MODULE *improved*

EXTENDS *Naturals*, *Sequences*

CONSTANTS
USERIDS,
SERVERS,
METADATAS,
IMAGES

VARIABLES
databaseState,
blobStoreState,
serverStates,

operations

$vars \triangleq \langle databaseState, blobStoreState, serverStates, operations \rangle$

Strong Typing

$UserIdVal \triangleq USERIDS \cup \{\text{“UNSET”}\}$
$MetadataVal \triangleq METADATAS \cup \{\text{“UNSET”}\}$
$ImageVal \triangleq IMAGES \cup \{\text{“UNSET”}\}$

Describes all possible states a server can be in.

$ServerStateVal \triangleq$
[
state : {
current:
“waiting”, next: *StartWrite* or *StartRead*
after: *StartWrite*
“started_write”, next: *WriteBlob* or *FailWrite*
after: *WriteBlob*
“wrote_blob”, next: *WriteMetadataAndReturn* or *FailWrite*
after: *StartRead*
“started_read”, next: *ReadMetadata*
after: *ReadMetadata*, *ReadMetadataAndReturnEmpty*
“read_metadata” next: *ReadBlobAndReturn*
},
userId : *UserIdVal*,
metadata : *MetadataVal*,
image : *ImageVal*
]

$OperationValue \triangleq [type : \{\text{“READ”}, \text{“WRITE”}\},$

$userId : UserIdVal,$
$metadata : MetadataVal,$
$image : ImageVal]$

$TypeOk \triangleq$
$\land \quad databaseState \in [USERIDS \to MetadataVal]$
$\land \quad blobStoreState \in [USERIDS \to ImageVal]$
$\land \quad serverStates \in [SERVERS \to ServerStateVal]$
$\land \quad operations \in Seq(OperationValue)$

$Init \triangleq$
$\land\ databaseState = [u \in USERIDS \mapsto \text{"UNSET"}]$
$\land\ blobStoreState = [u \in USERIDS \mapsto \text{"UNSET"}]$
$\land\ serverStates = [s \in SERVERS \mapsto [state \mapsto \text{"waiting"},$
$userId \mapsto \text{"UNSET"},$
$metadata \mapsto \text{"UNSET"},$
$image \mapsto \text{"UNSET"}$
$]]$
$\land\ operations = \langle\rangle$

State Machine: All of the states are functions of s (server), because the only actively modeled actors in this system are our servers, but there can be multiple working simultaneously.

Writes

$StartWrite(s) \triangleq$
$\land\ serverStates[s].state = \text{"waiting"}$
$\land \exists\, u \in USERIDS,\ m \in METADATAS,\ i \in IMAGES :$
$\land\ serverStates' = [serverStates \text{ EXCEPT}$
$![s].state = \text{"started\_write"},$
Set values for the upcoming write
$![s].userId = u,$
$![s].metadata = m,$
$![s].image = i]$
Record the write for observability
$\land\ operations' = Append(operations,$
$[$
$type \mapsto \text{"WRITE"},$
$userId \mapsto u,$
$metadata \mapsto m,$
$image \mapsto i$
$])$
$\land \text{ UNCHANGED } \langle databaseState,\ blobStoreState\rangle$

$WriteBlob(s) \triangleq$
LET $currentState \triangleq serverStates[s]$
IN
$\land\ currentState.state = \text{“started\_write”}$
$\land\ blobStoreState' = [blobStoreState \text{ EXCEPT}$
$\quad ![currentState.userId] = currentState.image]$

$\land\ serverStates' = [serverStates \text{ EXCEPT}$
$\quad ![s].state = \text{“wrote\_blob”}]$
$\land$ UNCHANGED $\langle databaseState, operations\rangle$

Writing the database is now the last part of a write operation

$WriteMetadataAndReturn(s) \triangleq$
LET $currentState \triangleq serverStates[s]$
IN
$\land\ currentState.state = \text{“wrote\_blob”}$
$\land\ databaseState' = [databaseState \text{ EXCEPT}$
$\quad ![currentState.userId] = currentState.metadata]$
$\land\ serverStates' = [serverStates \text{ EXCEPT}$
$\quad ![s].state = \text{“waiting”}]$
$\land$ UNCHANGED $\langle blobStoreState, operations\rangle$

$FailWrite(s) \triangleq$
$\land\ serverStates[s].state \in \{\text{“started\_write”}, \text{“wrote\_blob”}\}$
$\land\ serverStates' = [serverStates \text{ EXCEPT}$
$\quad ![s].state = \text{“waiting”},$
$\quad ![s].userId = \text{“UNSET”},$
$\quad ![s].metadata = \text{“UNSET”},$
$\quad ![s].image = \text{“UNSET”}]$
$\land$ UNCHANGED $\langle databaseState, blobStoreState, operations\rangle$

Reads

$StartRead(s) \triangleq$
Reading only starts when a server is waiting
$\land\ serverStates[s].state = \text{“waiting”}$
$\land \exists\, u \in USERIDS :$
$\quad serverStates' = [serverStates \text{ EXCEPT}$
$\quad\quad ![s].state = \text{“started\_read”},$
$\quad\quad ![s].userId = u]$

$\land$ UNCHANGED $\langle databaseState, blobStoreState\rangle$
$\land$ UNCHANGED $operations$

If database record is present

```
ReadMetadata(s) ≜
    LET currentState ≜ serverStates[s]
    IN
     ∧ currentState.state = "started_read"
     Represents reading the metadata while the database record is set
     ∧ databaseState[currentState.userId] ≠ "UNSET"
     ∧ serverStates' =
            [serverStates EXCEPT
                    ![s].state = "read_metadata",
                    ![s].metadata = databaseState[currentState.userId]]


     ∧ UNCHANGED ⟨databaseState, blobStoreState⟩
     ∧ UNCHANGED operations

 If database record is not present
ReadMetadataAndReturnEmpty(s) ≜
    LET currentState ≜ serverStates[s]
    IN
     ∧ currentState.state = "started_read"
     Represents reading the metadata while the database record is unset
     ∧ databaseState[currentState.userId] = "UNSET"
     ∧ serverStates' = [serverStates EXCEPT
                            ![s].state = "waiting"]

     ∧ operations' = Append(operations,
                            Returns an empty record
                            [
                                type ↦ "READ",
                                userId ↦ currentState.userId,
                                metadata ↦ "UNSET",
                                image ↦ "UNSET"
                            ])
     ∧ UNCHANGED ⟨databaseState, blobStoreState⟩

ReadBlobAndReturn(s) ≜
    LET currentState ≜ serverStates[s]
    IN
     ∧ currentState.state = "read_metadata"
     ∧ serverStates' = [serverStates EXCEPT
                            ![s].state = "waiting",
                            ![s].image = blobStoreState[currentState.userId]]

     ∧ operations' = Append(operations,
                            [
                                type ↦ "READ",
```

$userId \mapsto currentState.userId,$
$metadata \mapsto currentState.metadata,$
$image \mapsto blobStoreState[currentState.userId]$
$])$
$\land$ UNCHANGED $\langle databaseState, blobStoreState \rangle$

Specification / *Next*

$Next \triangleq$

For every step, pick a server and have it advance one state

$\exists\, s \in SERVERS :$
$\lor\ StartWrite(s)$
$\lor\ WriteBlob(s)$ New step
$\lor\ WriteMetadataAndReturn(s)$ New step
$\lor\ FailWrite(s)$
$\lor\ StartRead(s)$
$\lor\ ReadMetadata(s)$ New step
$\lor\ ReadMetadataAndReturnEmpty(s)$ New step
$\lor\ ReadBlobAndReturn(s)$

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

Invariants

$ConsistentReads \triangleq$

If there are no operations, they are consistent

$\lor\ operations = \langle\rangle$
$\lor\ \forall\, i \in 1 \,..\, Len(operations) :$ For every read operation
LET $readOp \triangleq operations[i]$ IN
$\lor\ \land\ readOp.type =$ "READ"
There must exist a write operation
$\land\ \lor\ \exists\, j \in 1 \,..\, i :$
LET $writeOp \triangleq operations[j]$ IN
$\land\ writeOp.type =$ "WRITE"
With the same data
$\land\ readOp.userId = writeOp.userId$
$\land\ readOp.metadata = writeOp.metadata$
$\land\ readOp.image = writeOp.image$
$\lor$ Ignore unset reads
$\land\ readOp.metadata =$ "UNSET"
$\land\ readOp.image =$ "UNSET"
$\lor\ readOp.type =$ "WRITE" Ignore writes

This is used for model checker configuration so the simulation doesn't go on forever.

$$StopAfter3Operations \triangleq$$
$$Len(operations) \leq 3$$