MODULE *AggCount*

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

CONSTANTS *Dataset*, *Storage*, *nil*

VARIABLES *replicas*, *pending_counters*

$vars \triangleq \langle replicas, pending\_counters \rangle$

$min\_repl\_id \triangleq 21$
$max\_repl\_id \triangleq 24$

$ReplicaID \triangleq min\_repl\_id \,..\, max\_repl\_id$

$Status \triangleq \{\text{“pending”}, \text{“written”}\}$

$AggStatus \triangleq \{\text{“need\_include”}, \text{“no\_action”}, \text{“need\_remove”}\}$

$ReplicaInfo \triangleq [ds : Dataset, status : Status, storage : Storage, agg : AggStatus]$

$Replica \triangleq [ReplicaID \to ReplicaInfo \cup \{nil\}]$

$PendingKey \triangleq Dataset \times Storage$

$PendingInfo \triangleq [count : 0 \,..\, 100, need\_update : \text{BOOLEAN}\ ]$

$TypeOK \triangleq$
$\quad \land \quad replicas \in Replica$
$\quad \land \quad pending\_counters \in [PendingKey \to PendingInfo]$

$initCounter \triangleq [count \mapsto 0, need\_update \mapsto \text{FALSE}]$

$Init \triangleq$
$\quad \land replicas = [id \in ReplicaID \mapsto nil]$
$\quad \land pending\_counters = [k \in PendingKey \mapsto initCounter]$

$addPendingReplicaImpl(id, ds, st) \triangleq$
$\quad$ LET
$\quad\quad new\_repl \triangleq [$
$\quad\quad\quad ds \mapsto ds, status \mapsto \text{“pending”},$
$\quad\quad\quad storage \mapsto st, agg \mapsto \text{“need\_include”}]$
$\quad\quad key \triangleq \langle ds, st \rangle$
$\quad$ IN
$\quad\quad \land replicas' = [replicas \text{ EXCEPT } ![id] = new\_repl]$
$\quad\quad \land pending\_counters' = [pending\_counters \text{ EXCEPT } ![key].need\_update = \text{TRUE}]$

$AddPendingReplica(id, ds, st) \triangleq$
$\quad \land replicas[id] = nil$

```
    ∧ addPendingReplicaImpl(id, ds, st)

addWrittenReplicaImpl(id, ds, st) ≜
    LET
        new_repl ≜ [
            ds ↦ ds, status ↦ "written",
            storage ↦ st, agg ↦ "no_action"]
        key ≜ ⟨ds, st⟩
    IN
        ∧ replicas' = [replicas EXCEPT ![id] = new_repl]
        ∧ pending_counters' = [pending_counters EXCEPT ![key].need_update = TRUE]

AddWrittenReplica(id, ds, st) ≜
    ∧ replicas[id] = nil
    ∧ addWrittenReplicaImpl(id, ds, st)

updateCounterAfterWritten(r) ≜
    LET
        k ≜ ⟨r.ds, r.storage⟩
    IN
        pending_counters' = [pending_counters EXCEPT ![k].need_update = TRUE]

computeAggStatusForWritten(old_val) ≜
    IF old_val = "no_action"
     THEN "need_remove"
     ELSE "no_action"

doUpdateReplicaToWritten(id) ≜
    LET
        old_repl ≜ replicas[id]

        need_remove_cond ≜
            ∨ ∧ old_repl.status = "pending"
              ∧ old_repl.agg = "no_action"
            ∨ ∧ old_repl.status = "written"
              ∧ old_repl.agg = "need_remove"

        new_agg ≜
            IF need_remove_cond
              THEN "need_remove"
              ELSE "no_action"

        new_repl ≜ [old_repl EXCEPT !.status = "written", !.agg = new_agg]
    IN
```

```
        replicas' = [replicas EXCEPT ![id] = new_repl]

UpdateToWritten(id) ≜
    ∧ replicas[id] ≠ nil
    ∧ replicas[id].status ≠ "written"
    ∧ doUpdateReplicaToWritten(id)
    ∧ updateCounterAfterWritten(replicas[id])

replicaHasKey(id, k) ≜
    ∧ replicas[id] ≠ nil
    ∧ replicas[id].ds = k[1]
    ∧ replicas[id].storage = k[2]

getPendingReplicas(k) ≜
    LET
        selectCond(id) ≜
            ∧ replicaHasKey(id, k)
            ∧ replicas[id].status = "pending"
    IN
        {id ∈ ReplicaID : selectCond(id)}

setAggToNoAction(k) ≜
    LET
        new_fn(id) ≜
            IF replicaHasKey(id, k)
                THEN [replicas[id] EXCEPT !.agg = "no_action"]
                ELSE replicas[id] unchanged
    IN
        replicas' = [id ∈ ReplicaID ↦ new_fn(id)]

doUpdatePendingCounter(k) ≜
    LET
        pending_repls ≜ getPendingReplicas(k)
        num ≜ Cardinality(pending_repls)

        old_counter ≜ pending_counters[k]
        new_counter ≜ [old_counter EXCEPT !.count = num, !.need_update = FALSE]
    IN
        ∧ pending_counters' = [pending_counters EXCEPT ![k] = new_counter]
        ∧ setAggToNoAction(k)

UpdatePendingCounter(k) ≜
    ∧ pending_counters[k].need_update = TRUE
    ∧ doUpdatePendingCounter(k)
```

$TerminateCond \triangleq$
$\land \forall\, id \in ReplicaID :$
$\land\ replicas[id] \neq nil$
$\land\ replicas[id].agg =$ "no_action"
$\land \forall\, key \in PendingKey : pending\_counters[key].need\_update =$ FALSE

$Terminated \triangleq$
$\land\ TerminateCond$
$\land$ UNCHANGED $vars$

$Next \triangleq$
$\lor \exists\, id \in ReplicaID,\ ds \in Dataset,\ st \in Storage :$
$\lor\ AddPendingReplica(id,\ ds,\ st)$
$\lor\ AddWrittenReplica(id,\ ds,\ st)$
$\lor \exists\, id \in ReplicaID :$
$UpdateToWritten(id)$
$\lor \exists\, k \in PendingKey :$
$UpdatePendingCounter(k)$
$\lor\ Terminated$

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

$FairSpec \triangleq\ Spec \land \mathrm{WF}_{vars}(Next)$

$AlwaysTerminate \triangleq \Diamond TerminateCond$

$allPendingReplicas(k) \triangleq$
LET
$checkCond(id) \triangleq$
$\land\ replicaHasKey(id,\ k)$
$\land\ replicas[id].status =$ "pending"
$S \triangleq \{id \in ReplicaID : checkCond(id)\}$
IN
$Cardinality(S)$

$numPendingByCounter(k) \triangleq$
LET
$isPending(id) \triangleq$
$\land\ replicaHasKey(id,\ k)$
$\land\ replicas[id].agg =$ "need_include"

$isNonPending(id) \triangleq$
$\land\ replicaHasKey(id,\ k)$
$\land\ replicas[id].agg =$ "need_remove"

$S1 \triangleq \{id \in ReplicaID : isPending(id)\}$
$S2 \triangleq \{id \in ReplicaID : isNonPending(id)\}$

IN

$Cardinality(S1) + pending\_counters[k].count - Cardinality(S2)$

$invalidStatePending \triangleq$
$\exists\, id \in ReplicaID :$
$\land\, replicas[id] \neq nil$
$\land\, replicas[id].status = \text{“pending”}$
$\land\, replicas[id].agg = \text{“need\_remove”}$

$invalidStateWritten \triangleq$
$\exists\, id \in ReplicaID :$
$\land\, replicas[id] \neq nil$
$\land\, replicas[id].status = \text{“written”}$
$\land\, replicas[id].agg = \text{“need\_include”}$

$Inv \triangleq$
$\land\, \forall\, k \in PendingKey :$
$allPendingReplicas(k) = numPendingByCounter(k)$
$\land\, \neg invalidStatePending$
$\land\, \neg invalidStateWritten$

$Sym \triangleq Permutations(Dataset) \cup Permutations(Storage)$