MODULE *UpdateCluster*

EXTENDS *Integers*, *FiniteSets*

CONSTANTS
- $\_Requests$, the requests sent by the user
- $\_Workers$, the pool of workers
- $NULL$

VARIABLES
- $lastVOK$, last successfully applied version
- $toApply$, the version to apply (last request that passed the initial tests)
- $cluster$, cluster state
- $requests$, the state of all requests
- $workers$, the state of all workers
- $clusterUpdating$ damn, I use a lock...

VARIABLES
- these variables are *tla+* details
- $confOK$, are we able to get a valid conf ?
- $reqCounter$ just to keep track of the order of submissions

$vars \triangleq \langle confOK, reqCounter, lastVOK, toApply, cluster, requests, workers, clusterUpdating \rangle$

$TypeInvariants \triangleq$
- $\land\ confOK \in$ BOOLEAN won't change for a specific behavior
- $\land\ clusterUpdating \in$ BOOLEAN
- $\land\ cluster.st \in \{$
  - "idle",
  - "starting",
  - "partial",
  - "failed"
  - $\}$
- $\land \forall\, r \in \_Requests : requests[r].st \in \{$
  - "waiting", the request (req) hasn't been submitted yet
  - "submitted", req has been submitted
  - "rejected", req has been rejected (auth problem)
  - "valid" auth etc passed
  - $\}$
- $\land \forall\, w \in \_Workers : workers[w].st \in \{$
  - "waiting",
  - "starting",
  - "working"
  - $\}$

Initial State

$Init \triangleq$
$\land\ requests = [r \in \_Requests \mapsto [st \mapsto \text{“waiting”},\ v \mapsto NULL]]$
$\land\ workers\ = [w \in \_Workers \mapsto [st \mapsto \text{“waiting”},\ v \mapsto NULL]]$
$\land\ cluster\ = [v \mapsto 0,\ st \mapsto \text{“idle”}]$
$\land\ lastVOK = 0$
$\land\ reqCounter = 0$
$\land\ toApply = 0$
$\land\ confOK \in \text{BOOLEAN}$
$\land\ clusterUpdating = \text{FALSE}$

Actions

$Submit(r) \triangleq$
update request received from the user
LET $newV \triangleq reqCounter + 1$ IN
$\land\ requests[r].st = \text{“waiting”}$
$\land\ reqCounter' = newV$
$\land\ requests' = [requests\ \text{EXCEPT}\ ![r].st = \text{“submitted”},\ ![r].v = newV]$
$\land\ \text{UNCHANGED}\ \langle confOK,\ lastVOK,\ toApply,\ cluster,\ workers,\ clusterUpdating\rangle$

$PushToPending(r) \triangleq$
the request is pushed to queue
$\land\ requests[r].st = \text{“submitted”}$
$\land\ \text{IF}\ toApply < requests[r].v$
THEN $\land\ requests' = [requests\ \text{EXCEPT}\ ![r].st = \text{“valid”}]$
$\land\ toApply' = requests[r].v$
$\land\ \text{UNCHANGED}\ \langle confOK,\ reqCounter,\ lastVOK,\ cluster,\ workers,\ clusterUpdating\rangle$
ELSE $\land\ requests' = [requests\ \text{EXCEPT}\ ![r].st = \text{“rejected”}]$
$\land\ \text{UNCHANGED}\ \langle confOK,\ reqCounter,\ lastVOK,\ toApply,\ cluster,\ workers,\ clusterUpdating\rangle$

$SpawnWorker(w) \triangleq$
spawns a new worker
$\land\ workers[w].st = \text{“waiting”}$
$\land\ toApply \neq lastVOK$
$\land\ clusterUpdating = \text{FALSE}$
$\land\ \ \lor\ cluster.st = \text{“idle”}$
$\lor\ cluster.st = \text{“failed”}$
$\land\ workers' = [workers\ \text{EXCEPT}\ ![w].v = toApply,\ ![w].st = \text{“starting”}]$
$\land\ clusterUpdating' = \text{TRUE}$
$\land\ \text{UNCHANGED}\ \langle confOK,\ reqCounter,\ lastVOK,\ toApply,\ requests,\ cluster\rangle$

$ApplyStart(w) \triangleq$
the cluster starts to be modified

$\land$ $workers[w].st$ = "starting"
$\land$ IF $workers[w].v \geq toApply$
THEN
IF $confOK$
THEN
$\land$ $cluster'$ = $[v \mapsto workers[w].v,\ st \mapsto$ "partial"$]$
$\land$ $workers'$ = $[workers$ EXCEPT $![w].st$ = "working"$]$
$\land$ UNCHANGED $\langle confOK,\ reqCounter,\ lastVOK,\ toApply,\ requests,\ clusterUpdating\rangle$
ELSE
$\land$ $clusterUpdating'$ = FALSE
$\land$ $workers'$ = $[workers$ EXCEPT $![w].st$ = "waiting", $![w].v = NULL]$
$\land$ UNCHANGED $\langle confOK,\ reqCounter,\ lastVOK,\ toApply,\ cluster,\ requests\rangle$
ELSE a new version has been submitted, no need to apply this one
$\land$ $workers'$ = $[workers$ EXCEPT $![w].st$ = "waiting", $![w].v = NULL]$
$\land$ $clusterUpdating'$ = FALSE
$\land$ UNCHANGED $\langle confOK,\ reqCounter,\ lastVOK,\ toApply,\ cluster,\ requests\rangle$

$RollbackVersion \triangleq$
to differenciate it from the original last *VOK* (in realworld, could be *conf* + timestamp)
$lastVOK + 10$

$ApplyFinish(w) \triangleq$
the cluster update finishes
$\land$ $workers[w].st$ = "working"
$\land$ $clusterUpdating'$ = FALSE
$\land$ $\exists\, ok \in$ BOOLEAN :
IF $ok \lor workers[w].v = RollbackVersion$ rollback always works
THEN
$\land$ $cluster'$ = $[cluster$ EXCEPT $!.st$ = "idle"$]$
$\land$ $lastVOK'$ = $workers[w].v$
$\land$ $workers'$ = $[workers$ EXCEPT $![w].st$ = "waiting", $![w].v = NULL]$
$\land$ UNCHANGED $\langle confOK,\ reqCounter,\ toApply,\ requests\rangle$
ELSE
$\land$ $cluster'$ = $[cluster$ EXCEPT $!.st$ = "failed"$]$
$\land$ $workers'$ = $[workers$ EXCEPT $![w].st$ = "waiting", $![w].v = NULL]$
$\land$ IF $workers[w].v < toApply$
THEN a newer version has been submitted
$\land$ UNCHANGED $\langle confOK,\ reqCounter,\ lastVOK,\ toApply,\ requests\rangle$
ELSE let's trigger a rollback
$\land$ $toApply'$ = $RollbackVersion$
$\land$ UNCHANGED $\langle confOK,\ reqCounter,\ lastVOK,\ requests\rangle$

Spec

$Next \triangleq$
$\lor \exists\, r \in \_Requests :$
$\lor Submit(r)$
$\lor PushToPending(r)$
$\lor \exists\, w \in \_Workers :$
$\lor SpawnWorker(w)$
$\lor ApplyStart(w)$
$\lor ApplyFinish(w)$

$Fairness \triangleq \forall\, r \in \_Requests,\ w \in \_Workers :$
$\land \mathrm{WF}_{vars}(Submit(r))$
$\land \mathrm{WF}_{vars}(PushToPending(r))$
$\land \mathrm{WF}_{vars}(SpawnWorker(w))$
$\land \mathrm{WF}_{vars}(ApplyStart(w))$
$\land \mathrm{WF}_{vars}(ApplyFinish(w))$

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

Expectations

$NoConcurrentUpdate \triangleq$
$\Box(Cardinality(\{w \in \text{DOMAIN } workers : workers[w].st = \text{“working”}\}) < 2)$

$NoPartialUpdateTermination \triangleq$
we don't want the cluster to end up in a partially update state
$\Diamond\Box(cluster.st = \text{“idle”})$

$EveryReqIsProcessed \triangleq$
$\Diamond\Box(\neg\exists\, r \in \_Requests : requests[r].st = \text{“waiting”})$

THEOREM $Spec \Rightarrow \Box(TypeInvariants)$
THEOREM $Spec \Rightarrow NoConcurrentUpdate$
THEOREM $Spec \Rightarrow NoPartialUpdateTermination$
THEOREM $Spec \Rightarrow EveryReqIsProcessed$