVector$ will finally be solved during viewchange
 $\wedge m.crashVector[m.sender] = vCrashVector[m.sender]$
 $\wedge \neg isLeader(m.dest.replicaId, vLView[m.dest])$
 $\wedge LET$
 $myServerId \triangleq m.dest$
 $syncedTxnIds \triangleq \{m.entries[i].txnId : i \in 1 \dots Len(m.entries)\}$
 $currentSyncPoint \triangleq Len(vLSyncPoint[myServerId])$
 IN
 $\vee \wedge currentSyncPoint < Len(m.entries)$
 $\wedge vLog' = [vLog \text{ EXCEPT } ![myServerId] = m.entries]$
 Kick synced entries out of $earlyBuffer$
 $\wedge vEarlyBuffer' = [$
 $\quad vEarlyBuffer \text{ EXCEPT } ![myServerId]$
 $\quad = \{msg \in vEarlyBuffer[myServerId] :$
 $\quad \quad msg.txnId \notin syncedTxnIds\}$
 $\quad]$
 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 unnecessary transmission in practice.
 $\wedge vLateBuffer' = [$
 $\quad vLateBuffer \text{ EXCEPT } ![myServerId]$
 $\quad = \{msg \in vLateBuffer[myServerId] :$
 $\quad \quad msg.txnId \notin syncedTxnIds\}$
 $\quad]$
 Kick synced entries out of timestamp quorum. These txns have been synced, no need to record in *TimestampQuorum*
 $\wedge vTimestampQuorum' = [$
 $\quad vTimestampQuorum \text{ EXCEPT } ![myServerId]$
 $\quad = \{msg \in vTimestampQuorum[myServerId] :$
 $\quad \quad msg.txnId \notin syncedTxnIds\}$
 $\quad]$
 $\wedge vLSyncPoint' = [$
 $\quad vLSyncPoint \text{ EXCEPT } ![myServerId] = Len(m.entries)]$
 Send slow-replies to coordinators
 $\wedge Send(\{[$
 $\quad mtype \mapsto MSlowReply,$
 $\quad sender \mapsto myServerId,$
 $\quad dest \mapsto m.entries[i].txnId.coordId,$
 $\quad gView \mapsto vGView[myServerId],$
 $\quad lView \mapsto vLView[myServerId],$

$$\begin{aligned}
& \text{txnId} \mapsto m.\text{entries}[i].\text{txnId}, \\
& \text{logId} \mapsto i \\
&] : i \in (\text{currentSyncPoint} + 1) \dots \text{Len}(m.\text{entries})\} \\
\vee \quad & \wedge \text{currentSyncPoint} \geq \text{Len}(m.\text{entries}) \\
& \text{Noting new to sync} \\
& \wedge \text{UNCHANGED } \langle \text{networkVars}, v\text{Log}, v\text{EarlyBuffer}, \\
& \quad v\text{LateBuffer}, v\text{TimestampQuorum}, v\text{LSyncPoint} \rangle
\end{aligned}$$

$\text{StartLeaderFail}(\text{serverId}) \triangleq$

This leader fails

LET

$$\begin{aligned}
& \text{serversInOneShard} \triangleq \{ \\
& \quad s \in \text{Servers} : s.\text{shardId} = \text{serverId}.\text{shardId} \\
& \} \\
& \text{aliveReplicas} \triangleq \{ \\
& \quad s \in \text{serversInOneShard} : \quad \wedge v\text{ServerStatus}[s] = \text{StNormal} \\
& \quad \quad \quad \quad \quad \quad \quad \wedge s \neq \text{serverId} \\
& \}
\end{aligned}$$

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 $\text{Cardinality}(\text{aliveReplicas}) > \text{QuorumSize}$ THEN

$v\text{ServerStatus}' = [v\text{ServerStatus} \text{ EXCEPT } ![\text{serverId}] = \text{StFailing}]$

ELSE UNCHANGED $\langle v\text{ServerStatus} \rangle$

$\text{DetectLeaderFail}(\text{cmReplicaId}) \triangleq$

$\exists \text{shardId} \in \text{Shards} :$

LET

$$\begin{aligned}
& l\text{View} \triangleq v\text{CMGInfo}[\text{cmReplicaId}].g\text{Vec}[\text{shardId}] \\
& \text{leaderId} \triangleq \text{LeaderID}(l\text{View}) \\
& \text{serverId} \triangleq [\\
& \quad \text{replicaId} \mapsto \text{leaderId}, \\
& \quad \text{shardId} \mapsto \text{shardId} \\
&]
\end{aligned}$$

IN

$v\text{ServerStatus}[\text{serverId}] = \text{StFailing}$

$\text{SelectProperLView}(\text{currentView}, \text{shardId}) \triangleq$

LET

$$\begin{aligned}
& \text{aliveReplicaId} \triangleq \text{CHOOSE } \text{replicaId} \in \text{Replicas} : \\
& \quad v\text{ServerStatus}[\text{shardId}][\text{replicaId}] = \text{StNormal}
\end{aligned}$$

IN

Ensure 1 the new view is larger than currentView

** (2) its corresponding leader happens to be the selected aliveReplicaId*

$$\begin{aligned}
& (currentView \div Cardinality(Replicas) + 1) * Cardinality(Replicas) + aliveReplicaId \\
PrepareViewChange(cmReplicaId) & \triangleq \\
\text{LET} & \\
& newGVec \triangleq [\\
& \quad shardId \in Shards \mapsto \\
& \quad \quad SelectProperLView(vCMGInfo[cmReplicaId].gVec[shardId], shardId) \\
&] \\
\text{IN} & \\
\wedge \quad vCMPPrepareGInfo' & = [vCMPPrepareGInfo \text{ EXCEPT } ![cmReplicaId] = \\
& [\\
& \quad gView \mapsto vCMGInfo[cmReplicaId].gView + 1, \\
& \quad gVec \mapsto newGVec \\
&] \\
&] \\
\wedge \quad Send(\{[& \\
& \quad mtype \mapsto MCMPrepare, \\
& \quad sender \mapsto cmReplicaId, \\
& \quad dest \mapsto dstRid, \\
& \quad cView \mapsto vCMView[cmReplicaId], \\
& \quad gView \mapsto vCMPPrepareGInfo'[cmReplicaId].gView, \\
& \quad gVec \mapsto newGVec \\
&] : dstRid \in Replicas\}) \\
\\
LaunchViewChange(cmReplicaId) & \triangleq \\
\text{IF} \quad \wedge \quad isLeader(cmReplicaId, vCMView[cmReplicaId]) & \\
& \wedge \quad DetectLeaderFail(cmReplicaId) \\
\text{THEN} & \\
& \quad PrepareViewChange(cmReplicaId) \\
\text{ELSE} & \\
& \quad \text{UNCHANGED } \langle networkVars \rangle \\
\\
HandleCMPPrepare(m) & \triangleq \\
& \wedge \quad m.cView = vCMView[m.dest] \\
& \wedge \quad m.gView > vCMGInfo[m.dest].gView \\
& \wedge \quad vCMPPrepareGInfo' = [vCMPPrepareGInfo \text{ EXCEPT } ![m.dest] = \\
& \quad [\\
& \quad \quad gView \mapsto m.gView, \\
& \quad \quad gVec \mapsto m.gVec \\
& \quad] \\
&] \\
& \wedge \quad Send(\{[
\end{aligned}$$

```

    mtype  ↦ MCMPrepareReply,
    sender  ↦ m.dest,
    dest    ↦ m.src,
    cView   ↦ m.cView,
    gView   ↦ m.gView
  })

```

```

HandleCMPrepareReply(m)  $\triangleq$ 
   $\wedge$  m.cView = vCMView[m.dest]
   $\wedge$  isLeader(m.dest, vCMView[m.dest])
   $\wedge$  m.gView = vCMPrepareGInfo[m.dest].gView
   $\wedge$  vCMPrepareReps' = [vCMPrepareReps EXCEPT ![m.dest] =
    vCMPrepareReps[m.dest]  $\cup$  {m}]
  ]
   $\wedge$  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
     $\wedge$  vCMGInfo' = [vCMGInfo EXCEPT ![m.dest] =
      vCMPrepareGInfo[m.dest]]
    ]
    notify other follower CM, so that they can catch up with the leader
     $\wedge$  Send([
      mtype  ↦ MCMCommit,
      sender  ↦ m.dest,
      dest    ↦ rid,
      cView   ↦ m.cView,
      gView   ↦ m.gView
    ] : rid  $\in$  {r  $\in$  Replicas : r  $\neq$  m.dest})
    start view change, broadcast view change request to every server
     $\wedge$  Send([
      mtype  ↦ MViewChangeReq,
      sender  ↦ m.dest,
      dest    ↦ serverId,
      gView   ↦ vCMGInfo'[m.dest].gView,
      gVec    ↦ vCMGInfo'[m.dest].gVec
    ] : serverId  $\in$  Servers)
  ELSE
    UNCHANGED (networkVars, vCMGInfo)

```

```

HandleCMCommit(m)  $\triangleq$ 
   $\wedge$  m.cView = vCMView[m.dest]
   $\wedge$   $\neg$ isLeader(m.dest, vCMView[m.dest])
   $\wedge$  m.gView = vCMPrepareGInfo[m.dest].gView

```

$$\wedge \text{ } vCMGInfo' = [vCMGInfo \text{ EXCEPT } ![m.dest] = \\ vCMPPrepareGInfo[m.dest]]$$

HandleViewChangeReq(*m*) \triangleq

LET

myServerId \triangleq *m.dest*

myLeader \triangleq CHOOSE *s* \in *Servers* :

\wedge *s.replicaId* = *LeaderID*(*m.gVec*[*myServerId.shardId*])

\wedge *s.shardId* = *myServerId.shardId*

IN

If the *msg*'s view is lower, ignore

\wedge *vGView*[*myServerId*] < *m.gView*

\wedge IF *vServerStatus*[*myServerId*] = *StNormal* THEN

\wedge *vServerStatus'* = [*vServerStatus* EXCEPT ![*myServerId*] = *StViewChange*]

\wedge *vLastNormView'* = [*vLastNormView* EXCEPT ![*myServerId*] = *vLView*[*myServerId*]]

ELSE UNCHANGED $\langle vServerStatus, vLastNormView \rangle$

\wedge *vGView'* = [

vGView EXCEPT ![*myServerId*] = *m.vGView*

]

\wedge *vGVec'* = [

vGVec EXCEPT ![*myServerId*] = *m.gVec*

]

\wedge *vLView'* = [

vLView EXCEPT ![*myServerId*] = *m.gVec*[*myServerId.shardId*]

]

Clear early buffer,

\wedge *vEarlyBuffer'* = [

vEarlyBuffer EXCEPT ![*myServerId*] = {}

]

Clear late buffer

\wedge *vLateBuffer'* = [

vLateBuffer EXCEPT ![*myServerId*] = {}

]

Clear timestamp quorum

\wedge *vTimestampQuorum'* = [

vTimestampQuorum EXCEPT ![*myServerId*] = {}

]

Clear *vCrossShardVerifyReps*

\wedge *vCrossShardVerifyReps'* = [

serverId \in *Servers* \mapsto {}

]

Send *ViewChange* to the *myLeader*

\wedge *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)  $\triangleq$ 
   $\vee$  a.timestamp < b.timestamp
   $\vee$   $\wedge$  a.timestamp = b.timestamp
     $\wedge$  a.txnId.coordId < b.txnId.coordId
   $\vee$   $\wedge$  a.timestamp = b.timestamp
     $\wedge$  a.txnId.coordId = b.txnId.coordId
     $\wedge$  a.txnId.rId < b.txnId.rId

isCrashVectorValid(m)  $\triangleq$ 
   $\wedge$   $\forall rr \in Replicas : vCrashVector[m.dest][rr] \leq m.cv[rr]$ 
   $\wedge$  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 :  $\exists e \in s$  :
       $\wedge$  e.timestamp = entry.timestamp
       $\wedge$  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 :
       $\forall sp \in lSyncPoints : sp \leq vc.lSyncPoint$ 

```

$$\begin{aligned}
& \text{syncedLogSeq} \triangleq \text{SubSeq}(\text{largestLSyncPointVC.entries}, 1, \text{largestLSyncPointVC.lSyncPoint}) \\
& \text{timestampBoundary} \triangleq \text{IF } \text{largestLSyncPointVC.lSyncPoint} = 0 \text{ THEN } 0 \\
& \quad \text{ELSE } \text{syncedLogSeq}[\text{largestLSyncPointVC.lSyncPoint}].\text{timestamp} \\
& \text{logSets} \triangleq \{\text{SeqToSet}(m.\text{entries}) : m \in \text{refinedQuorum}\} \\
& \text{allLogs} \triangleq \text{UNION } \text{logSets} \\
& \text{allUnSyncedLogs} \triangleq \{\text{entry} \in \text{allLogs} : \text{entry.timestamp} > \text{timestampBoundary}\} \\
& \text{unSyncedLogs} \triangleq \{\text{entry} \in \text{allUnSyncedLogs} : \\
& \quad \text{CountVotes}(\text{entry}, \text{logSets}) \geq \text{RecoveryQuorumSize}\} \\
& \text{unSyncedLogSeq} \triangleq \text{SetToSortSeq}(\text{unSyncedLogs}, \text{Compare}) \\
& \text{IN} \\
& \text{syncedLogSeq} \circ \text{unSyncedLogSeq} \\
& \text{SelectEntriesBeyondCommitPoint}(\text{entries}, \text{timestamp}) \triangleq \\
& \quad \text{LET} \\
& \quad \quad \text{validLogIndices} \triangleq \{ \\
& \quad \quad \quad i \in 1 \dots \text{Len}(\text{entries}) : \text{entries}[i].\text{timestamp} > \text{timestamp} \\
& \quad \quad \} \\
& \quad \quad \text{startIndex} \triangleq \text{PickMin}(\text{validLogIndices}) \\
& \quad \text{IN} \\
& \quad \text{SubSeq}(\text{entries}, \text{startIndex}, \text{Len}(\text{entries})) \\
& \text{HandleViewChange}(m) \triangleq \\
& \quad \text{LET} \\
& \quad \quad \text{myServerId} \triangleq m.\text{dest} \\
& \quad \quad \text{serversInOneShard} \triangleq \{s \in \text{Servers} : s.\text{shardId} = \text{myServerId.shardId}\} \\
& \quad \quad \text{leadersInAllShard} \triangleq \{ \\
& \quad \quad \quad s \in \text{Servers} : s.\text{replicaId} = \text{isLeader}(s.\text{replicaId}, m.\text{gVec}[s.\text{shardId}]) \\
& \quad \quad \} \\
& \quad \text{IN} \\
& \quad \wedge \vee \text{ViewGreater}(m.\text{gView}, m.\text{lView}, v\text{GView}[\text{myServerId}], v\text{LView}[\text{myServerId}]) \\
& \quad \quad \vee \wedge m.\text{gView} = v\text{GView}[\text{myServerId}] \\
& \quad \quad \quad \wedge m.\text{lView} = v\text{LView}[\text{myServerId}] \\
& \quad \quad \quad \wedge v\text{ServerStatus}[\text{myServerId}] = \text{StViewChange} \\
& \quad \wedge \text{isLeader}(\text{myServerId.replicaId}, m.\text{lView}) \\
& \quad \wedge v\text{GView}' = [v\text{GView} \text{ EXCEPT } ![\text{myServerId}] = m.\text{gView}] \\
& \quad \wedge v\text{LView}' = [v\text{LView} \text{ EXCEPT } ![\text{myServerId}] = m.\text{lView}] \\
& \quad \wedge v\text{GVec}' = [v\text{GVec} \text{ EXCEPT } ![\text{myServerId}] = m.\text{gVec}] \\
& \quad \wedge v\text{ViewChange}' = [\\
& \quad \quad \quad v\text{ViewChange} \text{ EXCEPT } ![\text{myServerId}] = \{ \\
& \quad \quad \quad \quad vc \in v\text{ViewChange}[\text{myServerId}] : \\
& \quad \quad \quad \quad \quad vc.\text{lView} = m.\text{lView} \\
& \quad \quad \quad \} \cup \{m\} \\
& \quad \quad] \\
& \quad \wedge \text{IF } \text{Cardinality}(v\text{ViewChange}'[\text{myServerId}]) = \text{QuorumSize} \text{ THEN} \\
& \quad \quad \quad \wedge v\text{Log}' = [v\text{Log} \text{ EXCEPT } ![\text{myServerId}] = \text{ReBuildLogs}(v\text{ViewChange}'[\text{myServerId}])]
\end{aligned}$$

$$\wedge vServerStatus' = [vServerStatus \text{ EXCEPT } ![myServerId] = StCrossShardSyncing]$$

Even after the *log* is recovered within one shard,

- * The newly elected leader cannot *StartView*
- * It needs to verify with other *shards'* leaders to ensure strict serializability

$$\wedge vViewChange' = [vViewChange \text{ EXCEPT } ![myServerId] = \{\}]$$

$$\wedge Send(\{[$$

$$\begin{array}{ll} mtype & \mapsto MCrossShardVerifyReq, \\ sender & \mapsto myServerId, \\ dest & \mapsto dst, \\ lView & \mapsto vLView'[myServerId], \\ gView & \mapsto vGView'[myServerId], \\ syncedDdl & \mapsto vLog[myServerId][vLSyncPoint[myServerId]].timestamp \end{array}$$

$$]: dst \in leadersInAllShard\})$$

ELSE

$$\wedge vServerStatus' = [vServerStatus \text{ EXCEPT } ![myServerId] = StViewChange]$$

$$\wedge \text{UNCHANGED } \langle networkVars, vLog \rangle$$

HandleCrossShardVerifyReq(*m*) \triangleq

LET

$$\begin{array}{l} myServerId \triangleq m.dest \\ myLog \triangleq vLog[myServerId] \\ logSet \triangleq SeqToSet(myLog) \\ unVerifiedLogs \triangleq \{ \\ \quad e \in logSet : \quad \wedge e.timestamp > m.syncedDdl \\ \quad \quad \quad \wedge m.sender \in e.shards \\ \} \\ unVerifiedLogList \triangleq SetToSortSeq(unVerifiedLogs, Compare) \end{array}$$

IN

$$\begin{array}{l} \wedge m.gView = vGView[myServerId] \\ \wedge m.lView = vGVec[myServerId][m.sender.shardId] \\ \wedge Send(\{[\\ \quad mtype \quad \mapsto MCrossShardVerifyRep, \\ \quad sender \quad \mapsto myServerId, \\ \quad dest \quad \mapsto m.sender, \\ \quad lView \quad \mapsto vLView[myServerId], \\ \quad gView \quad \mapsto vGView[myServerId], \\ \quad entries \quad \mapsto unVerifiedLogList \\ \quad]\}) \end{array}$$

HandleCrossShardVerifyRep(*m*) \triangleq

LET

$$\begin{array}{l} myServerId \triangleq m.dest \\ myLog \triangleq vLog[myServerId] \\ myLogSet \triangleq SeqToSet(myLog) \end{array}$$

IN

```

 $\wedge$   $m.gView = vGView[myServerId]$ 
 $\wedge$   $m.lView = vGVec[myServerId][m.sender.shardId]$ 
 $\wedge$   $vCrossShardVerifyReps' = [$ 
     $vCrossShardVerifyReps$  EXCEPT  $![myServerId] =$ 
     $vCrossShardVerifyReps[myServerId] \cup \{m\}$ 
 $\wedge$  IF  $Cardinality(vCrossShardVerifyReps'[myServerId]) = Cardinality(Shards)$ 
    THEN
        LET
             $unVerifiedLogs \triangleq$  UNION  $\{SeqToSet(mm.entries) :$ 
                 $mm \in vCrossShardVerifyReps'[myServerId]\}$ 
             $maxTimestampLogs \triangleq \{$ 
                 $e \in unVerifiedLogs :$ 
                 $\forall x \in unVerifiedLogs :$ 
                 $\vee x.txnId \neq e.txnId$ 
                 $\vee x.timestamp \leq e.timestamp$ 
             $\}$ 
             $agreedLogs \triangleq \{$ 
                 $e \in maxTimestampLogs :$ 
                the reciving shard is missing this  $txn$ 
                 $\vee \forall x \in myLogSet : x.txnId \neq e.txnId$ 
                 $\vee \exists x \in myLogSet : x.timestamp < e.timestamp$ 
             $\}$ 
             $goodLogs \triangleq \{$ 
                 $e \in myLogSet : \forall x \in agreedLogs : x.txnId \neq e.txnId$ 
             $\}$ 
             $completeLogs \triangleq goodLogs \cup agreedLogs$ 
             $newLogList \triangleq SetToSortSeq(completeLogs, Compare)$ 
        IN
             $vLog' = [vLog$  EXCEPT  $![myServerId] = newLogList]$ 
    ELSE
        UNCHANGED  $\langle vLog \rangle$ 

```

```

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

```

$$\begin{aligned}
& \text{HandleStartView}(m) \triangleq \\
& \quad \text{LET} \\
& \quad \quad \text{myServerId} \triangleq m.\text{dest} \\
& \quad \text{IN} \\
& \quad \wedge \vee \text{ViewGreater}(m.g\text{View}, m.l\text{View}, vG\text{View}[\text{myServerId}], vL\text{View}[\text{myServerId}]) \\
& \quad \vee \wedge m.g\text{View} = vG\text{View}[\text{myServerId}] \\
& \quad \wedge m.l\text{View} = vL\text{View}[\text{myServerId}] \\
& \quad \wedge \vee v\text{ServerStatus}[\text{myServerId}] = \text{StViewChange} \\
& \quad \quad \vee v\text{ServerStatus}[\text{myServerId}] = \text{StRecovering} \\
& \quad \wedge vG\text{View}' = [vG\text{View} \text{ EXCEPT } ![\text{myServerId}] = m.g\text{View}] \\
& \quad \wedge vL\text{View}' = [vL\text{View} \text{ EXCEPT } ![\text{myServerId}] = m.gL\text{View}] \\
& \quad \wedge vG\text{Vec}' = [vG\text{Vec} \text{ EXCEPT } ![\text{myServerId}] = m.vG\text{Vec}] \\
& \quad \wedge v\text{ServerStatus}' = [v\text{ServerStatus} \text{ EXCEPT } ![\text{myServerId}] = \text{StNormal}] \\
& \quad \wedge v\text{Log}' = [v\text{Log} \text{ EXCEPT } ![\text{myServerId}] = m.\text{entries}] \\
& \quad \wedge v\text{EarlyBuffer}' = [v\text{EarlyBuffer} \text{ EXCEPT } ![\text{myServerId}] = \{\}] \\
& \quad \wedge v\text{LateBuffer}' = [v\text{LateBuffer} \text{ EXCEPT } ![\text{myServerId}] = \{\}] \\
& \quad \wedge v\text{TimestampQuorum}' = [v\text{TimestampQuorum} \text{ EXCEPT } ![\text{myServerId}] = \{\}] \\
& \quad \wedge v\text{CrossShardVerifyReps}' = [\\
& \quad \quad v\text{CrossShardVerifyReps} \text{ EXCEPT } ![\text{myServerId}] = \{\} \\
& \quad] \\
& \quad \wedge v\text{LSyncPoint}' = [v\text{LSyncPoint} \text{ EXCEPT } ![\text{myServerId}] = \text{Len}(v\text{Log}'[\text{myServerId}])] \\
& \quad \wedge v\text{LastNormView}' = [v\text{LastNormView} \text{ EXCEPT } ![\text{myServerId}] = m.l\text{View}] \\
& \quad \wedge v\text{ViewChange}' = [v\text{ViewChange} \text{ EXCEPT } ![\text{myServerId}] = \{\}] \\
& \quad \wedge v\text{LSyncQuorum}' = [v\text{LSyncQuorum} \text{ EXCEPT } ![\text{myServerId}] = \{\}] \\
& \quad \wedge v\text{CrashVectorReps}' = [v\text{CrashVectorReps} \text{ EXCEPT } ![\text{myServerId}] = \{\}] \\
& \quad \wedge v\text{RecoveryReps}' = [v\text{RecoveryReps} \text{ EXCEPT } ![\text{myServerId}] = \{\}]
\end{aligned}$$

$$\begin{aligned}
& \text{ResetServerState}(\text{serverId}) \triangleq \\
& \quad \wedge v\text{Log}' = [v\text{Log} \text{ EXCEPT } ![\text{serverId}] = \langle \rangle] \\
& \quad \wedge v\text{EarlyBuffer}' = [v\text{EarlyBuffer} \text{ EXCEPT } ![\text{serverId}] = \{\}] \\
& \quad \wedge v\text{LateBuffer}' = [v\text{LateBuffer} \text{ EXCEPT } ![\text{serverId}] = \{\}] \\
& \quad \wedge v\text{TimestampQuorum}' = [v\text{TimestampQuorum} \text{ EXCEPT } ![\text{serverId}] = \{\}] \\
& \quad \wedge v\text{CrossShardVerifyReps}' = [\\
& \quad \quad v\text{CrossShardVerifyReps} \text{ EXCEPT } ![\text{serverId}] = \{\} \\
& \quad] \\
& \quad \wedge vG\text{View}' = [vG\text{View} \text{ EXCEPT } ![\text{serverId}] = 0] \\
& \quad \wedge vG\text{Vec}' = [vG\text{Vec} \text{ EXCEPT } ![\text{serverId}] = [s \in \text{Shards} \mapsto 0]] \\
& \quad \wedge vL\text{View}' = [vL\text{View} \text{ EXCEPT } ![\text{serverId}] = 0] \\
& \quad \wedge v\text{LastNormView}' = [v\text{LastNormView} \text{ EXCEPT } ![\text{serverId}] = 0] \\
& \quad \wedge v\text{ViewChange}' = [v\text{ViewChange} \text{ EXCEPT } ![\text{serverId}] = \{\}] \\
& \quad \wedge v\text{LSyncPoint}' = [v\text{LSyncPoint} \text{ EXCEPT } ![\text{serverId}] = 0] \\
& \quad \wedge v\text{LCommitPoint}' = [v\text{LCommitPoint} \text{ EXCEPT } ![\text{serverId}] = 0] \\
& \quad \wedge v\text{LSyncQuorum}' = [v\text{LSyncQuorum} \text{ EXCEPT } ![\text{serverId}] = \{\}] \\
& \quad \wedge v\text{CrashVector}' = [v\text{CrashVector} \text{ EXCEPT } ![\text{serverId}] = [
\end{aligned}$$

$$\begin{aligned}
& rr \in \text{Replicas} \mapsto 0 \\
&]] \\
& \wedge v\text{CrashVectorReps}' = [v\text{CrashVectorReps} \text{ EXCEPT } ![serverId] = \{\}] \\
& \wedge v\text{RecoveryReps}' = [v\text{RecoveryReps} \text{ EXCEPT } ![serverId] = \{\}] \\
& \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![serverId] = \{\}] \\
\text{StartServerRecovery}(serverId) & \triangleq \\
\text{LET} & \\
& serversInOneShard \triangleq \{ \\
& \quad s \in \text{Servers} : s.shardId = serverId.shardId \\
& \} \\
& nonce \triangleq v\text{UUIDCounter}[serverId] + 1 \\
\text{IN} & \\
& \wedge v\text{ServerStatus}' = [v\text{ServerStatus} \text{ EXCEPT } ![serverId] = StRecovering] \\
& \wedge v\text{UUIDCounter}' = [v\text{UUIDCounter} \text{ EXCEPT } ![serverId] = v\text{UUIDCounter}[serverId] + 1] \\
& \wedge \text{Send}(\{[\\
& \quad mtype \quad \mapsto M\text{CrashVectorReq}, \\
& \quad sender \quad \mapsto serverId, \\
& \quad dest \quad \mapsto dst, \\
& \quad nonce \quad \mapsto nonce \\
& \quad] : dst \in serversInOneShard\}) \\
\text{HandleCrashVectorReq}(m) & \triangleq \\
\text{LET} & \\
& myServerId \triangleq m.dest \\
\text{IN} & \\
& \wedge v\text{ServerStatus}[myServerId] = StNormal \\
& \wedge \text{Send}(\{[\\
& \quad mtype \quad \mapsto M\text{CrashVectorRep}, \\
& \quad sender \quad \mapsto myServerId, \\
& \quad dest \quad \mapsto m.sender, \\
& \quad nonce \quad \mapsto m.nonce, \\
& \quad cv \quad \mapsto v\text{CrashVector}[myServerId] \\
& \quad]\}) \\
\text{AggregateCV}(serverId) & \triangleq \\
\text{LET} & \\
& cv\text{Quorum} \triangleq \{m.cv : m \in v\text{CrashVectorReps}[serverId]\} \\
& cv\text{ValQuorum} \triangleq [rr \in \text{Replicas} \mapsto \{cv[rr] : cv \in cv\text{Quorum}\}] \\
\text{IN} & \\
& [rr \in \text{Replicas} \mapsto \text{PickMax}(cv\text{ValQuorum}[rr])] \\
\text{HandleCrashVectorRep}(m) & \triangleq \\
\text{LET} & \\
& myServerId \triangleq m.dest
\end{aligned}$$

```

    serversInOneShard  $\triangleq$   $\{s \in Servers : s.shardId = myServerId.shardId\}$ 
IN
 $\wedge$   $vServerStatus[myServerId] = StRecovering$ 
 $\wedge$   $vUUIDCounter[myServerId] = m.nonce$ 
 $\wedge$   $vCrashVectorReps' = [$ 
     $vCrashVectorReps$  EXCEPT  $![myServerId] = vCrashVectorReps \cup \{m\}$ 
 $]$ 
 $\wedge$  IF  $Cardinality(vCrashVectorReps'[myServerId]) = QuorumSize$  THEN
    LET
         $acv \triangleq AggregateCV(myServerId)$ 
         $myCV \triangleq [acv$  EXCEPT  $![myServerId] = acv[myServerId] + 1]$ 
    IN
         $\wedge$   $vCrashVector' = [$ 
             $vCrashVector$  EXCEPT  $![myServerId] = myCV$ 
         $]$ 
         $\wedge$  Send( $\{[$ 
             $mtype \mapsto MRecoveryReq,$ 
             $sender \mapsto myServerId,$ 
             $dest \mapsto dst,$ 
             $cv \mapsto myCV$ 
         $]: dst \in serversInOneShard\}$ )
    ELSE
        UNCHANGED  $\langle networkVars, vCrashVector \rangle$ 

```

```

HandleRecoveryReq( $m$ )  $\triangleq$ 
    LET
         $myServerId \triangleq m.dest$ 
    IN
         $\wedge$   $vServerStatus[myServerId] = StNormal$ 
         $\wedge$  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
         $\wedge$   $vServerStatus[myServerId] = StRecovering$ 
         $\wedge$   $vRecoveryReps' = [$ 

```

```

    vRecoveryReps EXCEPT ![myServerId]
      = vRecoveryReps[myServerId] ∪ {m}
  ]
  ∧ IF Cardinality(vRecoveryReps[myServerId]) = QuorumSize THEN
    LET
      lViewQuorum  $\triangleq$  {mm.lView : mm ∈ vRecoveryReps[myServerId]}
      gViewQuorum  $\triangleq$  {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)  $\triangleq$ 
  LET
    myServerId  $\triangleq$  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)  $\triangleq$ 
  LET
    leaderServerId  $\triangleq$  [

```

```

      replicaId  $\mapsto$  LeaderID( $vLView[serverId]$ ),
      shardId   $\mapsto$  serverId.shardId
    ]
  IN
   $\wedge$  vServerStatus[serverId] = StNormal
   $\wedge$  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
   $\wedge$  vServerStatus[myServerId] = StNormal
   $\wedge$  vLView[myServerId] =  $m.lView$ 
   $\wedge$  isLeader(myServerId.replicaId, vLView[myServerId])
   $\wedge$   $\forall mm \in lSyncQuorum :$ 
     $\vee$   $mm.sender \neq m.sender$ 
     $\vee$   $mm.lSyncPoint < m.lSyncPoint$ 
   $\wedge$  vLSyncQuorum' = [
    vLSyncQuorum EXCEPT ![myServerId] =
      { $mm \in lSyncQuorum : mm.sender \neq m.sender$ }  $\cup$  { $m$ }
  ]
   $\wedge$  IF Cardinality(vLSyncQuorum'[myServerId])  $\geq$  QuorumSize THEN
    LET
      candidateQuorum  $\triangleq$  {
         $R \in \text{SUBSET}(vLSyncQuorum'[myServerId]) :$ 
        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
     $\wedge$  vLCommitPoint' = [vLCommitPoint EXCEPT ![myServerId] = maxCommitPoint]
     $\wedge$  Send([
      mtype       $\mapsto$  MLocalCommit,

```

```

        sender      ↦ myServerId,
        dest        ↦ m.sender,
        lView       ↦ vLView[myServerId],
        lCommitPoint ↦ vLCommitPoint'[myServerId],
        cv          ↦ vCrashVector'[myServerId]
    })
ELSE    UNCHANGED ⟨vLCommitPoint, networkVars⟩

```

$HandleLocalCommit(m) \triangleq$

```

    LET
        myServerId  $\triangleq$  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⟩

```

$isCommitting(txn, timestampQ) \triangleq$

```

    LET quorum  $\triangleq$  {msg ∈ timestampQ : msg.entry.txnId = txn.txnId}
    IN    Cardinality(quorum) = Cardinality(txn.shards)

```

$ReleaseSequencer(serverId, currentTime) \triangleq$

```

    LET
        serversInOneShard  $\triangleq$  {s ∈ Servers : s.shardId = serverId.shardId}
        expireTxns  $\triangleq$ 
            {msg ∈ vEarlyBuffer[serverId] :
                ∧ msg.timestamp ≤ currentTime}
        sortedTxnList  $\triangleq$  SetToSortSeq(expireTxns, Compare)
        committingStatus  $\triangleq$ 
            [i ∈ 1 .. Len(sortedTxnList)
                ↦ isCommitting(sortedTxnList[i], vTimestampQuorum[serverId])
            ]
        canReleaseTxnIndices  $\triangleq$  {
            i ∈ 1 .. Len(sortedTxnList) :
                ∀ j ∈ 1 .. i : committingStatus[j] = TRUE}

```

Here we consider all txns are not commutative,
 Therefore, At most one txn can be speculatively executed with risk
 Refer to *RRSE* in Section 3.6

$$specTxnIndex \triangleq \{$$

$$i \in 1 \dots Len(sortedTxnList) :$$

$$\wedge \forall j \in 1 \dots (i - 1) : committingStatus[j] = \text{TRUE}$$

$$\wedge committingStatus[i] = \text{FALSE}$$

$$\}$$

IN

IF $Cardinality(canReleaseTxnIndices) = 0$ Nothing to release

THEN

\wedge UNCHANGED $\langle vLog, vEarlyBuffer, vLateBuffer, vTimestampQuorum \rangle$

While there is nothing to release, some txns might be speculatively executed

\wedge IF $Cardinality(specTxnIndex) > 0$ THEN

Send($\{$

$mtype \mapsto MFastReply,$

$sender \mapsto serverId,$

$dest \mapsto sortedTxnList[i].txnId.coordId,$

$gView \mapsto vGView[serverId],$

$lView \mapsto vLView[serverId],$

$txnId \mapsto sortedTxnList[i].txnId,$

$hash \mapsto [$
 $\quad log \mapsto vLog'[serverId],$
 $\quad cv \mapsto vCrashVector$
 $\quad],$

$t \mapsto sortedTxnList[i].timestamp,$

$logId \mapsto 0$ $logId = 0$ indicates this is a speculative txn with rollback risk,
 we need to compare the timestamps from different *shards* to decide
 whether the execution results are serializable

$\} : i \in specTxnIndex \}$

ELSE

UNCHANGED $\langle networkVars \rangle$

ELSE

LET

$releaseUpTo \triangleq \text{CHOOSE } i \in canReleaseTxnIndices :$

$\quad \forall j \in canReleaseTxnIndices : j \leq i$

$releaseSeq \triangleq SubSeq(sortedTxnList, 1, releaseUpTo)$

$releaseTxns \triangleq \{releaseSeq[i] : i \in 1 \dots Len(releaseSeq)\}$

IN

$\wedge vEarlyBuffer' = [$

$\quad vEarlyBuffer \text{ EXCEPT } ![serverId]$

$\quad = vEarlyBuffer[serverId] \setminus releaseTxns]$

$\wedge vTimestampQuorum' = [$

$\quad vTimestampQuorum \text{ EXCEPT } ![serverId]$

```

    = {msg ∈ vTimestampQuorum[serverId] :
        ∀ txn ∈ releaseTxns : txn.txnId ≠ msg.entry.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
    ],
    t ↦ 0, timestamp = 0 indicates this is not a speculative txn with rollback risk
    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})
  ∧ IF Cardinality(specTxnIndex) > 0 THEN
    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
      ],
    ]

```

$$\begin{aligned}
& t \mapsto \text{sortedTxnList}[i].\text{timestamp}, \\
& \text{logId} \mapsto 0 \quad \text{logId} = 0 \text{ indicates this is a speculative txn with rollback risk} \\
&] : i \in \text{specTxnIndex}\}) \\
& \text{ELSE} \\
& \quad \text{TRUE}
\end{aligned}$$

$$\begin{aligned}
& \text{ServerClockMove}(\text{serverId}) \triangleq \\
& \quad \text{IF } v\text{ServerClock}[\text{serverId}] \geq \text{MaxTime} \quad \text{THEN} \\
& \quad \quad \text{UNCHANGED } \langle \text{networkVars}, \text{serverStateVars} \rangle \\
& \quad \text{ELSE} \\
& \quad \quad \wedge v\text{ServerClock}' = [\\
& \quad \quad \quad v\text{ServerClock} \text{ EXCEPT } ![\text{serverId}] = v\text{ServerClock}[\text{serverId}] + 1] \\
& \quad \quad \wedge \text{IF } v\text{ServerStatus}[\text{serverId}] = \text{StNormal} \text{ THEN} \\
& \quad \quad \quad \wedge \text{ReleaseSequencer}(\text{serverId}, v\text{ServerClock}[\text{serverId}] + 1) \\
& \quad \quad \text{ELSE} \\
& \quad \quad \quad \text{UNCHANGED } \langle \text{networkVars}, v\text{Log}, v\text{EarlyBuffer}, \\
& \quad \quad \quad \quad v\text{LateBuffer}, v\text{TimestampQuorum} \rangle \\
& \quad \quad \wedge \text{UNCHANGED } \langle v\text{CrossShardVerifyReps}, \\
& \quad \quad \quad v\text{ServerStatus}, v\text{GView}, v\text{GVec}, v\text{LView}, v\text{LastNormView}, \\
& \quad \quad \quad v\text{ViewChange}, v\text{LSyncPoint}, v\text{LCommitPoint}, \\
& \quad \quad \quad v\text{LSyncQuorum}, v\text{UUIDCounter}, v\text{CrashVector}, \\
& \quad \quad \quad v\text{CrashVectorReps}, v\text{RecoveryReps}, v\text{ServerProcessed} \rangle
\end{aligned}$$

$$\begin{aligned}
& \text{CoordClockMove}(\text{coordId}) \triangleq \\
& \quad \vee \wedge v\text{CoordClock}[\text{coordId}] \geq \text{MaxTime} \\
& \quad \quad \wedge \text{UNCHANGED } \langle v\text{CoordClock} \rangle \\
& \quad \vee \wedge v\text{CoordClock}[\text{coordId}] < \text{MaxTime} \\
& \quad \quad \wedge v\text{CoordClock}' = [\\
& \quad \quad \quad v\text{CoordClock} \text{ EXCEPT } ![\text{coordId}] = v\text{CoordClock}[\text{coordId}] + 1]
\end{aligned}$$

$$\begin{aligned}
& \text{Init} \triangleq \\
& \quad \wedge \text{InitNetworkState} \\
& \quad \wedge \text{InitServerState} \\
& \quad \wedge \text{InitCoordState} \\
& \quad \wedge \text{InitConfigManagerState} \\
& \quad \wedge \text{ActionName} = \langle \text{"Init"} \rangle
\end{aligned}$$

$$\begin{aligned}
& \text{Next} \triangleq \\
& \quad \vee \wedge \text{ActionName}' = \langle \text{"Next"} \rangle \\
& \quad \quad \wedge \text{UNCHANGED } \langle \text{networkVars}, \text{serverStateVars}, \\
& \quad \quad \quad \text{coordStateVars}, \text{configManagerStateVars} \rangle \\
& \quad \vee \exists c \in \text{Coords} : \\
& \quad \quad \wedge \text{Cardinality}(v\text{CoordTxns}[c]) < \text{MaxReqNum}
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{CoordSubmitTxn}(c) \\
& \wedge \text{UNCHANGED } \langle \text{serverStateVars}, \text{configManagerStateVars}, \\
& \quad \text{vCoordProcessed} \rangle \\
& \wedge \text{ActionName}' = \langle \text{"CoordSubmitTxn"} \rangle \\
\vee \exists m \in \text{messages} : \\
& \wedge m.\text{mtype} = \text{MTxn} \\
& \wedge \text{vServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \wedge m \notin \text{vServerProcessed}[m.\text{dest}] \\
& \wedge \text{vServerProcessed}' = [\text{vServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \text{vServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \wedge \text{HandleTxn}(m) \\
& \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad \text{vLog}, \text{vTimestampQuorum}, \text{vCrossShardVerifyReps}, \\
& \quad \text{vServerStatus}, \text{vGView}, \text{vGVec}, \\
& \quad \text{vLView}, \text{vServerClock}, \text{vLastNormView}, \\
& \quad \text{vViewChange}, \text{vLSyncPoint}, \text{vLCommitPoint}, \\
& \quad \text{vLSyncQuorum}, \text{vUUIDCounter}, \text{vCrashVector}, \\
& \quad \text{vCrashVectorReps}, \text{vRecoveryReps} \rangle \\
& \wedge \text{ActionName}' = \langle \text{"HandleTxn"} \rangle \\
\vee \exists m \in \text{messages} : \\
& \wedge m.\text{mtype} = \text{MTimestampNotification} \\
& \wedge \text{vServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \wedge m \notin \text{vServerProcessed}[m.\text{dest}] \\
& \wedge \text{vServerProcessed}' = [\text{vServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \text{vServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \wedge \text{HandleTimestampNotification}(m) \\
& \wedge \text{UNCHANGED } \langle \text{networkVars}, \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad \text{vLog}, \text{vCrossShardVerifyReps}, \text{vLateBuffer}, \text{vServerStatus}, \\
& \quad \text{vGView}, \text{vGVec}, \text{vLView}, \text{vServerClock}, \text{vLastNormView}, \\
& \quad \text{vViewChange}, \text{vLSyncPoint}, \text{vLCommitPoint}, \text{vLSyncQuorum}, \\
& \quad \text{vUUIDCounter}, \text{vCrashVector}, \text{vCrashVectorReps}, \text{vRecoveryReps} \rangle \\
& \wedge \text{ActionName}' = \langle \text{"HandleTimestampNotification"} \rangle \\
\vee \exists m \in \text{messages} : \\
& \wedge m.\text{mtype} = \text{MInterReplicaSync} \\
& \wedge \text{vServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \wedge m \notin \text{vServerProcessed}[m.\text{dest}] \\
& \wedge \text{vServerProcessed}' = [\text{vServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \text{vServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \wedge \text{HandleInterReplicaSync}(m) \\
& \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad \text{vLog}, \text{vCrossShardVerifyReps}, \text{vLateBuffer}, \\
& \quad \text{vServerStatus}, \text{vGView}, \text{vGVec}, \text{vLView}, \\
& \quad \text{vServerClock}, \text{vLastNormView},
\end{aligned}$$

$$\begin{aligned}
& vViewChange, vLCommitPoint, \\
& vLSyncQuorum, vUUIDCounter, vCrashVector, \\
& vCrashVectorReps, vRecoveryReps \rangle \\
& \wedge ActionName' = \langle "HandleInterReplicaSync" \rangle
\end{aligned}$$

Some *Leader(s)* fail

$$\begin{aligned}
& \vee \exists serverId \in Servers : \\
& \quad \wedge vLView[serverId] < MaxViews \\
& \quad \wedge isLeader(serverId.replicaId, vLView[serverId]) \\
& \quad \wedge vServerStatus[serverId] = StNormal \\
& \quad \wedge StartLeaderFail(serverId) \\
& \quad \wedge UNCHANGED \langle networkVars, coordStateVars, configManagerStateVars, \\
& \quad \quad vLog, vEarlyBuffer, vLateBuffer, vTimestampQuorum, \\
& \quad \quad vCrossShardVerifyReps, vGView, vGVec, vLView, vServerClock, \\
& \quad \quad vLastNormView, vViewChange, vLSyncPoint, vLCommitPoint, \\
& \quad \quad vLSyncQuorum, vUUIDCounter, vCrashVector, vCrashVectorReps, \\
& \quad \quad vRecoveryReps, vServerProcessed \rangle \\
& \quad \wedge ActionName' = \langle "StartLeaderFail" \rangle
\end{aligned}$$

Config Manager notices some *leader(s)* fail and launch view change

$$\begin{aligned}
& \vee \exists cmReplicaId \in Replicas : \\
& \quad \wedge LaunchViewChange(cmReplicaId) \\
& \quad \wedge UNCHANGED \langle coordStateVars, serverStateVars, configManagerStateVars \rangle \\
& \quad \wedge ActionName' = \langle "LaunchViewChange" \rangle \\
& \vee \exists m \in messages : \\
& \quad \wedge m.mtype = MCMPPrepare \\
& \quad \wedge m \notin vCMProcessed[m.dest] \\
& \quad \wedge vCMProcessed' = [vCMProcessed \text{ EXCEPT } ![m.dest] = \\
& \quad \quad vCMProcessed[m.dest] \cup \{m\}] \\
& \quad \wedge vCMStatus[m.dest] = StNormal \\
& \quad \wedge HandleCMPPrepare(m) \\
& \quad \wedge UNCHANGED \langle coordStateVars, serverStateVars \rangle \\
& \quad \wedge ActionName' = \langle "HandleCMPPrepare" \rangle \\
& \vee \exists m \in messages : \\
& \quad \wedge m.mtype = MCMPPrepareReply \\
& \quad \wedge m \notin vCMProcessed[m.dest] \\
& \quad \wedge vCMProcessed' = [vCMProcessed \text{ EXCEPT } ![m.dest] = \\
& \quad \quad vCMProcessed[m.dest] \cup \{m\}] \\
& \quad \wedge vCMStatus[m.dest] = StNormal \\
& \quad \wedge HandleCMPPrepareReply(m) \\
& \quad \wedge UNCHANGED \langle coordStateVars, serverStateVars, \\
& \quad \quad vCMStatus, vCMView, vCMPPrepareGInfo \rangle \\
& \quad \wedge ActionName' = \langle "HandleCMPPrepareReply" \rangle
\end{aligned}$$

$$\begin{aligned}
& \vee \exists m \in \text{messages} : \\
& \quad \wedge m.\text{mtype} = \text{MCMCommit} \\
& \quad \wedge m \notin v\text{CMProcessed}[m.\text{dest}] \\
& \quad \wedge v\text{CMProcessed}' = [v\text{CMProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \quad v\text{CMProcessed}[m.\text{dest}] \cup \{m\}] \\
& \quad \wedge v\text{CMStatus}[m.\text{dest}] = \text{StNormal} \\
& \quad \wedge \text{HandleCMCommit}(m) \\
& \quad \wedge \text{UNCHANGED } \langle \text{networkVars}, \text{coordStateVars}, \text{serverStateVars}, \\
& \quad \quad v\text{CMStatus}, v\text{CMView}, v\text{CMPPrepareGInfo}, v\text{CMPPrepareReps} \rangle \\
& \quad \wedge \text{ActionName}' = \langle \text{"HandleCMCommit"} \rangle \\
\\
& \vee \exists m \in \text{messages} : \\
& \quad \wedge m.\text{mtype} = \text{MViewChangeReq} \\
& \quad \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \quad \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \quad v\text{ServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \quad \wedge \vee v\text{ServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \quad \quad \vee v\text{ServerStatus}[m.\text{dest}] = \text{StViewChange} \\
& \quad \wedge \text{HandleViewChangeReq}(m) \\
& \quad \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad \quad v\text{Log}, v\text{ServerClock}, v\text{ViewChange}, v\text{LSyncPoint}, \\
& \quad \quad v\text{LCommitPoint}, v\text{LSyncQuorum}, v\text{UUIDCounter}, \\
& \quad \quad v\text{CrashVector}, v\text{CrashVectorReps}, v\text{RecoveryReps} \rangle \\
& \quad \wedge \text{ActionName}' = \langle \text{"HandleViewChangeReq"} \rangle \\
\\
& \vee \exists m \in \text{messages} : \\
& \quad \wedge m.\text{mtype} = \text{MViewChange} \\
& \quad \wedge \text{isCrashVectorValid}(m) \\
& \quad \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \quad \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \quad v\text{ServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \quad \wedge \vee v\text{ServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \quad \quad \vee v\text{ServerStatus}[m.\text{dest}] = \text{StViewChange} \\
& \quad \wedge \text{HandleViewChange}(m) \\
& \quad \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad \quad v\text{GVec}, v\text{ServerClock}, v\text{LSyncPoint}, v\text{LastNormView}, \\
& \quad \quad v\text{LCommitPoint}, v\text{LSyncQuorum}, v\text{UUIDCounter}, \\
& \quad \quad v\text{CrashVector}, v\text{CrashVectorReps}, v\text{RecoveryReps} \rangle \\
& \quad \wedge \text{ActionName}' = \langle \text{"HandleViewChange"} \rangle \\
\\
& \vee \exists m \in \text{messages} : \\
& \quad \wedge m.\text{mtype} = \text{MCrossShardVerifyReq} \\
& \quad \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \quad \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] =
\end{aligned}$$

$$\begin{aligned}
& vServerProcessed[m.dest] \cup \{m\} \\
\wedge & vServerStatus[m.dest] = StCrossShardSyncing \\
\wedge & HandleCrossShardVerifyReq(m) \\
\wedge & UNCHANGED \langle coordStateVars, configManagerStateVars, \\
& vLog, vEarlyBuffer, vLateBuffer, vTimestampQuorum, \\
& vCrossShardVerifyReps, vServerStatus, \\
& vGView, vGVec, vLView, vServerClock, vLastNormView, \\
& vViewChange, vLSyncPoint, vLCommitPoint, vLSyncQuorum, \\
& vUUIDCounter, vCrashVector, vCrashVectorReps, \\
& vRecoveryReps \rangle \\
\wedge & ActionName' = \langle \text{"HandleCrossShardVerifyReq"} \rangle \\
\\
\vee & \exists m \in messages : \\
& \wedge m.mtype = MCrossShardVerifyRep \\
& \wedge m \notin vServerProcessed[m.dest] \\
& \wedge vServerProcessed' = [vServerProcessed \text{ EXCEPT } ![m.dest] = \\
& \quad vServerProcessed[m.dest] \cup \{m\}] \\
& \wedge vServerStatus[m.dest] = StCrossShardSyncing \\
& \wedge HandleCrossShardVerifyRep(m) \\
& \wedge UNCHANGED \langle coordStateVars, configManagerStateVars, \\
& \quad vEarlyBuffer, vLateBuffer, vTimestampQuorum, vServerStatus, \\
& \quad vGView, vGVec, vLView, vServerClock, vLastNormView, \\
& \quad vViewChange, vLSyncPoint, vLCommitPoint, \\
& \quad vLSyncQuorum, vUUIDCounter, vCrashVector, \\
& \quad vCrashVectorReps, vRecoveryReps \rangle \\
& \wedge ActionName' = \langle \text{"HandleCrossShardVerifyRep"} \rangle \\
\\
\vee & \exists m \in messages : \\
& \wedge m.mtype = MStartView \\
& \wedge isCrashVectorValid(m) \\
& \wedge m \notin vServerProcessed[m.dest] \\
& \wedge vServerProcessed' = [vServerProcessed \text{ EXCEPT } ![m.dest] = \\
& \quad vServerProcessed[m.dest] \cup \{m\}] \\
& \wedge vServerStatus[m.dest] \neq StFailing \\
& \wedge HandleStartView(m) \\
& \wedge UNCHANGED \langle coordStateVars, configManagerStateVars, \\
& \quad vServerClock, vLCommitPoint, \\
& \quad vUUIDCounter, vCrashVector \rangle \\
& \wedge ActionName' = \langle \text{"HandleStartView"} \rangle \\
\\
\text{Failed server rejoin} \\
\vee & \exists serverId \in Servers : \\
& \wedge vServerStatus[serverId] = StFailing \\
& \wedge vServerStatus' = [vServerStatus \text{ EXCEPT } ![serverId] = StRecovering] \\
& \wedge ResetServerState(serverId)
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{StartServerRecovery}(\text{serverId}) \\
& \wedge \text{UNCHANGED } \langle \text{networkVars}, \text{coordStateVars}, \text{configManagerStateVars} \rangle \\
& \wedge \text{ActionName}' = \langle \text{"StartReplicaRecovery"} \rangle \\
\vee \exists m \in \text{messages} : \\
& \wedge m.\text{mtype} = \text{MCrashVectorReq} \\
& \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad v\text{ServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \wedge v\text{ServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \wedge \text{HandleCrashVectorReq}(m) \\
& \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad v\text{Log}, v\text{EarlyBuffer}, v\text{LateBuffer}, v\text{TimestampQuorum}, \\
& \quad v\text{CrossShardVerifyReps}, v\text{ServerStatus}, v\text{GView}, v\text{GVec}, \\
& \quad v\text{LView}, v\text{ServerClock}, v\text{LastNormView}, v\text{ViewChange}, \\
& \quad v\text{LSyncPoint}, v\text{LCommitPoint}, v\text{LSyncQuorum}, v\text{UUIDCounter}, \\
& \quad v\text{CrashVector}, v\text{CrashVectorReps}, v\text{RecoveryReps} \rangle \\
& \wedge \text{ActionName}' = \langle \text{"HandleCrashVectorReq"} \rangle \\
\vee \exists m \in \text{messages} : \\
& \wedge m.\text{mtype} = \text{MCrashVectorRep} \\
& \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad v\text{ServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \wedge v\text{ServerStatus}[m.\text{dest}] = \text{StRecovering} \\
& \wedge \text{HandleCrashVectorRep}(m) \\
& \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad v\text{Log}, v\text{EarlyBuffer}, v\text{LateBuffer}, \\
& \quad v\text{TimestampQuorum}, v\text{CrossShardVerifyReps}, v\text{ServerStatus}, \\
& \quad v\text{GView}, v\text{GVec}, v\text{LView}, v\text{ServerClock}, v\text{LastNormView}, \\
& \quad v\text{ViewChange}, v\text{LSyncPoint}, v\text{LCommitPoint}, v\text{LSyncQuorum}, \\
& \quad v\text{UUIDCounter}, v\text{CrashVectorReps}, v\text{RecoveryReps} \rangle \\
& \wedge \text{ActionName}' = \langle \text{"HandleCrashVectorRep"} \rangle \\
\vee \exists m \in \text{messages} : \\
& \wedge m.\text{mtype} = \text{MRecoveryReq} \\
& \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad v\text{ServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \wedge v\text{ServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \wedge \text{isCrashVectorValid}(m) \\
& \wedge \text{HandleRecoveryReq}(m) \\
& \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad v\text{Log}, v\text{EarlyBuffer}, v\text{LateBuffer}, \\
& \quad v\text{TimestampQuorum}, v\text{CrossShardVerifyReps}, v\text{ServerStatus}, \\
& \quad v\text{GView}, v\text{GVec}, v\text{LView}, v\text{ServerClock}, v\text{LastNormView},
\end{aligned}$$

$vViewChange, vLSyncPoint, vLCommitPoint, vLSyncQuorum,$
 $vUUIDCounter, vCrashVectorReps, vRecoveryReps\}$
 $\wedge ActionName' = \langle "HandleRecoveryReq" \rangle$

$\vee \exists m \in messages :$
 $\wedge m.mtype = MRecoveryRep$
 $\wedge m \notin vServerProcessed[m.dest]$
 $\wedge vServerProcessed' = [vServerProcessed \text{ EXCEPT } ![m.dest] =$
 $\quad vServerProcessed[m.dest] \cup \{m\}]$
 $\wedge vServerStatus[m.dest] = StRecovering$
 $\wedge isCrashVectorValid(m)$
 $\wedge HandleRecoveryRep(m)$
 $\wedge UNCHANGED \langle coordStateVars, configManagerStateVars,$
 $\quad vLog, vEarlyBuffer, vLateBuffer,$
 $\quad vTimestampQuorum, vCrossShardVerifyReps, vServerStatus,$
 $\quad vGVec, vServerClock, vLastNormView, vViewChange,$
 $\quad vLSyncPoint, vLCommitPoint, vLSyncQuorum,$
 $\quad vUUIDCounter, vCrashVectorReps, vRecoveryReps\}$
 $\wedge ActionName' = \langle "HandleRecoveryRep" \rangle$

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

Periodic Sync

$\vee \exists serverId \in Servers :$
 $\wedge vServerStatus[serverId] = StNormal$
 $\wedge StartLocalSync(serverId)$
 $\wedge UNCHANGED \langle coordStateVars,$
 $\quad serverStateVars, configManagerStateVars \rangle$
 $\wedge ActionName' = \langle "StartLocalSync" \rangle$

$$\begin{aligned}
& \vee \exists m \in \text{messages} : \\
& \quad \wedge m.\text{mtype} = \text{MLocalSyncStatus} \\
& \quad \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \quad \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \quad v\text{ServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \quad \wedge v\text{ServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \quad \wedge \text{isCrashVectorValid}(m) \\
& \quad \wedge \text{HandleLocalSyncStatus}(m) \\
& \quad \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad \quad v\text{Log}, v\text{EarlyBuffer}, v\text{LateBuffer}, v\text{TimestampQuorum}, \\
& \quad \quad v\text{CrossShardVerifyReps}, v\text{ServerClock}, v\text{ViewChange}, \\
& \quad \quad v\text{GVec}, v\text{GView}, v\text{LSyncPoint}, v\text{LView}, v\text{LastNormView}, \\
& \quad \quad v\text{ServerStatus}, v\text{UUIDCounter}, v\text{CrashVectorReps}, \\
& \quad \quad v\text{RecoveryReps} \rangle \\
& \quad \wedge \text{ActionName}' = \langle \text{"HandleLocalSyncStatus"} \rangle \\
\\
& \vee \exists m \in \text{messages} : \\
& \quad \wedge m.\text{mtype} = \text{MLocalCommit} \\
& \quad \wedge m \notin v\text{ServerProcessed}[m.\text{dest}] \\
& \quad \wedge v\text{ServerProcessed}' = [v\text{ServerProcessed} \text{ EXCEPT } ![m.\text{dest}] = \\
& \quad \quad v\text{ServerProcessed}[m.\text{dest}] \cup \{m\}] \\
& \quad \wedge v\text{ServerStatus}[m.\text{dest}] = \text{StNormal} \\
& \quad \wedge \text{isCrashVectorValid}(m) \\
& \quad \wedge \text{HandleLocalCommit}(m) \\
& \quad \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars}, \\
& \quad \quad \text{networkVars}, v\text{Log}, v\text{EarlyBuffer}, v\text{LateBuffer}, \\
& \quad \quad v\text{TimestampQuorum}, v\text{CrossShardVerifyReps}, \\
& \quad \quad v\text{ServerStatus}, v\text{ServerClock}, v\text{GView}, v\text{GVec}, \\
& \quad \quad v\text{LView}, v\text{LastNormView}, v\text{ViewChange}, v\text{LSyncPoint}, \\
& \quad \quad v\text{LSyncQuorum}, v\text{UUIDCounter}, v\text{CrashVectorReps}, v\text{RecoveryReps} \rangle \\
& \quad \wedge \text{ActionName}' = \langle \text{"HandleLocalCommit"} \rangle
\end{aligned}$$

Clock Move

$$\begin{aligned}
& \vee \exists \text{serverId} \in \text{Servers} : \\
& \quad \wedge \text{ServerClockMove}(\text{serverId}) \\
& \quad \wedge \text{UNCHANGED } \langle \text{coordStateVars}, \text{configManagerStateVars} \rangle \\
& \quad \wedge \text{ActionName}' = \langle \text{"ServerClockMove"} \rangle \\
\\
& \vee \exists \text{coordId} \in \text{Coords} : \\
& \quad \wedge \text{CoordClockMove}(\text{coordId}) \\
& \quad \wedge \text{UNCHANGED } \langle \text{networkVars}, \text{serverStateVars}, \text{configManagerStateVars}, \\
& \quad \quad v\text{CoordTxns}, v\text{CoordProcessed} \rangle \\
& \quad \wedge \text{ActionName}' = \langle \text{"CoordClockMove"} \rangle
\end{aligned}$$

$$\text{Spec} \triangleq \text{Init} \wedge \Box[\text{Next}] \langle \text{networkVars}, \text{serverStateVars}, \text{coordStateVars},$$

$\langle \text{configManagerStateVars}, \text{ActionName} \rangle$

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

$\text{CommittedInView}(v, \text{shardId}, \text{txnId}) \triangleq$
 LET
 $\text{serversInOneShard} \triangleq \{s \in \text{Servers} : s.\text{shardId} = \text{shardId}\}$
 $\text{leaderServer} \triangleq [$
 $\text{replicaId} \mapsto \text{LeaderID}(v),$
 $\text{shardId} \mapsto \text{shardId}$
 $]$
 $\text{replySet} \triangleq \{$
 $m \in \text{messages} : \wedge \vee m.\text{mtype} = \text{MFastReply}$
 $\vee m.\text{mtype} = \text{MSlowReply}$
 $\wedge m.\text{txnId} = \text{txnId}$
 $\wedge m.\text{sender} \in \text{serversInOneShard}$
 $\wedge m.\text{lView} = v$
 $\}$
 IN
 IF $\forall \text{reply} \in \text{replySet} :$
 $\vee \text{reply}.\text{mtype} \neq \text{MFastReply}$
 $\vee \text{reply}.\text{sender} \neq \text{leaderServer}$
 THEN No leader's fast reply \rightarrow This txn is not committed
 FALSE
 ELSE
 LET
 $\text{leaderReply} \triangleq \text{CHOOSE } \text{reply} \in \text{replySet} :$
 $\wedge \text{reply}.\text{mtype} = \text{MFastReply}$
 $\wedge \text{reply}.\text{sender} = \text{leaderServer}$
 IN
 Committed in Fast Path

$$\begin{aligned}
& \vee \exists \text{fastQuorum} \in \text{SUBSET } \text{replySet} : \\
& \quad \wedge \text{leaderReply} \in \text{fastQuorum} \\
& \quad \wedge \text{Cardinality}(\text{fastQuorum}) = \text{FastQuorumSize} \\
& \quad \text{All replies have the same hash (or it is a slow reply)} \\
& \quad \wedge \forall \text{reply} \in \text{fastQuorum} : \\
& \quad \quad \vee \wedge \text{reply.mtype} = \text{MFastReply} \\
& \quad \quad \quad \wedge \text{reply.hash} = \text{leaderReply.hash} \\
& \quad \quad \text{Slow Reply can be used as fast reply} \\
& \quad \quad \vee \text{reply.mtype} = \text{MSlowReply} \\
& \quad \text{Committed in Slow Path} \\
& \vee \exists \text{slowQuorum} \in \text{SUBSET } \text{replySet} : \\
& \quad \wedge \text{leaderReply} \in \text{slowQuorum} \\
& \quad \wedge \text{Cardinality}(\text{slowQuorum}) = \text{QuorumSize} \\
& \quad \wedge \forall \text{reply} \in \text{slowQuorum} \setminus \{\text{leaderReply}\} : \\
& \quad \quad \text{reply.mtype} = \text{MSlowReply}
\end{aligned}$$

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

$\text{Durability} \triangleq$

$$\begin{aligned}
& \forall \text{shardId} \in \text{Shards} : \\
& \quad \forall v1, v2 \in 0 \dots \text{MaxViews} : \\
& \quad \quad \text{If a } \text{txn} \text{ is committed in lower view (v1),} \\
& \quad \quad \text{it is impossible to make this request uncommitted in higher view} \\
& \quad \neg(\wedge v1 < v2 \\
& \quad \quad \wedge \text{ShardRecovered}(\text{shardId}, v2) \\
& \quad \quad \wedge \exists c \in \text{Coords} : \\
& \quad \quad \quad \exists \text{txnId} \in v\text{CoordTxns}[c] : \\
& \quad \quad \quad \quad \wedge \text{CommittedInView}(v1, \text{shardId}, \text{txnId}) \\
& \quad \quad \quad \quad \wedge \neg \text{CommittedInView}(v2, \text{shardId}, \text{txnId}) \\
& \quad)
\end{aligned}$$

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

$\text{Consistency} \triangleq$

$$\begin{aligned}
& \forall \text{shardId} \in \text{Shards} : \\
& \quad \forall v1, v2 \in 1 \dots \text{MaxViews} : \\
& \quad \neg(\wedge v1 < v2
\end{aligned}$$

```

    To check Consistency of txns in higher views,
    the shard should have entered the higher views
     $\wedge \text{ShardRecovered}(\text{shardId}, v2)$ 
     $\wedge \exists c \in \text{Coords} :$ 
       $\exists \text{txnId} \in v\text{CoordTxns}[c] :$ 
        Durability has been checked in another invariant
        IF  $\wedge \text{CommittedInView}(v1, \text{shardId}, \text{txnId})$ 
           $\wedge \text{CommittedInView}(v2, \text{shardId}, \text{txnId})$ 
        THEN
          LET
             $v1\text{LeaderReply} \triangleq \text{CHOOSE } m \in \text{messages} :$ 
               $\wedge m.\text{mtype} = \text{MFastReply}$ 
               $\wedge m.\text{txnId} = \text{txnId}$ 
               $\wedge m.\text{lView} = v1$ 
               $\wedge m.\text{sender.shardId} = \text{shardId}$ 
               $\wedge m.\text{sender.replicaId} = \text{LeaderID}(v1)$ 
             $v2\text{LeaderReply} \triangleq \text{CHOOSE } m \in \text{messages} :$ 
               $\wedge m.\text{mtype} = \text{MFastReply}$ 
               $\wedge m.\text{txnId} = \text{txnId}$ 
               $\wedge m.\text{lView} = v2$ 
               $\wedge m.\text{sender.shardId} = \text{shardId}$ 
               $\wedge m.\text{sender.replicaId} = \text{LeaderID}(v2)$ 
          IN
             $v1\text{LeaderReply}.hash \neq v2\text{LeaderReply}.hash$ 
          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
     $\text{allTxns} \triangleq \text{UNION } \{v\text{CoordTxns}[c] : c \in \text{Coords}\}$ 
  IN
     $\forall \text{shardId} \in \text{Shards} :$ 
       $\forall \text{txnId1}, \text{txnId2} \in \text{allTxns} :$ 
        IF  $\text{txnId1} = \text{txnId2}$  THEN TRUE
        ELSE
           $\forall v1, v2 \in 1 \dots \text{MaxViews} :$ 
            IF  $\wedge \text{CommittedInView}(v1, \text{shardId}, \text{txnId1})$ 
               $\wedge \text{CommittedInView}(v2, \text{shardId}, \text{txnId2})$ 
            THEN
              LET
                 $v1\text{LeaderReply} \triangleq \text{CHOOSE } m \in \text{messages} :$ 
                   $\wedge m.\text{mtype} = \text{MFastReply}$ 

```

```

                                ∧ m.txnId = txnId1
                                ∧ m.lView = v1
                                ∧ m.sender.shardId = shardId
                                ∧ m.sender.replicaId = LeaderID(v1)
v2LeaderReply ≜ CHOOSE m ∈ messages :
                                ∧ m.mtype = MFastReply
                                ∧ m.txnId = txnId2
                                ∧ m.lView = v2
                                ∧ m.sender.shardId = shardId
                                ∧ m.sender.replicaId = LeaderID(v2)
IN
    They cannot be committed in the same log position, regardless of the view
    v1LeaderReply.logId ≠ 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 ≜
    LET
        allTxns ≜ UNION {vCoordTxns[c] : c ∈ 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
                                IN

```

```

                                 $\wedge m.lView = v$ 
                                 $\wedge m.sender.shardId = shardId1$ 
                                 $\wedge m.sender.replicaId = LeaderID(v)$ 
     $txn1\_LeaderReplyOnShard2 \triangleq$  CHOOSE  $m \in messages :$ 
                                 $\wedge m.mtype = MFastReply$ 
                                 $\wedge m.txnId = txnId1$ 
                                 $\wedge m.lView = v$ 
                                 $\wedge m.sender.shardId = shardId2$ 
                                 $\wedge m.sender.replicaId = LeaderID(v)$ 
     $txn2\_LeaderReplyOnShard2 \triangleq$  CHOOSE  $m \in messages :$ 
                                 $\wedge m.mtype = MFastReply$ 
                                 $\wedge m.txnId = txnId2$ 
                                 $\wedge m.lView = v$ 
                                 $\wedge m.sender.shardId = shardId2$ 
                                 $\wedge m.sender.replicaId = LeaderID(v)$ 
IN
IF   $\wedge txn1\_LeaderReplyOnShard1.t = txn1\_LeaderReplyOnShard2.t$ 
     $\wedge txn2\_LeaderReplyOnShard1.t = txn2\_LeaderReplyOnShard2.t$ 
THEN
     $\vee \wedge txn1\_LeaderReplyOnShard1.logId > txn2\_LeaderReplyOnShard1.logId$ 
     $\wedge txn1\_LeaderReplyOnShard2.logId > txn2\_LeaderReplyOnShard2.logId$ 
     $\vee \wedge txn1\_LeaderReplyOnShard1.logId < txn2\_LeaderReplyOnShard1.logId$ 
     $\wedge txn1\_LeaderReplyOnShard2.logId < txn2\_LeaderReplyOnShard2.logId$ 
ELSE
    if their timestamps are not equal, our coordinator will not consider them as committed,
    We do not need to check such cases
    TRUE
ELSE TRUE

```
