```
MODULE voldchain
EXTENDS $Integers$, $Sequences$, $FiniteSets$, $TLC$
CONSTANTS $N$, $C$, $STOP$, $FAILNUM$
ASSUME $N - FAILNUM \geq 1 \land STOP < 5 \land \land 0 \leq FAILNUM \land FAILNUM \leq 2$
$Nodes \triangleq 1 \,..\, N$
$Clients \triangleq N + 1 \,..\, N + C$
$Procs \triangleq 1 \,..\, N + C$
$Configurator \triangleq N + C + 1$

--algorithm $test${
    variable $FailNum = FAILNUM$, $step = 0$, $LSTWR = -1$, $CURRD = -1$, $lsttmp = -1$, $WRTFLG =$
                $msg = [j \in Procs \mapsto \langle\rangle]$,
                $up = [n \in Nodes \mapsto$ TRUE$]$,
                $db = [n \in Nodes \mapsto [ver \mapsto -1, val \mapsto -1, cli \mapsto -1]]$,
                $chain = \langle\rangle$ ;

    define
    {

        $UpNodes \triangleq \{i \in Nodes : up[i] =$ TRUE$\}$   Returns the set of up nodes
        $ChainNode \triangleq \{chain[i] : i \in 1 \,..\, Len(chain)\}$   Returns the elements in the chain as set
        $ChainNodes \triangleq$ IF $(Len(chain) = 0)$ THEN $\{-1\}$ ELSE $ChainNode$   Retruns $\{-1\}$ if chain is empty else t
        $InChain(s) \triangleq$ IF $(s \in ChainNodes)$ THEN TRUE ELSE FALSE   Check if the element is on chain
        $FreeUpNodes \triangleq \{i \in UpNodes : InChain(i) =$ FALSE $\land up[i] =$ TRUE$\}$   Retrun set of upnodes that are not
        $GetIndex(s) \triangleq$ CHOOSE $i \in 1 \,..\, Len(chain) : chain[i] = s$   Get the index position of $s$ in chain
        $GetNext(s) \triangleq$ IF $(GetIndex(s) = Len(chain))$ THEN $-1$ ELSE $chain[GetIndex(s) + 1]$   Get the successo
        $GetFreeNode \triangleq \{i \in FreeUpNodes : up[i] =$ TRUE$\}$
        $GetTail \triangleq chain[Len(chain)]$                                   Get the tail of chain
        $GetHead \triangleq chain[1]$
        $test(e) \triangleq$ IF $(up[e] =$ TRUE$)$ THEN TRUE ELSE FALSE
     }

    fair process ( $c \in Clients$ )
    variable $cntr = 0$, $hver = -1$, $lastelm = 0$, $tail = -1$, $head = -1$, $initial = 0$ ;
    {
        $C0$: await $(Len(chain) > 0)$ ;
        $CL$: while ( $cntr \leq STOP$ )
        {
          $CLR$:
          while ( $WRTFLG \neq self$ )   Iterate through read request
          {
            if ( $msg[self] \neq \langle\rangle$ )
            {
              if ( $msg[self].val = -1$ )
              {
                $hver := msg[self].ver + 1$ ;
              $CURRD := db[chain[Len(chain)]].ver \;||\; LSTWR := lsttmp$ ;
            if ( $WRTFLG = -1$ )
            {
              $WRTFLG := self$ ;
            } ;
          }
        } ;
        if ( $WRTFLG \neq self$ )
        {
          $tail := GetTail$ ;
          $msg[tail] := [ver \mapsto -1, val \mapsto -1, cli \mapsto self]$ ;
        }
      } ;
      CLW:
      while ( $WRTFLG = self$ )   Iterate through write request
      {
        if ( $msg[self].val \neq -1 \land msg[self].ver = hver$ )
        {
          $lsttmp := msg[self].ver$ ;
          $cntr := cntr + 1$ ;
          $WRTFLG := -1$ ;
        } ;
        if ( $WRTFLG = self$ )
        {
          $head := GetHead$ ;
          $msg[head] := [ver \mapsto hver, val \mapsto cntr, cli \mapsto self]$ ;
        }
      }
    }
}
fair process ( $n \in Nodes$ )
variable $nextnode = -1, clientid = 0$ ;
{
  ND: while ( TRUE )
  {
    either
    NM:
    {   react to message
      if ( $up[self] = \text{TRUE} \land msg[self] \neq \langle\rangle \land InChain(self) = \text{TRUE}$ )
      {
        $clientid := msg[self].cli$ ;
        if ( $msg[self].val = -1 \land self = GetTail$ )   handling Read message request
        {
          $msg[clientid] := [ver \mapsto db[self].ver, val \mapsto -1, cli \mapsto clientid] \;||\; msg[self] := \langle\rangle$ ;
        }
        else if ( $msg[self].val \neq -1$ )   handling Write message request
        {
          $db[self] := [ver \mapsto msg[self].ver,\ val \mapsto msg[self].val,\ cli \mapsto msg[self].cli]$ ;
          if ( $self = GetTail$ )        check if this is a tail node
          {
            $lsttmp := db[self].ver$ ;
            $msg[clientid] := [ver \mapsto db[self].ver,\ val \mapsto db[self].val,\ cli \mapsto clientid] \,\|\, msg[self] := \langle\rangle$ ;
           }
          else                           else get the successor
          {
            $nextnode := GetNext(self)$ ;
            $msg[nextnode] := msg[self] \,\|\, msg[self] := \langle\rangle$ ;
           }
         } ;
       }
     } or
    $NDF$:
    {
      if ( $FailNum > 0 \land up[self] = \text{TRUE}$ )   Storage node can fail
      {
        $up[self] := \text{FALSE}$ ;
        $FailNum := FailNum - 1$ ;
       }
      else if ( $up[self] = \text{FALSE}$ )    Or recover as a new node
      {
        $up[self]$   $:= \text{TRUE}$ ;
        $msg[self] := \langle\rangle$      ;
        $FailNum := FailNum + 1$ ;
       }
     }
   }
 }
fair process ( $p = Configurator$ )   Maintain the chain
variable $newnode = -1$ ;
{
  $P$: while ( TRUE )
  {
    $P1$:
    if ( $Len(chain) < 3$ )   Add a new node
    {
      if ( $FreeUpNodes \neq \{\}$ )
      {
        $newnode := \text{CHOOSE } k \in FreeUpNodes : \text{TRUE}$ ;
        if ( $Len(chain) = 0$ )
        {
              db[newnode] := [ver ↦ −1, val ↦ −1, cli ↦ 0] ;
              chain := Append(chain, newnode) ;
             }
            else
            {
              db[newnode] := db[chain[Len(chain)]] ;
              chain := Append(chain, newnode) ;
             } ;
           } ;
         }
        else   Delete fail node
        {
          chain := SelectSeq(chain, test) ;
         }
       }
     }
 }
```

```
 BEGIN TRANSLATION
VARIABLES FailNum, step, LSTWR, CURRD, lsttmp, WRTFLG, msg, up, db, chain, pc

 define statement
UpNodes ≜ {i ∈ Nodes : up[i] = TRUE}
ChainNode ≜ {chain[i] : i ∈ 1 . . Len(chain)}
ChainNodes ≜ IF (Len(chain) = 0) THEN {−1} ELSE  ChainNode
InChain(s)  ≜ IF (s ∈ ChainNodes) THEN TRUE ELSE  FALSE
FreeUpNodes ≜ {i ∈ UpNodes : InChain(i) = FALSE ∧ up[i] = TRUE}
GetIndex(s)  ≜ CHOOSE i ∈ 1 . . Len(chain) : chain[i] = s
GetNext(s) ≜ IF (GetIndex(s) = Len(chain)) THEN −1 ELSE  chain[GetIndex(s) + 1]
GetFreeNode ≜ {i ∈ FreeUpNodes : up[i] = TRUE}
GetTail   ≜ chain[Len(chain)]
GetHead ≜ chain[1]
test(e)    ≜ IF (up[e] = TRUE) THEN TRUE ELSE  FALSE

VARIABLES cntr, hver, lastelm, tail, head, initial, nextnode, clientid,
          newnode

vars ≜ ⟨FailNum, step, LSTWR, CURRD, lsttmp, WRTFLG, msg, up, db, chain,
        pc, cntr, hver, lastelm, tail, head, initial, nextnode, clientid,
        newnode⟩

ProcSet ≜ (Clients) ∪ (Nodes) ∪ {Configurator}

Init ≜  Global variables
        ∧ FailNum = FAILNUM
        ∧ step = 0
        ∧ LSTWR = −1
```

```
                ∧ CURRD = −1
                ∧ lsttmp   = −1
                ∧ WRTFLG = −1
                ∧ msg = [j ∈ Procs ↦ ⟨⟩]
                ∧ up = [n ∈ Nodes ↦ TRUE]
                ∧ db = [n ∈ Nodes ↦ [ver ↦ −1, val ↦ −1, cli ↦ −1]]
                ∧ chain = ⟨⟩
                 Process c
                ∧ cntr = [self ∈ Clients ↦ 0]
                ∧ hver = [self ∈ Clients ↦ −1]
                ∧ lastelm = [self ∈ Clients ↦ 0]
                ∧ tail  = [self ∈ Clients ↦ −1]
                ∧ head = [self ∈ Clients ↦ −1]
                ∧ initial = [self ∈ Clients ↦ 0]
                 Process n
                ∧ nextnode = [self ∈ Nodes ↦ −1]
                ∧ clientid  = [self ∈ Nodes ↦ 0]
                 Process p
                ∧ newnode = −1
                ∧ pc = [self ∈ ProcSet ↦ CASE self ∈ Clients → "C0"
                                            □   self ∈ Nodes → "ND"
                                            □   self = Configurator → "P"]

C0(self) ≜ ∧ pc[self] = "C0"
           ∧ (Len(chain) > 0)
           ∧ pc′ = [pc EXCEPT ![self] = "CL"]
           ∧ UNCHANGED ⟨FailNum, step, LSTWR, CURRD, lsttmp, WRTFLG, msg,
                          up, db, chain, cntr, hver, lastelm, tail, head,
                          initial, nextnode, clientid, newnode⟩

CL(self) ≜ ∧ pc[self] = "CL"
           ∧ IF cntr[self] ≤ STOP
                THEN ∧ pc′ = [pc EXCEPT ![self] = "CLR"]
                ELSE  ∧ pc′ = [pc EXCEPT ![self] = "Done"]
           ∧ UNCHANGED ⟨FailNum, step, LSTWR, CURRD, lsttmp, WRTFLG, msg,
                          up, db, chain, cntr, hver, lastelm, tail, head,
                          initial, nextnode, clientid, newnode⟩

CLR(self) ≜ ∧ pc[self]      = "CLR"
            ∧ IF WRTFLG ≠ self
                 THEN ∧ IF msg[self] ≠ ⟨⟩
                           THEN ∧ IF msg[self].val   = −1
                                     THEN ∧ hver′ = [hver EXCEPT ![self] = msg[self].ver + 1]
                                          ∧ ∧ CURRD′ = db[chain[Len(chain)]].ver
                                            ∧ LSTWR′ = lsttmp
                                          ∧ IF WRTFLG = −1
```

```
                                                           THEN ∧ WRTFLG′ = self
                                                           ELSE ∧ TRUE
                                                                ∧ UNCHANGED WRTFLG
                                              ELSE ∧ TRUE
                                                   ∧ UNCHANGED ⟨LSTWR, CURRD,
                                                                WRTFLG, hver⟩
                                  ELSE ∧ TRUE
                                       ∧ UNCHANGED ⟨LSTWR, CURRD, WRTFLG, hver⟩
                            ∧ IF WRTFLG′ ≠ self
                                  THEN ∧ tail′ = [tail EXCEPT ![self] = GetTail]
                                       ∧ msg′ = [msg EXCEPT ![tail′[self]] = [ver ↦ −1, val ↦ −1, cli ↦
                                  ELSE ∧ TRUE
                                       ∧ UNCHANGED ⟨msg, tail⟩
                            ∧ pc′ = [pc EXCEPT ![self] = "CLR"]
                     ELSE   ∧ pc′ = [pc EXCEPT ![self] = "CLW"]
                            ∧ UNCHANGED ⟨LSTWR, CURRD, WRTFLG, msg, hver, tail⟩
                ∧ UNCHANGED ⟨FailNum, step, lsttmp, up, db, chain, cntr,
                             lastelm, head, initial, nextnode, clientid,
                             newnode⟩

CLW(self) ≜  ∧ pc[self]      = "CLW"
                ∧ IF WRTFLG = self
                     THEN   ∧ IF msg[self].val ≠ −1 ∧ msg[self].ver = hver[self]
                                  THEN ∧ lsttmp′ = msg[self].ver
                                       ∧ cntr′ = [cntr EXCEPT ![self] = cntr[self] + 1]
                                       ∧ WRTFLG′ = −1
                                  ELSE ∧ TRUE
                                       ∧ UNCHANGED ⟨lsttmp, WRTFLG, cntr⟩
                            ∧ IF WRTFLG′ = self
                                  THEN ∧ head′ = [head EXCEPT ![self] = GetHead]
                                       ∧ msg′ = [msg EXCEPT ![head′[self]] = [ver ↦ hver[self], val ↦ cntr
                                  ELSE ∧ TRUE
                                       ∧ UNCHANGED ⟨msg, head⟩
                            ∧ pc′ = [pc EXCEPT ![self] = "CLW"]
                     ELSE   ∧ pc′ = [pc EXCEPT ![self] = "CL"]
                            ∧ UNCHANGED ⟨lsttmp, WRTFLG, msg, cntr, head⟩
                ∧ UNCHANGED ⟨FailNum, step, LSTWR, CURRD, up, db, chain, hver,
                             lastelm, tail, initial, nextnode, clientid,
                             newnode⟩

c(self) ≜ C0(self) ∨ CL(self) ∨ CLR(self) ∨ CLW(self)

ND(self) ≜ ∧ pc[self] = "ND"
              ∧ ∨ ∧ pc′ = [pc EXCEPT ![self] = "NM"]
                ∨ ∧ pc′ = [pc EXCEPT ![self] = "NDF"]
              ∧ UNCHANGED ⟨FailNum, step, LSTWR, CURRD, lsttmp, WRTFLG, msg,
                  $up, db, chain, cntr, hver, lastelm, tail, head,$
                  $initial, nextnode, clientid, newnode\rangle$

$NM(self) \triangleq \land pc[self] = \text{"NM"}$
            $\land \text{IF } up[self] = \text{TRUE} \land msg[self] \neq \langle\rangle \land InChain(self) = \text{TRUE}$
                $\text{THEN } \land clientid' = [clientid \text{ EXCEPT } ![self] = msg[self].cli]$
                     $\land \text{IF } msg[self].val = -1 \land self = GetTail$
                         $\text{THEN } \land msg' = [msg \text{ EXCEPT } ![clientid'[self]] = [ver \mapsto db[self].ver, val \mapsto$
                                                          $![self] = \langle\rangle]$
                              $\land \text{UNCHANGED } \langle lsttmp, db, nextnode\rangle$
                         $\text{ELSE } \land \text{IF } msg[self].val \neq -1$
                                  $\text{THEN } \land db' = [db \text{ EXCEPT } ![self] = [ver \mapsto msg[self].ver, val \mapsto$
                                       $\land \text{IF } self = GetTail$
                                           $\text{THEN } \land lsttmp' = db'[self].ver$
                                                $\land msg' = [msg \text{ EXCEPT } ![clientid'[self]] = [ve$
                                                                         $![self] = \langle\rangle]$
                                                $\land \text{UNCHANGED } nextnode$
                                           $\text{ELSE } \land nextnode' = [nextnode \text{ EXCEPT } ![self] = Ge$
                                                $\land msg' = [msg \text{ EXCEPT } ![nextnode'[self]] = m$
                                                                         $![self] = \langle\rangle]$
                                                $\land \text{UNCHANGED } lsttmp$
                                  $\text{ELSE } \land \text{TRUE}$
                                       $\land \text{UNCHANGED } \langle lsttmp, msg, db,$
                                                              $nextnode\rangle$
                $\text{ELSE } \land \text{TRUE}$
                     $\land \text{UNCHANGED } \langle lsttmp, msg, db, nextnode, clientid\rangle$
            $\land pc' = [pc \text{ EXCEPT } ![self] = \text{"ND"}]$
            $\land \text{UNCHANGED } \langle FailNum, step, LSTWR, CURRD, WRTFLG, up, chain,$
                              $cntr, hver, lastelm, tail, head, initial, newnode\rangle$

$NDF(self) \triangleq \land pc[self] = \text{"NDF"}$
             $\land \text{IF } FailNum > 0 \land up[self] = \text{TRUE}$
                 $\text{THEN } \land up' = [up \text{ EXCEPT } ![self] = \text{FALSE}]$
                      $\land FailNum' = FailNum - 1$
                      $\land msg' = msg$
                 $\text{ELSE } \land \text{IF } up[self] = \text{FALSE}$
                          $\text{THEN } \land up' = [up \text{ EXCEPT } ![self] = \text{TRUE}]$
                               $\land msg' = [msg \text{ EXCEPT } ![self] = \langle\rangle]$
                               $\land FailNum' = FailNum + 1$
                          $\text{ELSE } \land \text{TRUE}$
                               $\land \text{UNCHANGED } \langle FailNum, msg, up\rangle$
             $\land pc' = [pc \text{ EXCEPT } ![self] = \text{"ND"}]$
             $\land \text{UNCHANGED } \langle step, LSTWR, CURRD, lsttmp, WRTFLG, db, chain,$
                               $cntr, hver, lastelm, tail, head, initial,$
                               $nextnode, clientid, newnode\rangle$
```

```
n(self) ≜ ND(self) ∨ NM(self) ∨ NDF(self)

P ≜ ∧ pc[Configurator] = "P"
    ∧ pc' = [pc EXCEPT ![Configurator] = "P1"]
    ∧ UNCHANGED ⟨FailNum, step, LSTWR, CURRD, lsttmp, WRTFLG, msg, up, db,
                 chain, cntr, hver, lastelm, tail, head, initial, nextnode,
                 clientid, newnode⟩

P1 ≜ ∧ pc[Configurator] = "P1"
     ∧ IF Len(chain) < 3
          THEN ∧ IF FreeUpNodes ≠ {}
                    THEN ∧ newnode' = (CHOOSE k ∈ FreeUpNodes : TRUE)
                         ∧ IF Len(chain) = 0
                              THEN ∧ db' = [db EXCEPT ![newnode'] = [ver ↦ −1, val ↦ −1, cli ↦
                                   ∧ chain' = Append(chain, newnode')
                              ELSE ∧ db' = [db EXCEPT ![newnode'] = db[chain[Len(chain)]]]
                                   ∧ chain' = Append(chain, newnode')
                    ELSE ∧ TRUE
                         ∧ UNCHANGED ⟨db, chain, newnode⟩
          ELSE ∧ chain' = SelectSeq(chain, test)
               ∧ UNCHANGED ⟨db, newnode⟩
     ∧ pc' = [pc EXCEPT ![Configurator] = "P"]
     ∧ UNCHANGED ⟨FailNum, step, LSTWR, CURRD, lsttmp, WRTFLG, msg, up,
                  cntr, hver, lastelm, tail, head, initial, nextnode,
                  clientid⟩

p ≜ P ∨ P1

Next ≜ p
          ∨ (∃ self ∈ Clients : c(self))
          ∨ (∃ self ∈ Nodes : n(self))

Spec ≜ ∧ Init ∧ □[Next]_vars
       ∧ ∀ self ∈ Clients : WF_vars(c(self))
       ∧ ∀ self ∈ Nodes : WF_vars(n(self))
       ∧ WF_vars(p)

 END TRANSLATION
Invariant1 ≜ LSTWR = CURRD
Invariant2 ≜ ∀ i     ∈ 1 .. (Len(chain) − 1) : db[chain[i]].ver ≥ db[chain[i + 1]].ver
```

* Modification History

* Last modified *Thu Dec* 08 23:49:00 *EST* 2016 by *Rohit*

* Last modified *Thu Dec* 08 20:27:28 *EST* 2016 by *anand*

* Created *Thu Nov* 24 19:14:19 *EST* 2016 by *anand*

Members − *Anand Sankar Bhagavandas* − *UB id* − 50208048

$Rohit\ Joseph\ Sebastian - UB\ id - 50204806$

The code given above implements Replicated Storage with Chain Replication.

The invariant we have specified are -

1) The version number of the last write operation is equal to the version number of the current read operation.
2) The second invariant is that the *db.ver* is non-increasing when we traverse from head to tail.

Given below are our observations about this version of the protocol -

Voldchain is capable of tolerating more faults with less number of nodes. This is because in the *project*1 version of the protocol, the condition that the value for *ReadQ* and *WriteQ* should be atleast one greater than the value of *FAILNUM* ineffect also brings up the necessity that the majority of nodes should be up and running.

eg for the case $- ReadQ = 3$, $WriteQ = 3$, $FAILNUM = 2$ and Number of $Nodes\ (N) = 3$, This will bring up an error as there are not enough nodes to satisfy the size requirements for *ReadQ* and *WriteQ* if two nodes are down.

In the *VoldChain* protocol, the chain length only needs to be one greater than *FAILNUM* and the number of nodes can be equal to the chain length. This will ensure that atleast one node with the current value is up to serve the request of the client.

The Read quorum is analogus to the tail of the chain. The client sends read requests to the tail and the tail responds with the value of the version number that is available with it. Since the write messages are constantly put in queue, the tail will eventually possess the highest version number.

The Write quorum is analogus to the whole chain. The client contacts the head which passes on the message to the other nodes. Eventually all the nodes in the chain will possess the updated value.

The previous relation between the quorum and *FAILNUM* holds in this case too. The number of nodes in the chain should be one more than *FAILNUM*.

We have implemented a token based lock system to ensure that the write operation is consistent and that another client does not write while a write operation is in progress. While in possesion of this token, the client keeps sending write message till it gets the confirmation that the write operation is completed and the token is released from the client.

The system runs without the use of the aforementioned token system when the number of clients $C = 1$. But it doesnot satisfy the invariant without the token system when $C = 2$.