MODULE *Hermes*

EXTENDS *Integers*, *FiniteSets*

CONSTANTS *H_NODES*, *H_MAX_VERSION*

VARIABLES *msgs*, *nodeTS*, *nodeState*, *nodeRcvedAcks*, *nodeLastWriter*, *nodeLastWriteTS*, *nodeWriteEpochID*, *aliveNodes*, *epochID*

The consistent invariant: all alive nodes in valid state should have the same value / *TS*

$$HConsistent \triangleq \forall\, k, s \in aliveNodes : \lor\, nodeState[k] \neq \text{“valid”} \lor\, nodeState[s] \neq \text{“valid”} \lor\, nodeTS[k] = nodeTS[s]$$

$$HMessage \triangleq [type : \{\text{“INV”}, \text{“ACK”}\},\ sender : H\_NODES,\ epochID : 0\,..\,(Cardinality(H\_NODES)-1),\ version : 0\,..\,H\_MAX\_VERSION,\ tieBreaker : H\_NODES] \cup [type : \{\text{“VAL”}\},\ version : 0\,..\,H\_MAX\_VERSION,\ tieBreaker : H\_NODES]$$

Messages exchanged by the Protocol

$$\begin{aligned} HTypeOK \triangleq\ & \land\ msgs \subseteq HMessage \\ & \land \forall\, n \in H\_NODES : nodeRcvedAcks[n] \subseteq (H\_NODES \setminus \{n\}) \\ & \land\ nodeLastWriter \in [H\_NODES \to H\_NODES] \\ & \land\ nodeLastWriteTS \in [H\_NODES \to [version : 0\,..\,H\_MAX\_VERSION,\ tieBreaker : H\_NODES]] \\ & \land\ nodeTS \in [H\_NODES \to [version : 0\,..\,H\_MAX\_VERSION,\ tieBreaker : H\_NODES]] \\ & \land\ nodeState \in [H\_NODES \to \{\text{“valid”}, \text{“invalid”}, \text{“invalid\_write”}, \text{“write”}, \text{“replay”}\}] \end{aligned}$$

The type correctness invariant

membership and epoch id related

$$\begin{aligned} & \land\ aliveNodes \subseteq H\_NODES \\ & \land\ epochID \in 0\,..\,(Cardinality(H\_NODES)-1) \end{aligned}$$

$\land$ $nodeWriteEpochID \in [H\_NODES \to 0 \,..\, (Cardinality(H\_NODES) - 1)]$

$HInit \triangleq$ The initial predicate

$\land$ $msgs = \{\}$

membership and epoch id related

$\land$ $epochID = 0$

$\land$ $aliveNodes = H\_NODES$

$\land$ $nodeWriteEpochID = [n \in H\_NODES \mapsto 0]$

Init rest per node replica metadata

$\land$ $nodeRcvedAcks = [n \in H\_NODES \mapsto \{\}]$

$\land$ $nodeState = [n \in H\_NODES \mapsto$ "valid"$]$

$\land$ $nodeLastWriter = [n \in H\_NODES \mapsto \text{CHOOSE } k \in H\_NODES : \forall\, m \in H\_NODES : k \leq m]$

$\land$ $nodeTS = [n \in H\_NODES \mapsto [version \mapsto 0, tieBreaker \mapsto \text{CHOOSE } k \in H\_NODES : \forall\, m \in H\_NODES : k \leq m]]$

$\land$ $nodeLastWriteTS = [n \in H\_NODES \mapsto [version \mapsto 0, tieBreaker \mapsto \text{CHOOSE } k \in H\_NODES : \forall\, m \in H\_NODES : k \leq m]]$

A buffer maintaining all network messages. Messages are only appended to this variable (not removed once delivered) intentionally to check protocols tolerance in dublicates and reorderings

$send(m) \triangleq msgs' = msgs \cup \{m\}$

Check if all acknowledgments for a write have been received

$receivedAllAcks(n) \triangleq nodeRcvedAcks[n] = H\_NODES \setminus \{n\}$

$receivedAllAcks(n) \triangleq (aliveNodes \setminus \{n\}) \subseteq nodeRcvedAcks[n]$

$equalTS(v1,\, tb1,\, v2,\, tb2) \triangleq$ Timestamp equality

$\land$ $v1 = v2$

$\land$ $tb1 = tb2$

$greaterTS(v1,\, tb1,\, v2,\, tb2) \triangleq$ Timestamp comparison

$\lor$ $v1 > v2$

$\lor \land\ v1 = v2$

$\land\ tb1 > tb2$

$isAlive(n) \triangleq n \in aliveNodes$

$nodeFailure(n) \triangleq$ Emulate a node failure

Make sure that there are atleast 3 alive nodes before killing a node

$\land \exists\, k,\, m \in aliveNodes : \land\, k \neq n \land\, m \neq n$

$\wedge\ m \neq k$

$\wedge\ aliveNodes' = aliveNodes \setminus \{n\}$

$\wedge\ epochID = epochID + 1$

$\wedge$ UNCHANGED $\langle msgs,\ nodeState,\ nodeTS,\ nodeLastWriter,$
$nodeLastWriteTS,\ nodeRcvedAcks,\ nodeWriteEpochID\rangle$

$h\_upd\_not\_aliveNodes \triangleq$

$\wedge$ UNCHANGED $\langle aliveNodes,\ epochID,\ nodeWriteEpochID\rangle$

$h\_upd\_aliveNodes \triangleq$

$\wedge$ UNCHANGED $\langle msgs,\ nodeState,\ nodeTS,\ nodeLastWriter,$
$nodeLastWriteTS,\ nodeRcvedAcks,\ epochID,\ nodeWriteEpochID\rangle$

$h\_upd\_nothing \triangleq$

$\wedge\ h\_upd\_not\_aliveNodes$

$\wedge\ h\_upd\_aliveNodes$

$h\_upd\_state(n,\ newVersion,\ newTieBreaker,\ newState,\ newAcks) \triangleq$

$\wedge\ nodeLastWriter' = [nodeLastWriter$ EXCEPT $![n] = n]$

$\wedge\ nodeRcvedAcks' = [nodeRcvedAcks$ EXCEPT $![n] = newAcks]$

$\wedge\ nodeState' = [nodeState$ EXCEPT $![n] = newState]$

$\wedge\ nodeWriteEpochID' = [nodeWriteEpochID$ EXCEPT $![n] = epochID]$ we always use the latest *epochID*

$\wedge\ nodeTS' = [nodeTS$ EXCEPT $![n].version = newVersion,$
$![n].tieBreaker = newTieBreaker]$

$\wedge\ nodeLastWriteTS' = [nodeLastWriteTS$ EXCEPT $![n].version = newVersion,$
$![n].tieBreaker = newTieBreaker]$

$h\_send\_inv\_or\_ack(n,\ newVersion,\ newTieBreaker,\ msgType) \triangleq$

$\wedge\ send([type \mapsto msgType,$
$epochID \mapsto epochID,$ we always use the latest *epochID*
$sender \mapsto n,$
$version \mapsto newVersion,$
$tieBreaker \mapsto newTieBreaker])$

$h\_upd\_actions(n,\ newVersion,\ newTieBreaker,\ newState,\ newAcks) \triangleq$ Execute a write

$\wedge\ h\_upd\_state(n,\ newVersion,\ newTieBreaker,\ newState,\ newAcks)$

$\wedge\ h\_send\_inv\_or\_ack(n,\ newVersion,\ newTieBreaker,\ \text{“INV”})$

$\wedge$ UNCHANGED $\langle aliveNodes,\ epochID\rangle$

$h\_upd\_replay\_actions(n,\ acks) \triangleq$ Apply a write-replay using same *TS* (version, Tie Breaker) and either reset *acks* or keep already gathered *acks*

$\wedge\ h\_upd\_actions(n,\ nodeTS[n].version,\ nodeTS[n].tieBreaker,\ \text{“replay”},\ acks)$

$HRead(n) \triangleq$ Execute a read
$\land\ nodeState[n] = \text{“valid”}$
$\land\ h\_upd\_nothing$

$HWrite(n) \triangleq$ Execute a write
$\land\ nodeState[n] \in \{\text{“valid”}, \text{“invalid”}\}$
writes in invalid state are also supported as an optimization
$\land\ nodeState[n] \in \{\text{“valid”}\}$
$\land\ nodeTS[n].version < H\_MAX\_VERSION$
$\land\ h\_upd\_actions(n,\ nodeTS[n].version + 1,\ n,\ \text{“write”},\ \{\})$

$HCoordWriteReplay(n) \triangleq$ Execute a write-replay after a membership re-config
$\land\ nodeState[n] \in \{\text{“write”}, \text{“replay”}\}$
$\land\ nodeWriteEpochID[n] < epochID$
$\land\ \neg receivedAllAcks(n)$ optimization to not replay when we have gathered *acks* from all alive
$\land\ h\_upd\_replay\_actions(n,\ nodeRcvedAcks[n])$

$HRcvAck(n) \triangleq$ Process a received acknowledment
$\exists\, m \in msgs :$
$\land\ m.type = \text{“ACK”}$
$\land\ m.epochID = epochID$
$\land\ m.sender \neq n$
$\land\ m.sender \notin nodeRcvedAcks[n]$
$\land\ equalTS(m.version,$
$m.tieBreaker,$
$nodeLastWriteTS[n].version,$
$nodeLastWriteTS[n].tieBreaker)$
$\land\ nodeState[n] \in \{\text{“write”}, \text{“invalid\_write”}, \text{“replay”}\}$
$\land\ nodeRcvedAcks' = [nodeRcvedAcks \text{ EXCEPT } ![n] =$
$nodeRcvedAcks[n] \cup \{m.sender\}]$
$\land$ UNCHANGED $\langle msgs,\ nodeLastWriter,\ nodeLastWriteTS,$
$aliveNodes,\ nodeTS,\ nodeState,\ epochID,\ nodeWriteEpochID\rangle$

$HSendVals(n) \triangleq$ Send validations once received acknowledments from all alive nodes
$\land\ nodeState[n] \in \{\text{“write”}, \text{“replay”}\}$
$\land\ receivedAllAcks(n)$
$\land\ nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“valid”}]$
$\land\ send([type \mapsto \text{“VAL”},$
$version \mapsto nodeTS[n].version,$
$tieBreaker \mapsto nodeTS[n].tieBreaker])$
$\land$ UNCHANGED $\langle nodeTS,\ nodeLastWriter,\ nodeLastWriteTS,$
$aliveNodes,\ nodeRcvedAcks,\ epochID,\ nodeWriteEpochID\rangle$

$HCoordinatorActions(n) \triangleq$ Actions of a read/write coordinator

$\lor\ HRead(n)$
$\lor\ HCoordWriteReplay(n)$
$\lor\ HWrite(n)$
$\lor\ HRcvAck(n)$
$\lor\ HSendVals(n)$

$HRcvInv(n) \triangleq$ Process a received invalidation
$\exists\, m \in msgs :$
$\land\ m.type = \text{“INV”}$
$\land\ m.epochID = epochID$
$\land\ m.sender \neq n$
always acknowledge a received invalidation (irrelevant to the timestamp)
$\land\ send([type \mapsto \text{“ACK”},$
$sender \mapsto n,$
$epochID \mapsto epochID,$
$version \mapsto m.version,$
$tieBreaker \mapsto m.tieBreaker])$
$\land\ \lor\ \land\ greaterTS(m.version,$
$m.tieBreaker,$
$nodeTS[n].version,$
$nodeTS[n].tieBreaker)$
$\land\ nodeLastWriter' = [nodeLastWriter \text{ EXCEPT } ![n] = m.sender]$
$\land\ nodeTS' = [nodeTS \text{ EXCEPT } ![n].version = m.version,$
$![n].tieBreaker = m.tieBreaker]$
$\land\ \lor\ \land\ nodeState[n] \in \{\text{“valid”}, \text{“invalid”}, \text{“replay”}\}$
$\land\ nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“invalid”}]$
$\lor\ \land\ nodeState[n] \in \{\text{“write”}, \text{“invalid\_write”}\}$
$\land\ nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“invalid\_write”}]$
$\lor\ \land\ \neg greaterTS(m.version,$
$m.tieBreaker,$
$nodeTS[n].version,$
$nodeTS[n].tieBreaker)$
$\land\ \text{UNCHANGED } \langle nodeState, nodeTS, nodeLastWriter, nodeWriteEpochID\rangle$
$\land\ \text{UNCHANGED } \langle nodeLastWriteTS, aliveNodes, nodeRcvedAcks, epochID, nodeWriteEpochID\rangle$

$HRcvVal(n) \triangleq$ Process a received validation
$\exists\, m \in msgs :$
$\land\ nodeState[n] \neq \text{“valid”}$
$\land\ m.type = \text{“VAL”}$
$\land\ equalTS(m.version,$
$m.tieBreaker,$
$nodeTS[n].version,$
$nodeTS[n].tieBreaker)$

$\land\ nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“valid”}]$
$\land$ UNCHANGED $\langle msgs,\ nodeTS,\ nodeLastWriter,\ nodeLastWriteTS,$
$aliveNodes,\ nodeRcvedAcks,\ epochID,\ nodeWriteEpochID\rangle$

$HFollowerWriteReplay(n) \triangleq$ Execute a write-replay when coordinator failed
$\land\ nodeState[n] = \text{“invalid”}$
$\land\ \neg isAlive(nodeLastWriter[n])$
$\land\ h\_upd\_replay\_actions(n, \{\})$

$HFollowerActions(n) \triangleq$ Actions of a write follower
$\lor\ HRcvInv(n)$
$\lor\ HFollowerWriteReplay(n)$
$\lor\ HRcvVal(n)$

$HNext \triangleq$ Modeling *Hermes* protocol (Coordinator and Follower actions while emulating failures)
$\exists\, n \in aliveNodes :$
$\lor\ HFollowerActions(n)$
$\lor\ HCoordinatorActions(n)$
$\lor\ nodeFailure(n)$ emulate node failures
