MODULE *storagecleaner*

EXTENDS *Naturals*, *Sequences*, *FiniteSets*

CONSTANTS
USERIDS,
SERVERS,
METADATAS,
IMAGES,
UUIDS,
CLEANERS

VARIABLES
Implementation variables
databaseState,
blobStoreState,
serverStates,
cleanerStates,

We just added a time variable here
time, Natural number representing the number of hours that have passed

Observability variables
operations,
usedIds,

Temporal property variable: used to change state space on failure
failed

$helperVars \triangleq \langle time, operations, usedIds, failed \rangle$

$vars \triangleq \langle databaseState, blobStoreState,$
$serverStates, operations, cleanerStates, time, usedIds, failed \rangle$

$cleanerVars \triangleq \langle cleanerStates \rangle$

Strong Typing

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

$DatabaseRecord \triangleq [$
$metadata : MetadataVal,$
$imageId : UUIDVal$
$]$

```
BlobStoreRecord ≜ [
    image : ImageVal,
    created : Nat
] ∪ {[
   status ↦ "UNSET",
   image ↦ "UNSET"
]}

ServerStateVal ≜
    [
        state : {
            "waiting",
            "started_write",
            "wrote_blob",
            "started_read",
            "read_metadata"
        },
        userId : UserIdVal,
        metadata :  MetadataVal,
        imageId : UUIDVal,
        image : ImageVal,
        start  : Nat  added to track when a request starts
    ]

CleanerStateVal ≜
    [
        state : {
            "waiting",
            "got_blob_keys",
            "got_unused_keys",
            "deleting_keys"
        },
         This will be used to introduce a delay
        unusedKeyTime : Nat,
        blobKeys : SUBSET UUIDS,
        unusedBlobKeys : SUBSET UUIDS
    ]

OperationValue ≜ [type : {"READ", "WRITE"},
                  userId : UserIdVal,
                  metadata : MetadataVal,
                  image : ImageVal]

TypeOk ≜
    ∧  databaseState ∈ [USERIDS → DatabaseRecord]
```

$\land$ $blobStoreState \in [UUIDS \to BlobStoreRecord]$
$\land$ $serverStates \in [SERVERS \to ServerStateVal]$
$\land$ $cleanerStates \in [CLEANERS \to CleanerStateVal]$
$\land$ $operations \in Seq(OperationValue)$
$\land$ $time \in Nat$
$\land$ $usedIds \in \text{SUBSET } UUIDS$
$\land$ $failed \in Nat$

$Init \triangleq$
$\land$ $databaseState =$
$[u \in USERIDS \mapsto [metadata \mapsto$ "UNSET", $imageId \mapsto$ "UNSET"$]]$
$\land$ $blobStoreState =$
$[u \in UUIDS \mapsto [status \mapsto$ "UNSET", $image \mapsto$ "UNSET"$]]$
$\land$ $serverStates = [s \in SERVERS \mapsto [state \mapsto$ "waiting",
$userId \mapsto$ "UNSET",
$metadata \mapsto$ "UNSET",
$imageId \mapsto$ "UNSET",
$image \mapsto$ "UNSET",
will be set on start states
$start \mapsto 0$
$]]$
$\land$ $cleanerStates = [c \in CLEANERS \mapsto [$
$state \mapsto$ "waiting",
$blobKeys \mapsto \{\}$,
$unusedBlobKeys \mapsto \{\}$,
$unusedKeyTime \mapsto 0$
$]]$
$\land$ $operations = \langle\rangle$
$\land$ $time = 0$ Time starts at 0
$\land$ $usedIds = \{\}$
$\land$ $failed = 0$

State Machine:

$TimePasses \triangleq$
$\land$ $time' = time + 1$
$\land$ UNCHANGED $\langle serverStates, databaseState,$
$blobStoreState, cleanerStates\rangle$
$\land$ UNCHANGED $\langle operations, usedIds, failed\rangle$

Server Restart

$ServerRestart(s) \triangleq$
LET $currentState \triangleq serverStates[s]$ IN
LET $terminationTime \triangleq (currentState.start + 1)$ IN

```
    ∧ currentState.state ≠ "waiting"  Server must be active
     This is the only state a server can reach if past termination time
    ∧ time ⇒ terminationTime
    ∧ serverStates′ = [serverStates EXCEPT
                                ![s].state = "waiting",
                                ![s].userId = "UNSET",
                                ![s].metadata = "UNSET",
                                ![s].image = "UNSET",
                                ![s].imageId = "UNSET"]
    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerVars⟩
    ∧ UNCHANGED helperVars
```

Server Writes

```
ServerStartWrite(s) ≜
    ∧ serverStates[s].state = "waiting"
    ∧ ∃ u ∈ USERIDS, m ∈ METADATAS, i ∈ IMAGES :
        ∧ serverStates′ = [serverStates EXCEPT
                                ![s].state = "started_write",
                                ![s].userId = u,
                                ![s].metadata = m,
                                ![s].image = i,
                                 The time a write request starts
                                ![s].start = time
                               ]
        ∧ operations′ = Append(operations,
                                    [
                                        type ↦ "WRITE",
                                        userId ↦ u,
                                        metadata ↦ m,
                                        image ↦ i
                                    ])
    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerStates⟩
    ∧ UNCHANGED ⟨time, usedIds, failed⟩

ServerWriteBlob(s) ≜
    LET currentState ≜ serverStates[s]IN
    LET terminationTime ≜ (currentState.start + 1)IN
    ∧ time < terminationTime  Can only start this state if server is live
    ∧ currentState.state = "started_write"
    ∧ ∃ id ∈ UUIDS :
        ∧ id ∉ usedIds
        ∧ blobStoreState[id] = [status ↦ "UNSET", image ↦ "UNSET"]
        ∧ blobStoreState′ = [blobStoreState EXCEPT
                                ![id] = [
```

```
                                    image ↦ currentState.image,
                                    created ↦ time
                                    ]]
        ∧ serverStates′ = [serverStates EXCEPT
                                    ![s].state = "wrote_blob",
                                    ![s].imageId = id]
        ∧ usedIds′ = usedIds ∪ {id}
    ∧ UNCHANGED ⟨databaseState, cleanerVars⟩
    ∧ UNCHANGED ⟨time, operations, failed⟩

ServerWriteMetadataAndReturn(s) ≜
    LET currentState ≜ serverStates[s] IN
    LET terminationTime ≜ (currentState.start + 1) IN
    ∧ time < terminationTime
    ∧ currentState.state = "wrote_blob"
    ∧ databaseState′ = [databaseState EXCEPT
                            ![currentState.userId] = [
                                metadata ↦ currentState.metadata,
                                imageId ↦ currentState.imageId]]

    ∧ serverStates′ = [serverStates EXCEPT
                                    ![s].state = "waiting",
                                    ![s].userId = "UNSET",
                                    ![s].metadata = "UNSET",
                                    ![s].image = "UNSET",
                                    ![s].imageId = "UNSET"]
    ∧ UNCHANGED ⟨blobStoreState, cleanerVars⟩
    ∧ UNCHANGED helperVars

ServerFailWrite(s) ≜
    LET currentState ≜ serverStates[s] IN
    LET terminationTime ≜ (currentState.start + 1) IN
    ∧ time < terminationTime  Can only start this state if server is live
    ∧ serverStates[s].state ∈ {"started_write", "wrote_blob"}
    ∧ serverStates′ = [serverStates EXCEPT
                                    ![s].state = "waiting",
                                    ![s].userId = "UNSET",
                                    ![s].metadata = "UNSET",
                                    ![s].image = "UNSET",
                                    ![s].imageId = "UNSET"]
    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerVars⟩
    ∧ UNCHANGED helperVars
```

Server Reads

```
ServerStartRead(s) ≜
    LET currentState ≜ serverStates[s] IN
    ∧ serverStates[s].state = "waiting"
    ∧ ∃ u ∈ USERIDS :
            serverStates' = [serverStates EXCEPT
                                ![s].state = "started_read",
                                ![s].userId = u,
                                The time a read request starts
                                ![s].start = time
                               ]

    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerVars⟩
    ∧ UNCHANGED helperVars


ServerReadMetadata(s) ≜
    LET currentState ≜ serverStates[s] IN
    LET terminationTime ≜ (currentState.start + 1) IN
    ∧ time < terminationTime  Can only start this state if server is live
    ∧ currentState.state = "started_read"
    ∧ databaseState[currentState.userId].metadata ≠ "UNSET"
    ∧ serverStates' =
       [serverStates EXCEPT
          ![s].state = "read_metadata",
          ![s].metadata = databaseState[currentState.userId].metadata,
          ![s].imageId = databaseState[currentState.userId].imageId]
    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerVars⟩
    ∧ UNCHANGED helperVars


ServerReadMetadataAndReturnEmpty(s) ≜
    LET currentState ≜ serverStates[s] IN
    LET terminationTime ≜ (currentState.start + 1) IN
    ∧ time < terminationTime  Can only start this state if server is live
    ∧ currentState.state = "started_read"
    ∧ databaseState[currentState.userId].metadata = "UNSET"
    ∧ serverStates' = [serverStates EXCEPT
                          ![s].state = "waiting",
                          ![s].userId = "UNSET",
                          ![s].metadata = "UNSET",
                          ![s].image = "UNSET",
                          ![s].imageId = "UNSET"]

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

```
                                    metadata ↦ "UNSET",
                                    image ↦ "UNSET"
                                ])
    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerVars⟩
    ∧ UNCHANGED ⟨usedIds, time, failed⟩

ServerReadBlobAndReturn(s) ≜
    LET currentState ≜ serverStates[s] IN
    LET terminationTime ≜ (currentState.start + 1) IN
    ∧ time < terminationTime  Can only start this state if server is live
    ∧ currentState.state = "read_metadata"
    ∧ operations' = Append(operations,
                                [
                                    type ↦ "READ",
                                    userId ↦ currentState.userId,
                                    metadata ↦ currentState.metadata,
                                    image ↦ blobStoreState[currentState.imageId].image
                                ])
    ∧ serverStates' = [serverStates EXCEPT
                            ![s].state = "waiting",
                            ![s].userId = "UNSET",
                            ![s].metadata = "UNSET",
                            ![s].image = "UNSET",
                            ![s].imageId = "UNSET"]
    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerVars⟩
    ∧ UNCHANGED ⟨usedIds, time, failed⟩

Cleaner States

This is the main change in the logic.
CleanerStartGetBlobKeys(c) ≜
    LET current ≜ cleanerStates[c] IN
    ∧ current.state = "waiting"
    ∧ cleanerStates' = [
        cleanerStates EXCEPT
            ![c].state = "got_blob_keys",
             All keys in blockstore
            ![c].blobKeys = {
                k ∈ UUIDS :
                    LET earliestDeletionTime ≜ blobStoreState[k].created + 2 IN
                     That are not unset
                    ∧ blobStoreState[k] ≠ [
                        status ↦ "UNSET",
                        image ↦ "UNSET"]
                     It must be created 2 or more hours ago
```

```
                    ∧ earliestDeletionTime ≤ time
            }
        ]
    ∧ UNCHANGED ⟨serverStates, databaseState,
                 blobStoreState⟩
    ∧ UNCHANGED helperVars

CleanerGetUnusedKeys(c) ≜
    LET current ≜ cleanerStates[c] IN
    ∧ current.state = "got_blob_keys"
    ∧ cleanerStates' = [
        cleanerStates EXCEPT
            ![c].state = "got_unused_keys",
            ![c].unusedBlobKeys =
                {k ∈ current.blobKeys :
                    ∀ u ∈ USERIDS :
                        databaseState[u].imageId ≠ k},
            Mark the time the unused keys were retrieved
            ![c].unusedKeyTime = time
        ]
    ∧ UNCHANGED ⟨serverStates, databaseState,
                 blobStoreState⟩
    ∧ UNCHANGED helperVars

CleanerDeletingKeys(c) ≜
    LET current ≜ cleanerStates[c] IN
    Keys get deleted a minimum 1 hour after they are valid. Giving reads
    a the time to die.
    LET earliestDeleteTime ≜ current.unusedKeyTime + 1 IN
    ∧ time ≥ earliestDeleteTime
    ∧ current.state ∈ {"got_unused_keys", "deleting_keys"}
    ∧ Cardinality(current.unusedBlobKeys) ≠ 0
    ∧ ∃ k ∈ current.unusedBlobKeys :  pick a key to delete
        ∧ blobStoreState' =
            [blobStoreState EXCEPT
                ![k] = [status ↦ "UNSET", image ↦ "UNSET"]]
        ∧ cleanerStates' = [
            cleanerStates EXCEPT
                ![c].unusedBlobKeys = current.unusedBlobKeys \ {k}
            ]
    ∧ UNCHANGED ⟨serverStates, databaseState⟩
    ∧ UNCHANGED helperVars

CleanerFinished(c) ≜
    LET current ≜ cleanerStates[c] IN
    ∧ current.state = "deleting_keys"
```

```
    ∧ Cardinality(current.unusedBlobKeys) = 0
    ∧ cleanerStates' = [
        cleanerStates EXCEPT
            ![c].state = "waiting",
            ![c].blobKeys = {},
            ![c].unusedBlobKeys = {}
        ]
    ∧ UNCHANGED ⟨serverStates, databaseState,
                 blobStoreState⟩
    ∧ UNCHANGED helperVars

CleanerFail(c) ≜
    LET current ≜ cleanerStates[c] IN
    ∧ current.state ∈ {"got_blob_keys", "got_unused_keys", "deleting_keys"}
    ∧ cleanerStates' = [
        cleanerStates EXCEPT
            ![c].state = "waiting",
            ![c].blobKeys = {},
            ![c].unusedBlobKeys = {}
        ]
    change state space on failed
    ∧ failed' = failed + 1
    ∧ UNCHANGED ⟨serverStates, databaseState,
                 blobStoreState⟩
    ∧ UNCHANGED ⟨time, operations, usedIds⟩
```

Specification / *Next*

```
CleanerSteps ≜
    ∃ c ∈ CLEANERS :
        ∨ CleanerStartGetBlobKeys(c)
        ∨ CleanerGetUnusedKeys(c)
        ∨ CleanerDeletingKeys(c)
        ∨ CleanerFinished(c)
        ∨ CleanerFail(c)

Next ≜
    Time can pass now
    ∨ TimePasses
    ∨ ∃ s ∈ SERVERS :
        ∨ ServerStartWrite(s)
        ∨ ServerWriteBlob(s)
        ∨ ServerWriteMetadataAndReturn(s)
        ∨ ServerFailWrite(s)
        ∨ ServerStartRead(s)
        ∨ ServerReadMetadata(s)
```

$\lor$ *ServerReadMetadataAndReturnEmpty*(s)
$\lor$ *ServerReadBlobAndReturn*(s)
$\lor$ *CleanerSteps*

Spec $\triangleq$ $\land$ *Init*
$\land \Box[Next]_{vars}$
$\land \mathrm{WF}_{vars}(CleanerSteps)$ The cleaner will always get to run

Invariants

Note that the success criteria hasn't changed this whole time

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 exists 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

NoOrphanFiles $\triangleq$
There does not exist a key
$\neg\exists\, k \in UUIDS :$
That is in the block store
$\land$ *blobStoreState*$[k] \neq [status \mapsto$ "UNSET", *image* $\mapsto$ "UNSET"$]$
And not in database
$\land \forall\, u \in USERIDS :$
databaseState$[u]$.*imageId* $\neq k$

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

EventuallyNoOrphanFiles $\triangleq \Box\Diamond NoOrphanFiles$

*StopAfter*3*Operations* $\triangleq$

$\land\ Len(operations) \leq 3$
$\land\ time \leq 3$

$StopAfter5Operations \triangleq$
$Len(operations) \leq 5$