MODULE *voldchain*

/*Replicated storage protocol with chain replication.

EXTENDS *Integers*, *Sequences*, *FiniteSets*, *TLC*

CONSTANTS N, C, $STOP$, $FAILNUM$

ASSUME $N - FAILNUM \geq 1 \land STOP < 5 \land 0 \leq FAILNUM \land FAILNUM \leq 2$

$Nodes \triangleq 1 \,..\, N$

$Clients \triangleq N + 1 \,..\, N + C$ /*should give different *ID* space to Client

$Procs \triangleq 1 \,..\, N + C$

$Configurator \triangleq N + C + 1$ /**Configurator* is unfallable

```
--algorithm voldchain{
    variable FailNum = FAILNUM,
             msg = [j ∈ Procs ↦ ⟨"X", −1, −1, "X", −1⟩],   /*default message
             up = [n ∈ Nodes ↦ TRUE],
             db = [n ∈ Nodes ↦ [ver ↦ −1, val ↦ −1, cli ↦ −1]],
              /*db is single record only
             chain = ⟨1⟩,
              /*chain is a sequence initially empty
             status = "Writing" ;

    define {
    UpNodes ≜ {n ∈ Nodes : up[n] = TRUE}
    InChain(s) ≜ ∃ i ∈ 1 .. Len(chain) : chain[i] = s
    ChainNodes ≜ {chain[j] : j ∈ 1 .. Len(chain)}
    FreeUpNode ≜ IF (UpNodes \ ChainNodes) ≠ {} THEN
                   CHOOSE i ∈ (UpNodes \ ChainNodes) : i > 0
                 ELSE 0
    GetIndex(s) ≜ CHOOSE i ∈ 1 .. Len(chain) : chain[i] = s
     /*Assume InChain(s), returns index of s in chain
    GetNext(s) ≜ chain[GetIndex(s) + 1]
     /*Assume InChain(s), returns successor of s in chain
    IsUp(s) ≜ up[s] = TRUE
     }

    fair process ( c ∈ Clients )   /*Client process
    variable ctr = −1, hver = −1 ;
    {
     C0: await (Len(chain) > 0) ;
     CL: while ( ctr ≤ STOP ) {
           status := "Reading" ;
           CLR: hver := db[chain[Len(chain)]].ver + 1 ;
                ctr := ctr + 1 ;
           CLW: while ( msg[self][1] ≠ "ACK" ∨
                        msg[self][2] ≠ hver ∨
                        msg[self][3] ≠ ctr ∨
                        msg[self][4] ≠ "UPDATE" ) {
```

```
                        msg[chain[1]] := ⟨"SYN", hver, ctr, "UPDATE", self⟩ ;
                      } ;
              status := "Writing" ;
              msg[self] := ⟨"X", −1, −1, "X", −1⟩ ;
        } ;
 }

fair process ( n ∈ Nodes )   /*Storage node
{
 ND: while ( TRUE ) {
       either
       NM: { if ( msg[self][1] = "SYN" ∧ Len(chain) > 0 ) {
                 /*react to message
                 if ( self = chain[Len(chain)] ∧ msg[self][4] = "QUERY" ) {
                   msg[msg[self][5]] := ⟨"ACK", db[self].ver, db[self].val, "QUERY", self⟩ ||
                   msg[self] := ⟨"X", −1, −1, "X", −1⟩ ;
                 }
                 else if ( self = chain[Len(chain)] ∧ msg[self][4] = "UPDATE" ) {
                   db[self].ver := msg[self][2] || db[self].val := msg[self][3] ;
                   msg[msg[self][5]] := ⟨"ACK", db[self].ver, db[self].val, "UPDATE", self⟩ ||
                   msg[self] := ⟨"X", −1, −1, "X", −1⟩ ;
                 }
                 else if ( self ≠ chain[Len(chain)] ∧
                           msg[self][4] = "UPDATE" ∧ InChain(self) = TRUE ) {
                   db[self].ver := msg[self][2] || db[self].val := msg[self][3] ;
                   msg[GetNext(self)] :=
                   ⟨"SYN", msg[self][2], msg[self][3], "UPDATE", msg[self][5]⟩ ||
                   msg[self] := ⟨"X", −1, −1, "X", −1⟩ ;
                 }
               }
        } or
       NDF: {
               if ( FailNum > 0 ∧ up[self] = TRUE ) {   /*Storage node can fail
                 up[self] := FALSE ;
                 FailNum := FailNum − 1 ; }
               else if ( up[self] = FALSE ) {   /*Or recover
                 up[self] := TRUE ;
                 msg[self] := ⟨"X",    −1, −1, "X", −1⟩ ;
                 FailNum := FailNum + 1 ; }
          }
  } ;
 }

fair process ( p = Configurator )   /*Maintain the chain
variables un = 0 ;
```

```
   {
    P: while ( TRUE ) {
         if ( Len(chain) = 0 ) {
           chain := Append(chain, FreeUpNode) ; }
         else if ( Len(chain) < 3 ∧ FreeUpNode > 0 ) {
           un := FreeUpNode ;
           db[un].ver := db[chain[Len(chain)]].ver
           || db[un].val := db[chain[Len(chain)]].val
           || db[un].cli := db[chain[Len(chain)]].cli ;
           chain := Append(SelectSeq(chain, IsUp), FreeUpNode) ; }
         else if ( Len(chain) > 0 ) {
           chain := SelectSeq(chain, IsUp) ; }
       } ;
    }

}
```

****

BEGIN TRANSLATION

VARIABLES $FailNum$, msg, up, db, $chain$, $status$, pc

define statement

$UpNodes \triangleq \{n \in Nodes : up[n] = \text{TRUE}\}$

$InChain(s) \triangleq \exists\, i \in 1 \,..\, Len(chain) : chain[i] = s$

$ChainNodes \triangleq \{chain[j] : j \in 1 \,..\, Len(chain)\}$

$FreeUpNode \triangleq$ IF $(UpNodes \setminus ChainNodes) \neq \{\}$ THEN
CHOOSE $i \in (UpNodes \setminus ChainNodes) : i > 0$
ELSE 0

$GetIndex(s) \triangleq$ CHOOSE $i \in 1 \,..\, Len(chain) : chain[i] = s$

$GetNext(s) \triangleq chain[GetIndex(s) + 1]$

$IsUp(s) \triangleq up[s] = \text{TRUE}$

VARIABLES ctr, $hver$, un

$vars \triangleq \langle FailNum,\ msg,\ up,\ db,\ chain,\ status,\ pc,\ ctr,\ hver,\ un\rangle$

$ProcSet \triangleq (Clients) \cup (Nodes) \cup \{Configurator\}$

$Init \triangleq$ Global variables
$\wedge\ FailNum = FAILNUM$
$\wedge\ msg = [j \in Procs \mapsto \langle \text{“X”},\ -1,\ -1,\ \text{“X”},\ -1\rangle]$
$\wedge\ up = [n \in Nodes \mapsto \text{TRUE}]$
$\wedge\ db = [n \in Nodes \mapsto [ver \mapsto -1,\ val \mapsto -1,\ cli \mapsto -1]]$
$\wedge\ chain = \langle 1\rangle$
$\wedge\ status = \text{“Writing”}$
Process c

$\land\ ctr = [self \in Clients \mapsto -1]$
$\land\ hver = [self \in Clients \mapsto -1]$
Process p
$\land\ un = 0$
$\land\ pc = [self \in ProcSet \mapsto$ CASE $self \in Clients \to$ "C0"
$\square\ self \in Nodes \to$ "ND"
$\square\ self = Configurator \to$ "P"$]$

$C0(self) \triangleq \land\ pc[self] =$ "C0"
$\land\ (Len(chain) > 0)$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "CL"$]$
$\land$ UNCHANGED $\langle FailNum, msg, up, db, chain, status, ctr, hver, un\rangle$

$CL(self) \triangleq \land\ pc[self] =$ "CL"
$\land$ IF $ctr[self] \leq STOP$
THEN $\land\ status' =$ "Reading"
$\land\ pc' = [pc$ EXCEPT $![self] =$ "CLR"$]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] =$ "Done"$]$
$\land$ UNCHANGED $status$
$\land$ UNCHANGED $\langle FailNum, msg, up, db, chain, ctr, hver, un\rangle$

$CLR(self) \triangleq \land\ pc[self] =$ "CLR"
$\land\ hver' = [hver$ EXCEPT $![self] = db[chain[Len(chain)]].ver + 1]$
$\land\ ctr' = [ctr$ EXCEPT $![self] = ctr[self] + 1]$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "CLW"$]$
$\land$ UNCHANGED $\langle FailNum, msg, up, db, chain, status, un\rangle$

$CLW(self) \triangleq \land\ pc[self] =$ "CLW"
$\land$ IF $msg[self][1] \neq$ "ACK" $\lor$
$msg[self][2] \neq hver[self] \lor$
$msg[self][3] \neq ctr[self] \lor$
$msg[self][4] \neq$ "UPDATE"
THEN $\land\ msg' = [msg$ EXCEPT $![chain[1]] = \langle$"SYN", $hver[self]$, $ctr[self]$, "UPDATE", se
$\land\ pc' = [pc$ EXCEPT $![self] =$ "CLW"$]$
$\land$ UNCHANGED $status$
ELSE $\land\ status' =$ "Writing"
$\land\ msg' = [msg$ EXCEPT $![self] = \langle$"X", -1, -1, "X", $-1\rangle]$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "CL"$]$
$\land$ UNCHANGED $\langle FailNum, up, db, chain, ctr, hver, un\rangle$

$c(self) \triangleq C0(self) \lor CL(self) \lor CLR(self) \lor CLW(self)$

$ND(self) \triangleq \land\ pc[self] =$ "ND"
$\land\ \lor\ \land\ pc' = [pc$ EXCEPT $![self] =$ "NM"$]$
$\lor\ \land\ pc' = [pc$ EXCEPT $![self] =$ "NDF"$]$
$\land$ UNCHANGED $\langle FailNum, msg, up, db, chain, status, ctr, hver, un\rangle$

$NM(self) \triangleq \land pc[self] = \text{“NM”}$
$\land \text{IF } msg[self][1] = \text{“SYN”} \land Len(chain) > 0$
$\text{THEN } \land \text{IF } self = chain[Len(chain)] \land msg[self][4] = \text{“QUERY”}$
$\text{THEN } \land msg' = [msg \text{ EXCEPT } ![msg[self][5]] = \langle \text{“ACK”}, db[self].ver, db[self]$
$![self] = \langle \text{“X”}, -1, -1, \text{“X”}, -1 \rangle]$
$\land db' = db$
$\text{ELSE } \land \text{IF } self = chain[Len(chain)] \land msg[self][4] = \text{“UPDATE”}$
$\text{THEN } \land db' = [db \text{ EXCEPT } ![self].ver = msg[self][2],$
$![self].val = msg[self][3]]$
$\land msg' = [msg \text{ EXCEPT } ![msg[self][5]] = \langle \text{“ACK”}, db'[self]$
$![self] = \langle \text{“X”}, -1, -1, \text{“X”}, -$
$\text{ELSE } \land \text{IF } self \neq chain[Len(chain)] \land$
$msg[self][4] = \text{“UPDATE”} \land InChain(self) = \text{TRUE}$
$\text{THEN } \land db' = [db \text{ EXCEPT } ![self].ver = msg[self][2]$
$![self].val = msg[self][3]$
$\land msg' = [msg \text{ EXCEPT } ![GetNext(self)] = \langle \text{“}$
$![self] = \langle \text{“X”}, -1,$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } \langle msg,$
$db \rangle$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } \langle msg, db \rangle$
$\land pc' = [pc \text{ EXCEPT } ![self] = \text{“ND”}]$
$\land \text{UNCHANGED } \langle FailNum, up, chain, status, ctr, hver, un \rangle$

$NDF(self) \triangleq \land pc[self] = \text{“NDF”}$
$\land \text{IF } FailNum > 0 \land up[self] = \text{TRUE}$
$\text{THEN } \land up' = [up \text{ EXCEPT } ![self] = \text{FALSE}]$
$\land FailNum' = FailNum - 1$
$\land msg' = msg$
$\text{ELSE } \land \text{IF } up[self] = \text{FALSE}$
$\text{THEN } \land up' = [up \text{ EXCEPT } ![self] = \text{TRUE}]$
$\land msg' = [msg \text{ EXCEPT } ![self] = \langle \text{“X”}, -1, -1, \text{“X”}, -1 \rangle]$
$\land FailNum' = FailNum + 1$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } \langle FailNum, msg, up \rangle$
$\land pc' = [pc \text{ EXCEPT } ![self] = \text{“ND”}]$
$\land \text{UNCHANGED } \langle db, chain, status, ctr, hver, un \rangle$

$n(self) \triangleq ND(self) \lor NM(self) \lor NDF(self)$

$P \triangleq \land pc[Configurator] = \text{“P”}$
$\land \text{IF } Len(chain) = 0$
$\text{THEN } \land chain' = Append(chain, FreeUpNode)$
$\land \text{UNCHANGED } \langle db, un \rangle$
$\text{ELSE } \land \text{IF } Len(chain) < 3 \land FreeUpNode > 0$

$$
\begin{array}{l}
\quad\quad\quad\quad \text{THEN} \land un' = \mathit{FreeUpNode} \\
\quad\quad\quad\quad\quad\quad \land db' = [db \text{ EXCEPT } ![un'].ver = db[chain[Len(chain)]].ver, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad ![un'].val = db[chain[Len(chain)]].val, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad ![un'].cli = db[chain[Len(chain)]].cli] \\
\quad\quad\quad\quad\quad\quad \land chain' = Append(SelectSeq(chain, IsUp), \mathit{FreeUpNode}) \\
\quad\quad\quad\quad \text{ELSE} \land \text{IF } Len(chain) > 0 \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{THEN} \land chain' = SelectSeq(chain, IsUp) \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{ELSE} \land \text{TRUE} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \land chain' = chain \\
\quad\quad\quad\quad\quad\quad \land \text{UNCHANGED } \langle db, un \rangle \\
\quad \land pc' = [pc \text{ EXCEPT } ![Configurator] = \text{“P”}] \\
\quad \land \text{UNCHANGED } \langle FailNum, msg, up, status, ctr, hver \rangle
\end{array}
$$

$$p \triangleq P$$

$$
\begin{array}{l}
Next \triangleq p \\
\quad\quad \lor (\exists\, self \in Clients : c(self)) \\
\quad\quad \lor (\exists\, self \in Nodes : n(self))
\end{array}
$$

$$
\begin{array}{l}
Spec \triangleq \land Init \land \Box[Next]_{vars} \\
\quad\quad\quad \land \forall\, self \in Clients : \mathrm{WF}_{vars}(c(self)) \\
\quad\quad\quad \land \forall\, self \in Nodes : \mathrm{WF}_{vars}(n(self)) \\
\quad\quad\quad \land \mathrm{WF}_{vars}(p)
\end{array}
$$

END TRANSLATION

$$
\begin{array}{l}
Invariant\_1 \triangleq \forall\, cl \in Clients : (status = \text{“Writing”} \Rightarrow \\
\quad\quad (hver[cl] = db[chain[Len(chain)]].ver \land ctr[cl] = db[chain[Len(chain)]].val)) \\
Invariant\_2 \triangleq \forall\, nl \in 1 \,..\, Len(chain) - 1 : ((Len(chain) > 1) \Rightarrow (db[chain[nl]].ver \geq db[chain[nl + 1]].ver))
\end{array}
$$

(* Observations: This project is server side routing, which acted as a black box to client- systems. This is different to Project 1 where global variables were used to do client side routing. Also there were no configurator in *Project*1 as all the responsibilities for data management was being done by the client. In this case (theoretically), a dedicated *PAXOS* based configurator is deployed to maintain the state of replicated chain. However in this project, configurator is not directly accessed by nodes, which posed its own challenges to define what constitutes an active tail or head node. We here, always consider current chain config while identifying head or tail.

1. As clients were not supposed to communicate with configurator for this project, deciding active head and tail was a big challange.
2. The assumption that atleast one node will be up all the time was a huge help in calculation of head and tail.
3. We faced problems while initializing the system and came to conclusion that system should always intialize with n nodes such that $n + 1 > \mathit{FAILNUM}$ (At least one node should be up for system to sustain consistent behavior).
4. Initializing with 1 or two nodes in this case was voilating consistency as both nodes can go down (because fo *FAILNUM*) and entire service will go down and system will loose data.

5. In case of 2 clients problem arises when say *client_1* read *ver* 3 and could not proceed to write. Meanwhile *client_2* kept on reading and writing till it finish up writing lets say 5. And if now *client_1* writes its data that will violate consistency for *client_2*.
6. Again after lot of test we found a case when configurator process is not getting picked; meanwhile a node went down and after a write cycle it comes back on (This causes the tail to fail and come back up with its *db* reset to default - while the configurator did not remove/re-configure nodes as of yet). Because of this one of the *invariant_1* is failing but we feel that this the limitation of this system design in the requirements.
7. A better design would be nodes communicating with configurator to get details about successors and copying etc.
8. *Voldchain* is more fail *safe*(*fault tolerant*) when compared with project 1, as it can still serve with single up node.
9. In *Voldchain*, tailnode is analogus to *readQ* (in project 1) and headnode is analogus to the *writeQ*. Server side routing is hence more fault tolerant with even lesser number of nodes required for maintaining single copy consistency.
10. In *Project*1 failnum must be less than the size of *readQ*, but *voldchain* can work with 1 upnode as discussed above.
11. Also 2 clients scenario will fail *invariant_1* as *db* of node can be consistant with only one client at a time and not with all of them.

*)

* *teamMember_1*: Debaditya *Basak*, 50206177

* *teamMember_2*: Divyank *Sharma*, 50207091