

| | |
|--|---|
| MODULE <i>Hermes</i> | |
| EXTENDS | <i>Integers</i> , <i>FiniteSets</i> |
| CONSTANTS | <i>H_NODES</i> , <i>H_MAX_VERSION</i> |
| VARIABLES | <i>msgs</i> , <i>nodeTS</i> , <i>nodeState</i> , <i>nodeRcvdAcks</i> , <i>nodeLastWriter</i> , <i>nodeLastWriteTS</i> , <i>nodeWriteEpochID</i> , <i>aliveNodes</i> , <i>epochID</i> |
| all <i>Hermes</i> (+ environment) variables | |
| <i>hvars</i> \triangleq | $\langle \textit{msgs}, \textit{nodeTS}, \textit{nodeState}, \textit{nodeRcvdAcks}, \textit{nodeLastWriter},$ $\textit{nodeLastWriteTS}, \textit{nodeWriteEpochID}, \textit{aliveNodes}, \textit{epochID} \rangle$ |
| $HMessage \triangleq$ Messages exchanged by the Protocol | |
| $[type : \{ "INV", "ACK" \}, sender : H_NODES,$ $epochID : 0 \dots (Cardinality(H_NODES) - 1),$ $version : 0 \dots H_MAX_VERSION,$ $tieBreaker : H_NODES]$ | |
| Note that we need not send Value w/ <i>INV</i> s, timestamp suffice to check consistency | |
| \cup | |
| $[type : \{ "VAL" \},$ $epochID : 0 \dots (Cardinality(H_NODES) - 1),$ $version : 0 \dots H_MAX_VERSION,$ $tieBreaker : H_NODES]$ | |
| $HTypeOK \triangleq$ The type correctness invariant | |
| $\wedge \textit{msgs} \subseteq HMessage$ $\wedge \forall n \in H_NODES : \textit{nodeRcvdAcks}[n] \subseteq (H_NODES \setminus \{n\})$ $\wedge \textit{nodeLastWriter} \in [H_NODES \rightarrow H_NODES]$ $\wedge \textit{nodeLastWriteTS} \in [H_NODES \rightarrow [version : 0 \dots H_MAX_VERSION,$ $tieBreaker : H_NODES]]$ $\wedge \textit{nodeTS} \in [H_NODES \rightarrow [version : 0 \dots H_MAX_VERSION,$ $tieBreaker : H_NODES]]$ $\wedge \textit{nodeState} \in [H_NODES \rightarrow \{ "valid", "invalid", "invalid_write",$ $"write", "replay" \}]$ | |
| membership and epoch id related | |
| $\wedge \textit{aliveNodes} \subseteq H_NODES$ | |

$$\begin{aligned} \wedge \text{ epochID} & \in 0 \dots (\text{Cardinality}(H_NODES) - 1) \\ \wedge \text{ nodeWriteEpochID} & \in [H_NODES \rightarrow 0 \dots (\text{Cardinality}(H_NODES) - 1)] \end{aligned}$$

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

$$\begin{aligned} \forall k, s \in \text{aliveNodes} : & \quad \vee \text{ nodeState}[k] \neq \text{"valid"} \\ & \quad \vee \text{ nodeState}[s] \neq \text{"valid"} \\ & \quad \vee \text{ nodeTS}[k] = \text{nodeTS}[s] \end{aligned}$$

$HInit \triangleq$ The initial predicate

$$\begin{aligned} \wedge \text{ msgs} & = \{\} \\ & \text{membership and epoch id related} \\ \wedge \text{ epochID} & = 0 \\ \wedge \text{ aliveNodes} & = H_NODES \\ \wedge \text{ nodeWriteEpochID} & = [n \in H_NODES \mapsto 0] \\ & \text{Init rest per node replica metadata} \\ \wedge \text{ nodeRcvdAcks} & = [n \in H_NODES \mapsto \{\}] \\ \wedge \text{ nodeState} & = [n \in H_NODES \mapsto \text{"valid"}] \\ \wedge \text{ nodeLastWriter} & = [n \in H_NODES \mapsto \text{CHOOSE } k \in H_NODES : \\ & \quad \forall m \in H_NODES : k \leq m] \\ \wedge \text{ nodeTS} & = [n \in H_NODES \mapsto [\text{version} \mapsto 0, \\ & \quad \text{tieBreaker} \mapsto \\ & \quad \text{CHOOSE } k \in H_NODES : \\ & \quad \quad \forall m \in H_NODES : k \leq m]] \\ \wedge \text{ nodeLastWriteTS} & = [n \in H_NODES \mapsto [\text{version} \mapsto 0, \\ & \quad \text{tieBreaker} \mapsto \\ & \quad \text{CHOOSE } k \in H_NODES : \\ & \quad \quad \forall m \in H_NODES : k \leq m]] \end{aligned}$$

A buffer maintaining all network messages. Messages are only appended to this variable (not removed once delivered) intentionally to check protocols tolerance in duplicates and reorderings
 $\text{send}(m) \triangleq \text{msgs}' = \text{msgs} \cup \{m\}$

Check if all acknowledgments for a write have been received
 $\text{receivedAllAcks}(n) \triangleq \text{nodeRcvdAcks}[n] = H_NODES \setminus \{n\}$
 $\text{receivedAllAcks}(n) \triangleq (\text{aliveNodes} \setminus \{n\}) \subseteq \text{nodeRcvdAcks}[n]$

$\text{equalTS}(v1, tb1, v2, tb2) \triangleq$ Timestamp equality
 $\wedge \quad v1 = v2$
 $\wedge \quad tb1 = tb2$

$\text{greaterTS}(v1, tb1, v2, tb2) \triangleq$ Timestamp comparison
 $\vee v1 > v2$
 $\vee \wedge \quad v1 = v2$

$$\wedge \text{tb1} > \text{tb2}$$

$$\text{isAlive}(n) \triangleq n \in \text{aliveNodes}$$

$$\begin{aligned} \text{nodeFailure}(n) &\triangleq \text{Emulate a node failure} \\ &\text{Make sure that there are atleast 3 alive nodes before killing a node} \\ &\wedge \text{Cardinality}(\text{aliveNodes}) > 2 \\ &\wedge \text{aliveNodes}' = \text{aliveNodes} \setminus \{n\} \\ &\wedge \text{epochID}' = \text{epochID} + 1 \\ &\wedge \text{UNCHANGED} \langle \text{msgs}, \text{nodeState}, \text{nodeTS}, \text{nodeLastWriter}, \\ &\quad \text{nodeLastWriteTS}, \text{nodeRcvdAcks}, \text{nodeWriteEpochID} \rangle \end{aligned}$$

$$\begin{aligned} \text{h_upd_not_aliveNodes} &\triangleq \\ &\wedge \text{UNCHANGED} \langle \text{aliveNodes}, \text{epochID}, \text{nodeWriteEpochID} \rangle \end{aligned}$$

$$\begin{aligned} \text{h_upd_aliveNodes} &\triangleq \\ &\wedge \text{UNCHANGED} \langle \text{msgs}, \text{nodeState}, \text{nodeTS}, \text{nodeLastWriter}, \text{nodeLastWriteTS}, \text{nodeRcvdAcks} \rangle \end{aligned}$$

$$\begin{aligned} \text{h_upd_nothing} &\triangleq \\ &\wedge \text{h_upd_not_aliveNodes} \\ &\wedge \text{h_upd_aliveNodes} \end{aligned}$$

$$\begin{aligned} \text{h_upd_state}(n, \text{newVersion}, \text{newTieBreaker}, \text{newState}, \text{newAcks}) &\triangleq \\ &\wedge \text{nodeLastWriter}' = [\text{nodeLastWriter} \text{ EXCEPT } ![n] = n] \\ &\wedge \text{nodeRcvdAcks}' = [\text{nodeRcvdAcks} \text{ EXCEPT } ![n] = \text{newAcks}] \\ &\wedge \text{nodeState}' = [\text{nodeState} \text{ EXCEPT } ![n] = \text{newState}] \\ &\wedge \text{nodeWriteEpochID}' = [\text{nodeWriteEpochID} \text{ EXCEPT } ![n] = \text{epochID}] \text{ we always use the latest epochID} \\ &\wedge \text{nodeTS}' = [\text{nodeTS} \text{ EXCEPT } ![n].\text{version} = \text{newVersion}, \\ &\quad \quad \quad ![n].\text{tieBreaker} = \text{newTieBreaker}] \\ &\wedge \text{nodeLastWriteTS}' = [\text{nodeLastWriteTS} \text{ EXCEPT } ![n].\text{version} = \text{newVersion}, \\ &\quad \quad \quad ![n].\text{tieBreaker} = \text{newTieBreaker}] \end{aligned}$$

$$\begin{aligned} \text{h_send_inv_or_ack}(n, \text{newVersion}, \text{newTieBreaker}, \text{msgType}) &\triangleq \\ &\wedge \text{send}([\text{type} \mapsto \text{msgType}, \\ &\quad \text{epochID} \mapsto \text{epochID}, \text{ we always use the latest epochID} \\ &\quad \text{sender} \mapsto n, \\ &\quad \text{version} \mapsto \text{newVersion}, \\ &\quad \text{tieBreaker} \mapsto \text{newTieBreaker}]) \end{aligned}$$

$$\begin{aligned} \text{h_actions_for_upd}(n, \text{newVersion}, \text{newTieBreaker}, \text{newState}, \text{newAcks}) &\triangleq \text{Execute a write} \\ &\wedge \text{h_upd_state}(n, \text{newVersion}, \text{newTieBreaker}, \text{newState}, \text{newAcks}) \\ &\wedge \text{h_send_inv_or_ack}(n, \text{newVersion}, \text{newTieBreaker}, \text{"INV"}) \\ &\wedge \text{UNCHANGED} \langle \text{aliveNodes}, \text{epochID} \rangle \end{aligned}$$

$$\text{h_actions_for_upd_replay}(n, \text{acks}) \triangleq \text{Apply a write-replay using same TS (version, tie-breaker)}$$

and either reset *acks* or keep already gathered *acks*

$$\wedge h_actions_for_upd(n, nodeTS[n].version, nodeTS[n].tieBreaker, "replay", acks)$$

$HRead(n) \triangleq$ Execute a read
 $\wedge nodeState[n] = \text{"valid"}$
 $\wedge h_upd_nothing$

$HWrite(n) \triangleq$ Execute a write
 $\wedge nodeState[n] \in \{\text{"valid"}, \text{"invalid"}\}$
 writes in invalid state are also supported as an optimization
 $\wedge nodeState[n] \in \{\text{"valid"}\}$
 $\wedge nodeTS[n].version < H_MAX_VERSION$ Only to configurably terminate the model checking
 $\wedge h_actions_for_upd(n, nodeTS[n].version + 1, n, \text{"write"}, \{\})$

$HCoordWriteReplay(n) \triangleq$ Execute a write-replay after a membership re-config
 $\wedge nodeState[n] \in \{\text{"write"}, \text{"replay"}\}$
 $\wedge nodeWriteEpochID[n] < epochID$
 $\wedge \neg receivedAllAcks(n)$ optimization to not replay when we have gathered *acks* from all alive
 $\wedge h_actions_for_upd_replay(n, nodeRcvdAcks[n])$

$HRcvAck(n) \triangleq$ Process a received acknowledgment
 $\exists m \in msgs :$
 $\wedge m.type = \text{"ACK"}$
 $\wedge m.epochID = epochID$
 $\wedge m.sender \neq n$
 $\wedge m.sender \notin nodeRcvdAcks[n]$
 $\wedge equalTS(m.version, m.tieBreaker,$
 $\quad nodeLastWriteTS[n].version,$
 $\quad nodeLastWriteTS[n].tieBreaker)$
 $\wedge nodeState[n] \in \{\text{"write"}, \text{"invalid_write"}, \text{"replay"}\}$
 $\wedge nodeRcvdAcks' = [nodeRcvdAcks \text{ EXCEPT } ![n] =$
 $\quad nodeRcvdAcks[n] \cup \{m.sender\}]$
 $\wedge \text{UNCHANGED } \langle msgs, nodeLastWriter, nodeLastWriteTS,$
 $\quad aliveNodes, nodeTS, nodeState, epochID, nodeWriteEpochID \rangle$

$HSendVals(n) \triangleq$ Send validations once acknowledgments are received from all alive nodes
 $\wedge nodeState[n] \in \{\text{"write"}, \text{"replay"}\}$
 $\wedge receivedAllAcks(n)$
 $\wedge nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{"valid"}]$
 $\wedge send([type \mapsto \text{"VAL"},$
 $\quad version \mapsto nodeTS[n].version,$
 $\quad tieBreaker \mapsto nodeTS[n].tieBreaker])$
 $\wedge \text{UNCHANGED } \langle nodeTS, nodeLastWriter, nodeLastWriteTS,$

$aliveNodes, nodeRcvdAcks, epochID, nodeWriteEpochID\rangle$

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

$HRcvInv(n) \triangleq$ Process a received invalidation
 $\exists m \in msgs :$
 $\wedge m.type = \text{"INV"}$
 $\wedge m.epochID = epochID$
 $\wedge m.sender \neq n$
 always acknowledge a received invalidation (irrelevant to the timestamp)
 $\wedge send([type \mapsto \text{"ACK"},$
 $sender \mapsto n,$
 $epochID \mapsto epochID,$
 $version \mapsto m.version,$
 $tieBreaker \mapsto m.tieBreaker])$
 $\wedge \text{IF } greaterTS(m.version, m.tieBreaker,$
 $nodeTS[n].version, nodeTS[n].tieBreaker)$
 THEN $\wedge nodeLastWriter' = [nodeLastWriter \text{ EXCEPT } ![n] = m.sender]$
 $\wedge nodeTS' = [nodeTS \text{ EXCEPT } ![n].version = m.version,$
 $![n].tieBreaker = m.tieBreaker]$
 $\wedge \text{IF } nodeState[n] \in \{\text{"valid"}, \text{"invalid"}, \text{"replay"}\}$
 THEN
 $nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{"invalid"}]$
 ELSE
 $nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{"invalid_write"}]$
 ELSE
 UNCHANGED $\langle nodeState, nodeTS, nodeLastWriter, nodeWriteEpochID \rangle$
 \wedge UNCHANGED $\langle nodeLastWriteTS, aliveNodes, nodeRcvdAcks, epochID, nodeWriteEpochID \rangle$

$HRcvVal(n) \triangleq$ Process a received validation
 $\exists m \in msgs :$
 $\wedge nodeState[n] \neq \text{"valid"}$
 $\wedge m.type = \text{"VAL"}$
 $\wedge equalTS(m.version, m.tieBreaker,$
 $nodeTS[n].version,$
 $nodeTS[n].tieBreaker)$
 $\wedge nodeState' = [nodeState \text{ EXCEPT } ![n] = \text{"valid"}]$
 \wedge UNCHANGED $\langle msgs, nodeTS, nodeLastWriter, nodeLastWriteTS,$

$aliveNodes, nodeRcvdAcks, epochID, nodeWriteEpochID\rangle$

$HFollowerWriteReplay(n) \triangleq$ Execute a write-replay when coordinator failed
 $\wedge nodeState[n] \in \{\text{"invalid"}, \text{"invalid_write"}\}$
 $\wedge \neg isAlive(nodeLastWriter[n])$
 $\wedge h_actions_for_upd_replay(n, \{\})$

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

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

$H_Spec \triangleq HInit \wedge \Box [HNext]_{hvars}$

THEOREM $H_Spec \Rightarrow (\Box HTypeOK) \wedge (\Box HConsistent)$
