MODULE *Hermes*

EXTENDS *Integers*, *FiniteSets*

CONSTANTS *H_NODES*, *H_MAX_VERSION*

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

all *Hermes*(+ environment) variables

$hvars \triangleq \langle msgs, nodeTS, nodeState, nodeRcvedAcks, nodeLastWriter, nodeLastWriteTS, nodeWriteEpochID, aliveNodes, epochID\rangle$

$HMessage \triangleq$ Messages exchanged by the Protocol

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

Note that we need not send Value w/ *INVs*, timestamp suffice to check consistency

$\cup$

$[type : \{\text{“VAL”}\},$ optimization: *epochID* is not required for *VALs* $epochID : 0 \,..\, (Cardinality(H\_NODES) - 1), version : 0 \,..\, H\_MAX\_VERSION, tieBreaker : H\_NODES]$

$HTypeOK \triangleq$ The type correctness invariant

$\land\ msgs \subseteq HMessage$

$\land \forall n \in H\_NODES : nodeRcvedAcks[n] \subseteq (H\_NODES \setminus \{n\})$

$\land\ nodeLastWriter \in [H\_NODES \to H\_NODES]$

$\land\ nodeLastWriteTS \in [H\_NODES \to [version : 0 \,..\, H\_MAX\_VERSION, tieBreaker : H\_NODES]]$

$\land\ nodeTS \in [H\_NODES \to [version : 0 \,..\, H\_MAX\_VERSION, tieBreaker : H\_NODES]]$

$\land\ nodeState \in [H\_NODES \to \{\text{“valid”}, \text{“invalid”}, \text{“invalid\_write”}, \text{“write”}, \text{“replay”}\}]$

membership and epoch id related

$\land\ aliveNodes \subseteq H\_NODES$

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

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

$HConsistent \triangleq$
$\forall\, k,\, s \in aliveNodes :$ $\lor\ nodeState[k] \neq$ "valid"
$\lor\ nodeState[s] \neq$ "valid"
$\lor\ nodeTS[k] = nodeTS[s]$

$HInit \triangleq$ 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"$]$
$\land$ $nodeLastWriter = [n \in H\_NODES \mapsto$ CHOOSE $k \in H\_NODES :$
$\forall\, m \in H\_NODES : k \leq m]$
$\land$ $nodeTS = [n \in H\_NODES \mapsto [version \mapsto 0,$
$tieBreaker \mapsto$
CHOOSE $k \in H\_NODES :$
$\forall\, m \in H\_NODES : k \leq m]]$
$\land$ $nodeLastWriteTS = [n \in H\_NODES \mapsto [version \mapsto 0,$
$tieBreaker \mapsto$
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

$\land\ Cardinality(aliveNodes) > 2$
$\land\ aliveNodes' = aliveNodes \setminus \{n\}$
$\land\ epochID = epochID + 1$
$\land$ UNCHANGED $\langle msgs, nodeState, nodeTS, nodeLastWriter, nodeLastWriteTS, nodeRcvedAcks, nodeWriteEpochID\rangle$

$h\_upd\_not\_aliveNodes \triangleq$
$\land$ UNCHANGED $\langle aliveNodes, epochID, nodeWriteEpochID\rangle$

$h\_upd\_aliveNodes \triangleq$
$\land$ UNCHANGED $\langle msgs, nodeState, nodeTS, nodeLastWriter, nodeLastWriteTS, nodeRcvedAcks, epochID, nodeWriteEpochID\rangle$

$h\_upd\_nothing \triangleq$
$\land\ h\_upd\_not\_aliveNodes$
$\land\ h\_upd\_aliveNodes$

$h\_upd\_state(n, newVersion, newTieBreaker, newState, newAcks) \triangleq$
$\land\ nodeLastWriter' = [nodeLastWriter$ EXCEPT $![n] = n]$
$\land\ nodeRcvedAcks' = [nodeRcvedAcks$ EXCEPT $![n] = newAcks]$
$\land\ nodeState' = [nodeState$ EXCEPT $![n] = newState]$
$\land\ nodeWriteEpochID' = [nodeWriteEpochID$ EXCEPT $![n] = epochID]$ we always use the latest *epochID*
$\land\ nodeTS' = [nodeTS$ EXCEPT $![n].version = newVersion,$ $![n].tieBreaker = newTieBreaker]$
$\land\ nodeLastWriteTS' = [nodeLastWriteTS$ EXCEPT $![n].version = newVersion,$ $![n].tieBreaker = newTieBreaker]$

$h\_send\_inv\_or\_ack(n, newVersion, newTieBreaker, msgType) \triangleq$
$\land\ send([type \mapsto msgType,$
$epochID \mapsto epochID,$ we always use the latest *epochID*
$sender \mapsto n,$
$version \mapsto newVersion,$
$tieBreaker \mapsto newTieBreaker])$

$h\_actions\_for\_upd(n, newVersion, newTieBreaker, newState, newAcks) \triangleq$ Execute a write
$\land\ h\_upd\_state(n, newVersion, newTieBreaker, newState, newAcks)$
$\land\ h\_send\_inv\_or\_ack(n, newVersion, newTieBreaker,$ "INV")
$\land$ UNCHANGED $\langle aliveNodes, epochID\rangle$

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

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

$HRead(n) \triangleq$ Execute a read

$\quad \land\ nodeState[n] = \text{“valid”}$

$\quad \land\ h\_upd\_nothing$

$HWrite(n) \triangleq$ Execute a write

$\quad \land\ nodeState[n] \in \{\text{“valid”},\ \text{“invalid”}\}$ writes in invalid state are also supported as an optimization

$\quad \land\ nodeState[n] \in \{\text{“valid”}\}$

$\quad \land\ nodeTS[n].version < H\_MAX\_VERSION$ Only to configurably terminate the model checking

$\quad \land\ h\_actions\_for\_upd(n,\ nodeTS[n].version + 1,\ n,\ \text{“write”},\ \{\})$

$HCoordWriteReplay(n) \triangleq$ Execute a write-replay after a membership re-config

$\quad \land\ nodeState[n] \in \{\text{“write”},\ \text{“replay”}\}$

$\quad \land\ nodeWriteEpochID[n] < epochID$

$\quad \land\ \neg receivedAllAcks(n)$ optimization to not replay when we have gathered *acks* from all alive

$\quad \land\ h\_actions\_for\_upd\_replay(n,\ nodeRcvedAcks[n])$

$HRcvAck(n) \triangleq$ Process a received acknowledment

$\quad \exists\, m \in msgs :$

$\quad\quad \land\ m.type = \text{“ACK”}$

$\quad\quad \land\ m.epochID = epochID$

$\quad\quad \land\ m.sender \neq n$

$\quad\quad \land\ m.sender \notin nodeRcvedAcks[n]$

$\quad\quad \land\ equalTS(m.version,\ m.tieBreaker,$
$\quad\quad\quad\quad nodeLastWriteTS[n].version,$
$\quad\quad\quad\quad nodeLastWriteTS[n].tieBreaker)$

$\quad\quad \land\ nodeState[n] \in \{\text{“write”},\ \text{“invalid\_write”},\ \text{“replay”}\}$

$\quad\quad \land\ nodeRcvedAcks' = [nodeRcvedAcks \text{ EXCEPT } ![n] =$
$\quad\quad\quad\quad nodeRcvedAcks[n] \cup \{m.sender\}]$

$\quad\quad \land\ \text{UNCHANGED } \langle msgs,\ nodeLastWriter,\ nodeLastWriteTS,$
$\quad\quad\quad\quad aliveNodes,\ nodeTS,\ nodeState,\ epochID,\ nodeWriteEpochID\rangle$

$HSendVals(n) \triangleq$ Send validations once acknowledments are received from all alive nodes

$\quad \land\ nodeState[n] \in \{\text{“write”},\ \text{“replay”}\}$

$\quad \land\ receivedAllAcks(n)$

$\quad \land\ nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{“valid”}]$

$\quad \land\ send([type \mapsto \text{“VAL”},$
$\quad\quad\quad version \mapsto nodeTS[n].version,$
$\quad\quad\quad tieBreaker \mapsto nodeTS[n].tieBreaker])$

$\land$ UNCHANGED $\langle nodeTS,\ nodeLastWriter,\ nodeLastWriteTS,$
$aliveNodes,\ nodeRcvedAcks,\ epochID,\ nodeWriteEpochID\rangle$

$HCoordinatorActions(n) \triangleq$ Actions of a read/write coordinator
$\lor\ HRead(n)$
$\lor\ HCoordWriteReplay(n)$ After failures
$\lor\ HWrite(n)$
$\lor\ HRcvAck(n)$
$\lor\ HSendVals(n)$

$HRcvInv(n) \triangleq$ 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)
$\land\ send([type \mapsto \text{“ACK”},$
$sender \mapsto n,$
$epochID \mapsto epochID,$
$version \mapsto m.version,$
$tieBreaker \mapsto m.tieBreaker])$
$\land$ IF $greaterTS(m.version,\ m.tieBreaker,$
$nodeTS[n].version,\ nodeTS[n].tieBreaker)$
THEN $\land\ nodeLastWriter' = [nodeLastWriter$ EXCEPT $![n] = m.sender]$
$\land\ nodeTS' = [nodeTS$ EXCEPT $![n].version = m.version,$
$![n].tieBreaker = m.tieBreaker]$
$\land$ IF $nodeState[n] \in \{\text{“valid”},\ \text{“invalid”},\ \text{“replay”}\}$
THEN
$nodeState' = [nodeState$ EXCEPT $![n] = \text{“invalid”}]$
ELSE
$nodeState' = [nodeState$ EXCEPT $![n] = \text{“invalid\_write”}]$
ELSE
UNCHANGED $\langle nodeState,\ nodeTS,\ nodeLastWriter,\ nodeWriteEpochID\rangle$
$\land$ UNCHANGED $\langle nodeLastWriteTS,\ aliveNodes,\ nodeRcvedAcks,\ epochID,\ nodeWriteEpochID\rangle$

$HRcvVal(n) \triangleq$ Process a received validation
$\exists\, m \in msgs :$
$\land\ nodeState[n] \neq \text{“valid”}$
$\land\ m.type = \text{“VAL”}$
$\land\ equalTS(m.version,\ m.tieBreaker,$
$nodeTS[n].version,$
$nodeTS[n].tieBreaker)$
$\land\ nodeState' = [nodeState$ 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
$\land\ nodeState[n] =$ "invalid"
$\land\ \neg isAlive(nodeLastWriter[n])$
$\land\ h\_actions\_for\_upd\_replay(n, \{\})$

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

$HNext \triangleq$ *Hermes* (read/write) protocol (Coordinator and Follower actions) + failures
$\exists\, n \in aliveNodes :$
$\lor\ HFollowerActions(n)$
$\lor\ HCoordinatorActions(n)$
$\lor\ nodeFailure(n)$

$HSpec \triangleq HInit \land \Box[HNext]_{hvars}$

THEOREM $HSpec \Rightarrow (\Box HTypeOK) \land (\Box HConsistent)$
