MODULE *IR_consensus*

This is a TLA+ specification of the *Inconsistent* Replication algorithm. (And a mechanically-checked proof of its correctness using *TLAPS*)

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

Constants

Constant parameters: Replicas: the set of all replicas (Replica *IDs*)
Clients: the set of all clients (Client *IDs*)
Quorums: the set of all quorums *SuperQuorums*: the set of all super quorums Results: the set of all possible result types *OperationBody*: the set of all possible operation bodies (with arguments, etc. - can be infinite)
S: shard *id* of the shard *Replicas* constitute
f: maximum number of failures allowed (half of n)

Constants used to bound variables, for model checking (*Nat* is bounded) *max_vc*: maximum number of View-Changes allowed for each replicas *max_req*: maximum number of *op* requests performed by clients

CONSTANTS *Replicas*, *Clients*, *Quorums*, *SuperQuorums*, *Results*, *OpBody*,
AppClientFail, *AppReplicaFail*,
SuccessfulInconsistentOp(_, _, _), *SuccessfulConsensusOp*(_, _, _, _),
Merge(_, _, _),
Sync(_),
ExecInconsistent(_),
ExecConsensus(_),
Decide(_),
f,
S, *Shards*, $S =$ shard *id*
max_vc, *max_req*

ASSUME $IsFiniteSet(Replicas)$

ASSUME $QuorumAssumption \triangleq$
$\land\ Quorums \subseteq \text{SUBSET}\ Replicas$
$\land\ SuperQuorums \subseteq \text{SUBSET}\ Replicas$
$\land\ \forall\ Q1,\ Q2 \in Quorums : Q1 \cap Q2 \neq \{\}$
$\land\ \forall\ Q \in Quorums,\ R1,\ R2 \in SuperQuorums :$
$Q \cap R1 \cap R2 \neq \{\}$

ASSUME $FailuresAssumption \triangleq$
$\forall\ Q \in Quorums : Cardinality(Q) > f$

The possible states of a replica and the two types of operations currently defined by *IR*.

$ReplicaState \triangleq \{$"NORMAL", "FAILED", "RECOVERING", "VIEW-CHANGING"$\}$
$ClientState \triangleq \{$"NORMAL", "FAILED"$\}$

$OpType \triangleq \{\text{“Inconsistent”}, \text{“Consensus”}\}$
$OpStatus \triangleq \{\text{“TENTATIVE”}, \text{“FINALIZED”}\}$

Definition of operation space

$MessageId \triangleq [cid : Clients,\ msgid : Nat]$
$Operations \triangleq [type : OpType,\ body : OpBody] \cup \{\langle\rangle\}$

Message is defined to be the set of all possible messages

Assume unique message ids
Assume no more than f replica failures
We use shard to specify for what shard this message was
(we share the variables)

$$
\begin{array}{ll}
Message \triangleq & \\
 & [type : \{\text{“PROPOSE”}\}, \\
 & \ id : MessageId, \\
 & \ op : Operations, \\
 & \ v : Nat] \\
\cup & [type : \{\text{“REPLY”}\}, \quad \text{reply no result} \\
 & \ id : MessageId, \\
 & \ v : Nat, \\
 & \ src : Replicas] \\
\cup & \\
 & [type : \{\text{“REPLY”}\}, \quad \text{reply with result} \\
 & \ id : MessageId, \\
 & \ v : Nat, \\
 & \ res : Results, \\
 & \ src : Replicas] \\
 & \ v = \text{ view num.} \\
\cup & \\
 & [type : \{\text{“START-VIEW-CHANGE”}\}, \\
 & \ v : Nat, \\
 & \ src : Replicas] \\
\cup & \\
 & [type : \{\text{“DO-VIEW-CHANGE”}\}, \\
 & \ r : \text{SUBSET } ([msgid : MessageId, \\
 & \qquad\qquad op : Operations, \\
 & \qquad\qquad res : Results, \\
 & \qquad\qquad status : OpStatus] \\
 & \qquad \cup [msgid : MessageId, \\
 & \qquad\ op : Operations, \\
 & \qquad\ status : OpStatus]), \\
 & \ v : Nat, \\
 & \ lv : Nat,
\end{array}
$$

$src : Replicas,$
$dst : \text{SUBSET } Replicas]$

$\cup$

$[type : \{\text{“START-VIEW”}\},$
$v : Nat,$
$r : \text{SUBSET } ([msgid : MessageId,$
$op : Operations,$
$res : Results,$
$status : OpStatus]$
$\cup\, [msgid : MessageId,$
$op : Operations,$
$status : OpStatus]),$
$src : Replicas]$

$\cup$

$[type : \{\text{“FINALIZE”}\},$ finalize with no result
$id : MessageId,$
$op : Operations,$
$res : Results,$
$v : Nat]$

$\cup$

$[type : \{\text{“FINALIZE”}\},$ finalize with result
$id : MessageId,$
$op : Operations,$
$res : Results,$
$v : Nat]$

$\cup$

$[type : \{\text{“CONFIRM”}\},$
$v : Nat,$
$id : MessageId,$
$op : Operations,$
$res : Results,$
$src : Replicas]$

Variables and State Predicates

* Variables: 1. State at each replica:
rState = Denotes current replica state. Either:
- *NORMAL* (processing operations)
- VIEW-CHANGING (participating in recovery)

rRecord = Unordered set of operations and their results *rViewNumber* = current view number

2. State of communication medium: *sentMsg* = sent (but not yet received) messages
3. State at client: *cCurrentOperation* = *crt* operation requested by the client *cMmessageCounter* = the message I must use for the next operation

```
* \

VARIABLES rState, rRecord, rCrtView, rLastView, rViewReplies,
          rViewOnDisk,
          rNonce, sentMsg, cCrtOp,
          cCrtOpToFinalize, cMsgCounter, cCrtOpReplies, cCrtOpConfirms,
          cState, aSuccessful, arRecord, aVisibility, gViewChangesNo

Defining these tuples makes it easier to express which varibles remain unchanged

 Replica variables.
rVars ≜ ⟨rState, rRecord, rCrtView, rViewOnDisk, rLastView,
          rViewReplies, rNonce⟩
 Client variables.
cVars ≜ ⟨cCrtOp,            current operation at a client
          cCrtOpToFinalize,
          cCrtOpReplies,    current operation replies
          cCrtOpConfirms,
          cMsgCounter,
          cState⟩
 Application variables.
aVars ≜ ⟨aSuccessful, arRecord, aVisibility⟩   we use them to write invariants
 Other variables.
oVars ≜ ⟨sentMsg, gViewChangesNo⟩
 All variables.
vars ≜ ⟨rVars, cVars, aVars, oVars⟩

TypeOK ≜
  ∧ rState[S] ∈ [Replicas → ReplicaState]
  ∧ rRecord[S] ∈ [Replicas → SUBSET ([msgid : MessageId,
                                      op : Operations,
                                      res : Results,
                                      status : OpStatus]
                                    ∪ [msgid : MessageId,
                                       op : Operations,
                                       status : OpStatus])]
  ∧ rCrtView[S] ∈ [Replicas → Nat]
  ∧ rViewOnDisk[S] ∈ [Replicas → Nat]
  ∧ rLastView[S] ∈ [Replicas → Nat]
  ∧ rViewReplies[S] ∈ [Replicas → SUBSET ([type : {"start-view-change"},
                                           v : Nat,
                                           src : Replicas]
                                         ∪ [type : {"do-view-change"},
                                            v : Nat,
                                            lv : Nat,
                                            r : SUBSET ([msgid : MessageId,
```

```
                                                  op : Operations,
                                                  res : Results,
                                                  status : OpStatus]
                                             ∪ [msgid : MessageId,
                                                op : Operations,
                                                status : OpStatus]),
                                     src : Replicas])]
  ∧ rNonce[S] ∈ [Replicas → Nat]
  ∧ sentMsg[S] ∈ SUBSET Message
  ∧ cCrtOp[S] ∈ [Clients → Operations]
  ∧ cCrtOpToFinalize[S] ∈ [Clients → Operations]
  ∧ cCrtOpReplies[S] ∈ [Clients → SUBSET ([viewNumber : Nat,
                                            res : Results,
                                            src : Replicas]
                                      ∪    [viewNumber : Nat,
                                            src : Replicas])]
  ∧ cCrtOpConfirms[S] ∈ [Clients → SUBSET [viewNumber : Nat,
                                            res : Results,
                                            src : Replicas]]
  ∧ cMsgCounter[S] ∈ [Clients → Nat]
  ∧ cState ∈ [Clients → ClientState]
  ∧ aSuccessful ∈ SUBSET ([mid : MessageId,
                            op  : Operations,
                            res : Results]
                       ∪ [mid  : MessageId,
                          op   : Operations])
  ∧ aVisibility[S] ∈ [MessageId → SUBSET MessageId]
  ∧ arRecord[S] ∈ [Replicas → SUBSET ([msgid : MessageId,
                                        op : Operations,
                                        res : Results,
                                        status : OpStatus]
                                  ∪    [msgid : MessageId,
                                        op : Operations,
                                        status : OpStatus])]
  ∧ gViewChangesNo[S] ∈ Nat

Init ≜
  ∧ rState = [r ∈ Replicas ↦ "NORMAL"]
  ∧ rRecord = [r ∈ Replicas ↦ {}]
  ∧ rCrtView = [r ∈ Replicas ↦ 0]
  ∧ rViewOnDisk = [r ∈ Replicas ↦ 0]
  ∧ rLastView = [r ∈ Replicas ↦ 0]
  ∧ rViewReplies = [r ∈ Replicas ↦ {}]
  ∧ rNonce = [r ∈ Replicas ↦ 0]
  ∧ sentMsg = {}
```

$\land\ cCrtOp = [c \in Clients \mapsto \langle\rangle]$
$\land\ cCrtOpToFinalize = [c \in Clients \mapsto \langle\rangle]$
$\land\ cCrtOpReplies = [c \in Clients \mapsto \{\}]$
$\land\ cCrtOpConfirms = [c \in Clients \mapsto \{\}]$
$\land\ cMsgCounter = [c \in Clients \mapsto 0]$
$\land\ cState = [c \in Clients \mapsto \text{“NORMAL”}]$
$\land\ aSuccessful = \{\}$
$\land\ aVisibility = [o \in MessageId \mapsto \{\}]$
$\land\ arRecord = [r \in Replicas \mapsto \{\}]$
$\land\ gViewChangesNo = 0$

Actions

$Send(m) \triangleq sentMsg' = [sentMsg \text{ EXCEPT } ![S] = @ \cup \{m\}]$

$AmLeader(r, v) \triangleq r = (v\%Cardinality(Replicas)) + 1$
$IsLeader(r, v) \triangleq AmLeader(r, v)$
$NotIsLeader(r, v) \triangleq r \neq (v\%Cardinality(Replicas)) + 1$
$LeaderOf(v) \triangleq \text{CHOOSE } x \in Replicas : IsLeader(x, v)$

Client Actions

Note: CHOOSE does not introduce nondeterminism (the same value is chosen each time)

* Client sends a request
$ClientRequest(c, op) \triangleq$
$\land\ cCrtOp[S][c] = \langle\rangle$ the client is not waiting for a result of another operation
$\land\ cCrtOpToFinalize[S][c] = \langle\rangle$ the client is not waiting to finalize operation
$\land\ cMsgCounter' = [cMsgCounter \text{ EXCEPT } ![S][c] = @ + 1]$
$\land\ cCrtOp' = [cCrtOp \text{ EXCEPT } ![S][c] = op]$
$\land\ Send([type \mapsto \text{“PROPOSE”},$
$id \mapsto [cid \mapsto c, msgid \mapsto cMsgCounter[S][c] + 1],$
$op \mapsto op,$
$v \mapsto 0])$
$\land\ \text{UNCHANGED } \langle rVars, aVars, cCrtOpReplies, cCrtOpToFinalize, cCrtOpConfirms, cState, gViewChangesNo\rangle$
$\land\ cMsgCounter[S][c] < max\_req$ BOUND the number of requests a client can make (useful for model checking)

* Client received a reply
$ClientReceiveReply(c) \triangleq$

```
  ∃ msg ∈ sentMsg[S] :
    ∧ msg.type = "REPLY"
    ∧ cCrtOp[S][c] ≠ ⟨⟩
    ∧  reply to c's request for crt op
       msg.id = [cid ↦ c, msgid ↦ cMsgCounter[S][c]]
    ∧ ∨ ∧ cCrtOp[S][c].type = "Inconsistent"
         ∧ cCrtOpReplies' = [cCrtOpReplies EXCEPT ![S][c] = @ ∪
                                                  {[viewNumber ↦ msg.v,
                                                    src        ↦ msg.src]}]
      ∨ ∧ cCrtOp[S][c].type = "Consensus"
         ∧ cCrtOpReplies' = [cCrtOpReplies EXCEPT ![S][c] = @ ∪
                                                  {[viewNumber ↦ msg.v,
                                                    res        ↦ msg.res,
                                                    src        ↦ msg.src]}]
    ∧ UNCHANGED ⟨cCrtOp, cCrtOpToFinalize, cCrtOpConfirms,
                 cMsgCounter, cState, rVars, aVars, oVars⟩

  \* "Helper" formulas
__matchingViewNumbers(Q, c) ≜
      a (super)quorum of replies with matching view numbers
    ∧ ∀ r ∈ Q :
        ∧ ∃ reply ∈ cCrtOpReplies[S][c]: reply.src = r
      ∧ ∀ p ∈ Q : ∃ rr, pr ∈ cCrtOpReplies[S][c] :
                    ∧ rr.src = r
                    ∧ pr.src = p
                    ∧ rr.viewNumber = pr.viewNumber

__matchingViewNumbersAndResults(Q, c) ≜
      a (super)quorum of replies with matching view numbers
      and results
    ∧ ∀ r ∈ Q :
        ∧ ∃ reply ∈ cCrtOpReplies[S][c]: reply.src = r
      ∧ ∀ p ∈ Q : ∃ rr, pr ∈ cCrtOpReplies[S][c] :
                    ∧ rr.src = r
                    ∧ pr.src = p
                    ∧ rr.viewNumber = pr.viewNumber
                    ∧ rr.res = pr.res

  \* IR Client received enough responses to decide
  \* what to do with the operation
ClientSendFinalize(c) ≜
  ∧ cCrtOp[S][c] ≠ ⟨⟩
  ∧ ∨ ∃ Q ∈ Quorums :
      I. The IR Client got a simple quorum of replies
        ∧ ∀ r ∈ Q :
            ∃ reply ∈ cCrtOpReplies[S][c] : reply.src = r
```

$\land \lor \land cCrtOp[S][c].type = \text{“Inconsistent”}$
$\land \_\_matchingViewNumbers(Q,\ c)$
$\land aSuccessful' = aSuccessful \cup$
$\{[mid \mapsto [cid \mapsto c,$
$msgid \mapsto cMsgCounter[S][c]],$
$op \mapsto cCrtOp[S][c]]\}$
$\land SuccessfulInconsistentOp(c,\ S,\ cCrtOp[S][c])$
$\land Send([type \mapsto \text{“FINALIZE”},$
$id \mapsto [cid \mapsto c,\ msgid \mapsto cMsgCounter[S][c]],$
$op \mapsto cCrtOp[S][c],$
$v \mapsto 0])$
$\land$ UNCHANGED $\langle cCrtOpToFinalize \rangle$

$\lor \land cCrtOp[S][c].type = \text{“Consensus”}$
$\land \_\_matchingViewNumbers(Q,\ c)$
$\land$ LET $res \triangleq$ IF $\_\_matchingViewNumbersAndResults(Q,\ c)$
THEN
CHOOSE $result \in$
$\{res \in Results :$
$\exists\, reply \in cCrtOpReplies[S][c] :$
$\land reply.src \in Q$
$\land reply.res = res\}$: TRUE
ELSE
$Decide(cCrtOpReplies[S][c])$
IN
$\land Send([type \mapsto \text{“FINALIZE”},$
$id \mapsto [cid \mapsto c,\ msgid \mapsto cMsgCounter[S][c]],$
$op \mapsto cCrtOp[S][c],$
$res \mapsto res,$
$v \mapsto 0])$
$\land cCrtOpToFinalize' = [cCrtOp$ EXCEPT $![S][c] = cCrtOp[S][c]]$
$\land$ UNCHANGED $\langle aSuccessful \rangle$

$\lor \exists\, SQ \in SuperQuorums :$
II. The *IR* Client got super quorum of responses
$\land \forall\, r \in SQ :$
$\exists\, reply \in cCrtOpReplies[S][c] : reply.src = r$
$\land cCrtOp[S][c].type = \text{“Consensus”}$ only care if consensus *op*
$\land \_\_matchingViewNumbersAndResults(SQ,\ c)$
$\land$ LET $res \triangleq$ CHOOSE $result \in$
$\{res \in Results :$
$\exists\, reply \in cCrtOpReplies[S][c] :$
$\land reply.src \in SQ$
$\land reply.res = res\}$: TRUE
IN

```
                    ∧ Send([type ↦ "FINALIZE",
                            id   ↦ [cid ↦ c, msgid ↦ cMsgCounter[S][c]],
                            op   ↦ cCrtOp[S][c],
                            res  ↦ res,
                            v    ↦ 0])
                    ∧ aSuccessful' = aSuccessful ∪
                                     {[mid ↦ [cid ↦ c,
                                              msgid ↦ cMsgCounter[S][c]],
                                       op  ↦ cCrtOp[S][c],
                                       res ↦ res]}
                    ∧ SuccessfulConsensusOp(c, S, cCrtOp[S][c], res)
                ∧ UNCHANGED ⟨cCrtOpToFinalize⟩
            ∧ cCrtOp' = [cCrtOp EXCEPT ![S][c] = ⟨⟩]
            ∧ cCrtOpReplies' = [cCrtOpReplies EXCEPT ![S][c] = {}]
            ∧ UNCHANGED ⟨cMsgCounter, cState, cCrtOpConfirms, rVars, arRecord, aVisibility, gViewChangesNo⟩

        \* Client received a confirm
    ClientReceiveConfirm(c) ≜
        ∃ msg ∈ sentMsg[S] :
            ∧ msg.type = "CONFIRM"
            ∧ cCrtOpToFinalize[S][c] ≠ ⟨⟩
            ∧ msg.id = [cid ↦ c, msgid ↦ cMsgCounter[S][c]]  reply to c's request for crt op
            ∧ cCrtOpConfirms' = [cCrtOpConfirms EXCEPT ![S][c] = @ ∪
                                                    {[viewNumber ↦ msg.v,
                                                      res        ↦ msg.res,
                                                      src        ↦ msg.src]}]
            ∧ UNCHANGED ⟨cCrtOp, cCrtOpReplies, cCrtOpToFinalize, cMsgCounter,
                        cState, rVars, aVars, oVars⟩


        \* An operation is finalized by a client and result returned to the application
    ClientFinalizeOp(c) ≜
        ∧ cCrtOpToFinalize[S][c] ≠ ⟨⟩
        ∧ ∃ Q ∈ Quorums :
            IR client received a quorum of confirms
            ∧ ∀ r ∈ Q :
                ∃ reply ∈ cCrtOpConfirms[S][c] : reply.src = r
            ∧ LET
                  take the result in the biggest view number
                reply ≜ CHOOSE reply ∈ cCrtOpConfirms[S][c] :
                                        ¬∃ rep ∈ cCrtOpConfirms[S][c] :
                                            rep.viewNumber > reply.viewNumber
              IN
                ∧ aSuccessful' = aSuccessful ∪
                                 {[mid ↦ [cid ↦ c,
                                          msgid ↦ cMsgCounter[S][c]],
```

```
                                    op ↦ cCrtOpToFinalize[S][c],
                                    res ↦ reply.res]}
          ∧ SuccessfulConsensusOp(c, S, cCrtOp[S][c], reply.res)   respond to app
   ∧ cCrtOpToFinalize' = [cCrtOpToFinalize EXCEPT ![S][c] = ⟨⟩]
   ∧ cCrtOpConfirms' = [cCrtOpConfirms EXCEPT ![S][c] = {}]
   ∧ UNCHANGED ⟨rVars, cCrtOp, cCrtOpReplies,
                cMsgCounter, cState, arRecord, aVisibility, oVars⟩

 \* Client fails and looses all data
ClientFail(c) ≜
   ∧ cState' = [cState EXCEPT ![S][c] = "FAILED"]
   ∧ cMsgCounter' = [cMsgCounter EXCEPT ![S][c] = 0]
   ∧ cCrtOp' = [cCrtOp EXCEPT ![S][c] = ⟨⟩]
   ∧ cCrtOpReplies' = [cCrtOpReplies EXCEPT ![S][c] = {}]
   ∧ AppClientFail
   ∧ UNCHANGED ⟨rVars, aVars, oVars⟩

 \* Client recovers
 \* Not implemented yet
ClientRecover(c) ≜ FALSE
```

Replica Actions

```
 \* Replica sends a reply
ReplicaReceiveRequest(r) ≜
  ∃ msg ∈ sentMsg[S] :
    ∧ msg.type = "PROPOSE"
    ∧  not already replied for this op
      ¬(∃ rec ∈ rRecord[S][r] : rec.msgid = msg.id)
    ∧ ∨ ∧ msg.op.type = "Inconsistent"
        ∧ Send([type ↦ "REPLY",
                id   ↦ msg.id,
                v    ↦ rCrtView[S][r],
                src  ↦ r])
        ∧ rRecord' = [rRecord EXCEPT ![S][r] =
                        @ ∪ {[msgid ↦ msg.id,
                              op ↦ msg.op,
                              status ↦ "TENTATIVE"]}]
      ∨ ∧ msg.op.type = "Consensus"
        ∧ LET res ≜ ExecConsensus(msg.op)
          IN
            ∧ Send([type ↦ "REPLY",
                    id   ↦ msg.id,
                    v    ↦ rCrtView[S][r],
                    res  ↦ res,
```

```
                        src   ↦ r])
                ∧ rRecord′ = [rRecord EXCEPT ![S][r] =
                                 @ ∪ {[msgid ↦ msg.id,
                                       op ↦ msg.op,
                                       res ↦ res,
                                       status ↦ "TENTATIVE"]}]
    ∧ UNCHANGED ⟨rState, rCrtView, rLastView, rViewOnDisk,
                 rViewReplies, rNonce, cVars, aVars, gViewChangesNo⟩

\* Replica receives a message from an IR Client to finalize an op
\* For inconsistent oprations the replica just
\* executes the operation.
ReplicaReceiveFinalize(r) ≜
  ∃ msg ∈ sentMsg[S] :
    ∧ msg.type = "FINALIZE"
    ∧ msg.v ≥ rCrtView[S][r]
    ∧ LET recs ≜ {rec ∈ rRecord[S][r] : Must be only 1 record
                                        ∧ rec.msgid = msg.id
                                        ∧ rec.op = msg.op}
      IN
        ∨ ∧ msg.op.type = "Inconsistent"
          ∧ IF
                Replica knows of this op
              recs ≠ {}
            THEN
              IF ∀ rec ∈ recs : rec.status ≠ "FINALIZED"
              THEN ExecInconsistent(msg.op)
              ELSE TRUE
            ELSE
                Replica didn't hear of this op
              ExecInconsistent(msg.op)
          ∧ rRecord′ = [rRecord EXCEPT ![S][r] = (@ \ recs) ∪
                                                 {[msgid ↦ msg.id,
                                                   op ↦ msg.op,
                                                   status ↦ "FINALIZED"]}]
          ∧ UNCHANGED ⟨sentMsg⟩
        ∨ ∧ msg.op.type = "Consensus"
          ∧ rRecord′ = [rRecord EXCEPT ![S][r] = (@ \ recs) ∪
                                                 {[msgid ↦ msg.id,
                                                   op ↦ msg.op,
                                                   res ↦ msg.res,
                                                   status ↦ "FINALIZED"]}]
          ∧ Send([type ↦ "CONFIRM",
                  v    ↦ rCrtView[S][r],
                  id   ↦ msg.id,
```

$op \mapsto msg.op,$
$res \mapsto msg.res,$
$src \mapsto r])$
$\land$ UNCHANGED $\langle rState,\ rCrtView,\ rLastView,\ rViewReplies,\ rViewOnDisk,$
$rNonce,\ cVars,\ aVars,\ gViewChangesNo\rangle$

* A recovering replica starts the view change procedure
$ReplicaSendDoViewChange(r) \triangleq$
$\land \lor \land rState[S][r] =$ "NORMAL" $\lor rState[S][r] =$ "VIEW-CHANGING"
$\land rCrtView' = [rCrtView$ EXCEPT $![S][r] = @ + 1]$
$\land rViewOnDisk' = [rViewOnDisk$ EXCEPT $![S][r] = rCrtView[S][r] + 1]$
$\land rState' = [rState$ EXCEPT $![S][r] =$ "VIEW-CHANGING"$]$
$\land Send([type \mapsto$ "DO-VIEW-CHANGE",
$v \mapsto rCrtView[S][r] + 1,$
$lv \mapsto rLastView[S][r],$
$r \mapsto rRecord[S][r],$
$src \mapsto r,$
$dst \mapsto Replicas])$
$\lor \land rState[S][r] =$ "FAILED"
$\land rState' = [rState$ EXCEPT $![S][r] =$ "RECOVERING"$]$
$\land rCrtView' = [rCrtView$ EXCEPT $![S][r] = rViewOnDisk[S][r]]$
$\land Send([type \mapsto$ "DO-VIEW-CHANGE",
$v \mapsto rViewOnDisk[S][r],$
$lv \mapsto rLastView[S][r],$
$r \mapsto rRecord[S][r],$
$src \mapsto r,$
$dst \mapsto Replicas \setminus \{x \in Replicas : IsLeader(x,\ rViewOnDisk[S][r])\}])$
$\land$ UNCHANGED $\langle rViewOnDisk\rangle$
$\lor \land rState[S][r] =$ "RECOVERING"
$\land rCrtView' = [rCrtView$ EXCEPT $![S][r] = @ + 1]$
$\land Send([type \mapsto$ "DO-VIEW-CHANGE",
$v \mapsto rCrtView[S][r] + 1,$
$lv \mapsto rLastView[S][r],$
$r \mapsto rRecord[S][r],$
$src \mapsto r,$
$dst \mapsto Replicas \setminus \{x \in Replicas : IsLeader(x,\ rCrtView[S][r] + 1)\}])$
$\land$ UNCHANGED $\langle rViewOnDisk,\ rState\rangle$
$\land$ UNCHANGED $\langle cVars,\ rLastView,\ rViewReplies,\ rRecord,$
$rNonce,\ aVars\rangle$
$\land gViewChangesNo[S] < max\_vc$ BOUND on number of view changes
$\land gViewChangesNo' = [gViewChangesNo$ EXCEPT $![S] = @ + 1]$

* Replica received DO-VIEW-CHANGE message
$ReplicaReceiveDoViewChange(r) \triangleq$
$\land \exists\, msg \in sentMsg[S] :$

$\land\ msg.type = \text{“DO-VIEW-CHANGE”}$
$\land\ r \in msg.dst$
$\land\ \lor\ \land\ rState[S][r] = \text{“NORMAL”}$
$\quad\quad\land\ msg.v > rCrtView[S][r]$
$\quad\lor\ \land\ rState[S][r] = \text{“VIEW-CHANGING”}$
$\quad\quad\land\ msg.v \geq rCrtView[S][r]$
$\land\ rState' = [rState \text{ EXCEPT } ![S][r] = \text{“VIEW-CHANGING”}]$
keep only the one with the higher view (v)
$\land\ \lor\ \land\ IsLeader(r,\ msg.v)$
$\quad\quad\land\ \text{LET}$
$\quad\quad\quad existingRecord \triangleq \{x \in rViewReplies[S][r] :$
$\quad\quad\quad\quad\land\ x.type = \text{“do-view-change”}$
$\quad\quad\quad\quad\land\ x.src = msg.src\}$ should only be one item in set
$\quad\quad\text{IN}$
$\quad\quad\quad\text{IF } \forall\, x \in existingRecord : x.v < msg.v$
$\quad\quad\quad\text{THEN } rViewReplies' = [rViewReplies \text{ EXCEPT } ![S][r] =$
$\quad\quad\quad\quad(@ \setminus existingRecord) \cup$
$\quad\quad\quad\quad\{[type \mapsto \text{“do-view-change”},$
$\quad\quad\quad\quad\ v \mapsto msg.v,$
$\quad\quad\quad\quad\ lv \mapsto msg.lv,$
$\quad\quad\quad\quad\ r \mapsto msg.r,$
$\quad\quad\quad\quad\ src \mapsto msg.src]\}]$
$\quad\quad\quad\text{ELSE FALSE}$
$\quad\lor\ \text{UNCHANGED } \langle rViewReplies\rangle$
$\land\ rCrtView' = [rCrtView \text{ EXCEPT } ![S][r] = msg.v]$
$\land\ rViewOnDisk' = [rViewOnDisk \text{ EXCEPT } ![S][r] = msg.v]$
$\land\ Send([type \mapsto \text{“DO-VIEW-CHANGE”},$
$\quad\quad v \mapsto msg.v,$
$\quad\quad lv \mapsto rLastView[S][r],$
$\quad\quad r \mapsto rRecord[S][r],$
$\quad\quad src \mapsto r,$
$\quad\quad dst \mapsto Replicas])$
$\land\ \text{UNCHANGED } \langle cVars,\ rLastView,\ rRecord,\ rNonce,$
$\quad\quad aVars,\ gViewChangesNo\rangle$

Note: Assume one reply for view change per replica in Q
(in *ReplicaReceiveDoViewChange* we keep only the most recent reply)
$ReplicaSendStartView(r) \triangleq$
$\land\ \exists\, Q \in Quorums :$
$\quad\land\ \forall\, r1 \in Q :$
$\quad\quad\land\ \forall\, r2 \in Q : \exists\, rr,\ pr \in rViewReplies[S][r] :$
$\quad\quad\quad\quad\land\ rr.type = \text{“do-view-change”}$
$\quad\quad\quad\quad\land\ pr.type = \text{“do-view-change”}$
$\quad\quad\quad\quad\land\ rr.src = r1$
$\quad\quad\quad\quad\land\ pr.src = r2$

$\land\ rr.v = pr.v$
$\land\ rr.v \geq rCrtView[S][r]$

received at a least a quorum of replies

$\land$ LET

$A \triangleq$

set of all do-view-change replies from Q,

$\{x \in rViewReplies[S][r] : \land\ x.src \in Q$
$\land\ x.type = \text{“do-view-change”}\}$

$B \triangleq$

keep only the replies from the maximum view

$\{x \in A : \forall\, y \in A : y.lv \leq x.lv\}$

$C \triangleq$

set of all records received in replies in B

UNION $\{x.r : x \in B\}$

$recoveredConsensusOps\_R \triangleq$

any finalized consensus operation (in at least one record,
in the maximum latest view)

$\{[msgid \mapsto y.msgid,\ op \mapsto y.op,\ res \mapsto y.res] : y \in$
$\{x \in C :$
$\land\ x.op.type = \text{“Consensus”}$
$\land\ x.status = \text{“FINALIZED”}\}\}$

$recoveredConsensusOps\_d \triangleq$

any consensus operation found in at least a majority of a Quorum

$\{[msgid \mapsto y.msgid,\ op \mapsto y.op,\ res \mapsto y.res] : y \in$
$\{x \in C :$
$\land\ x.op.type = \text{“Consensus”}$
$\land\ x.status = \text{“TENTATIVE”}$
$\land\ \exists\, P \in SuperQuorums :$
$\forall\, replica \in Q \cap P :$
$\exists\, reply \in B :$
$\land\ reply.src = replica$
$\land\ x \in reply.r\}\} \setminus recoveredConsensusOps\_R$

$recoveredConsensusOps\_u \triangleq$

the rest of consensus ops found in at least one record
(discard the result)

$\{[msgid \mapsto z.msgid,\ op \mapsto z.op] : z \in$
$((\{[msgid \mapsto y.msgid,\ op \mapsto y.op,\ res \mapsto y.res] : y \in$
$\{x \in C : x.op.type = \text{“Consensus”}\}\}$
$\setminus recoveredConsensusOps\_d) \setminus recoveredConsensusOps\_R)\}$

$recoveredInconsistentOps\_R \triangleq$

any inconsistent operation found in any received record

```
        {[msgid ↦ y.msgid, op ↦ y.op] : y ∈
          {x ∈ C : x.op.type = "Inconsistent"}}

        mergedRecordInconsistent ≜
          {x ∈ Merge(recoveredConsensusOps_R
                  ∪ recoveredInconsistentOps_R,
                recoveredConsensusOps_d,
                recoveredConsensusOps_u) : x.op.type = "Inconsistent"}

        mergedRecordConsensus ≜
          {x ∈ Merge(recoveredConsensusOps_R
                  ∪ recoveredInconsistentOps_R,
                recoveredConsensusOps_d,
                recoveredConsensusOps_u) : x.op.type = "Consensus"}

        masterRecord ≜
          {[msgid ↦ x.msgid,
            op ↦ x.op,
            status ↦ "FINALIZED"] :
              x ∈ mergedRecordInconsistent}
        ∪
          {[msgid ↦ x.msgid,
            op ↦ x.op,
            res ↦ x.res,
            status ↦ "FINALIZED"] :
              x ∈ mergedRecordConsensus}

        v_new ≜
            the one decided by quorum Q
          CHOOSE v ∈ {x.v : x ∈ A} : TRUE
    IN
        ∧ rRecord' = [rRecord EXCEPT ![S][r] = masterRecord]
        ∧ Sync(masterRecord)
        ∧ Send([type ↦ "START-VIEW",
                v    ↦ v_new,
                r    ↦ masterRecord,
                src  ↦ r])
        ∧ rCrtView' = [rCrtView EXCEPT ![S][r] = v_new]
        ∧ rLastView' = [rLastView EXCEPT ![S][r] = v_new]
          ∧ Assert(Cardinality(masterRecord) = 0, "Should fail - ReplicaSendStartView")
    ∧ rState' = [rState EXCEPT ![S][r] = "NORMAL"]
    ∧ rViewReplies' = [rViewReplies EXCEPT ![S][r] = {}]
    ∧ UNCHANGED ⟨rNonce, rViewOnDisk, cVars, aVars, gViewChangesNo⟩

  \* A replica receives a start view message
ReplicaReceiveStartView(r) ≜
```

```
  ∧ ∃ msg ∈ sentMsg[S] :
    ∧ msg.type = "START-VIEW"
    ∧ ∨ ∧ rState[S][r] = "NORMAL"
         ∧ msg.v > rCrtView[S][r]
      ∨ ∧ ∨ rState[S][r] = "VIEW-CHANGING"
           ∨ rState[S][r] = "RECOVERING"
         ∧ msg.v ≥ rCrtView[S][r]
    ∧ rCrtView' = [rCrtView EXCEPT ![S][r] = msg.v]
    ∧ rLastView' = [rLastView EXCEPT ![S][r] = msg.v]
    ∧ rRecord' = [rRecord EXCEPT ![S][r] = msg.r]
    ∧ Sync(msg.r)
    ∧  Check if the operations received in the master record
       must be added to the aSuccessful
       LET
         successfulOps ≜ {x ∈ msg.r : ∃ Q ∈ Quorums :
                                        ∀ r1 ∈ Q :
                                          ∨ x ∈ rRecord[S][r1]
                                          ∨ x ∈ arRecord[S][r1]
                                          ∨ r1 = r}
       IN
         aSuccessful' = aSuccessful ∪
                        {[mid ↦ x.msgid,
                          op  ↦ x.op,
                          res ↦ x.res] :
                            x ∈
                              {y ∈ successfulOps :
                                 y.op.type = "Consensus"}}
                        ∪
                        {[mid ↦ x.msgid,
                          op  ↦ x.op] :
                            x ∈
                              {y ∈ successfulOps :
                                 y.op.type = "Inconsistent"}}
    ∧ rViewOnDisk' = [rViewOnDisk EXCEPT ![S][r] = msg.v + 1]
  ∧ rState' = [rState EXCEPT ![S][r] = "NORMAL"]
  ∧ rViewReplies' = [rViewReplies EXCEPT ![S][r] = {}]
  ∧ UNCHANGED ⟨rNonce, cVars, arRecord, aVisibility, oVars⟩

 \* A replica fails and looses everything except the view number
 \* The view number has been written to disk
ReplicaFail(r) ≜
  ∧ rState' = [rState EXCEPT ![S][r] = "FAILED"]
  ∧ rRecord' = [rRecord EXCEPT ![S][r] = {}]
  ∧ arRecord' = [arRecord EXCEPT ![S][r] = rRecord[S][r]]
                                                  save record only for
```

invariant purposes

$\land\ rCrtView' = [rCrtView \text{ EXCEPT } ![S][r] = 0]$
$\land\ rLastView' = [rLastView \text{ EXCEPT } ![S][r] = 0]$
$\land\ rViewReplies' = [rViewReplies \text{ EXCEPT } ![S][r] = \{\}]$
$\land$ UNCHANGED $\langle rViewOnDisk,\ rNonce,\ cVars,\ aSuccessful,\ aVisibility,\ oVars\rangle$
$\land$ We assume less than f replicas fail simultaneously
$Cardinality(\{re \in Replicas :$
$\lor\ rState[S][re] =$ "FAILED"
$\lor\ rState[S][re] =$ "RECOVERING"$\}) < f$

High-Level Actions

$ClientAction(c) \triangleq$
$\lor\ \land\ cState[c] =$ "NORMAL"
$\land$ $\lor\ ClientRequest(c)$ * some client tries to replicate commit an operation
$\lor\ ClientReceiveReply(c)$ some client receives a reply from a replica
$\lor\ ClientReceiveConfirm(c)$ some client receives a confirm from a replica
$\lor\ ClientFail(c)$ * some client fails
$\lor\ ClientSendFinalize(c)$ an operation is successful at some client
$\lor\ ClientFinalizeOp(c)$ an operation was finalized at some client
$\lor\ \land\ cState[c] =$ "FAILED"
$\land\ \lor\ ClientRecover(c)$

$ReplicaAction(r) \triangleq$
$\lor\ \land\ rState[S][r] =$ "NORMAL"
$\land\ \lor\ ReplicaReceiveRequest(r)$ some replica sends a reply to a *PROPOSE msg*
$\lor\ ReplicaReceiveFinalize(r)$
$\lor\ ReplicaSendDoViewChange(r)$
$\lor\ ReplicaReceiveDoViewChange(r)$
$\lor\ ReplicaReceiveStartView(r)$
$\lor\ ReplicaFail(r)$ some replica fails
$\lor\ \land\ rState[S][r] =$ "FAILED"
$\land\ \lor\ ReplicaSendDoViewChange(r)$ start view-change protocol
$\lor\ \land\ rState[S][r] =$ "RECOVERING"
$\land\ \lor\ ReplicaSendDoViewChange(r)$ re-start view-change protocol (assume a timeout and still no response from the new leader)
$\lor\ ReplicaReceiveStartView(r)$
$\lor\ \land\ rState[S][r] =$ "VIEW-CHANGING"
$\land\ \lor\ ReplicaSendDoViewChange(r)$
$\lor\ ReplicaReceiveDoViewChange(r)$
$\lor\ ReplicaSendStartView(r)$
$\lor\ ReplicaReceiveStartView(r)$
$\lor\ ReplicaFail(r)$

$Next \triangleq$

$\lor \exists\, c \in \mathit{Clients} : \mathit{ClientAction}(c)$
$\lor \exists\, r \in \mathit{Replicas} : \mathit{ReplicaAction}(r)$
$\lor$ * Avoid deadlock by termination
$(\forall\, i \in 1 \,..\, \mathit{Cardinality}(\mathit{Replicas}):\ \mathit{rLastView}[S][i] = \mathit{max\_vc})\ \lor$ UNCHANGED $\langle \mathit{vars} \rangle$

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

$\mathit{FaultTolerance} \triangleq$
$\land \forall\, \mathit{successfulOp} \in \mathit{aSuccessful},\ Q \in \mathit{Quorums} :$
$(\forall\, r \in Q : \mathit{rState}[S][r] = \text{“NORMAL”} \lor \mathit{rState}[S][r] = \text{“VIEW-CHANGING”})$
$\Rightarrow (\exists\, p \in Q : \exists\, \mathit{rec} \in \mathit{rRecord}[S][p] :$
$\land\ \mathit{successfulOp.mid} = \mathit{rec.msgid}$
$\land\ \mathit{successfulOp.op} = \mathit{rec.op})$ Not necessarily same result

* Modification History
* Last modified *Thu Feb* 15 10:51:10 *PST* 2018 by *o*80053527
* Last modified *Fri Aug* 28 10:58:02 *PDT* 2015 by *aaasz*
* Created *Fri Dec* 12 17:42:14 *PST* 2014 by *aaasz*