MODULE *Galene*

Galene: is a linearizable protocol used in *ccKVS* of Scale-out *ccNUMA* [*Eurosys'18*]

EXTENDS *Integers*, *FiniteSets*

CONSTANTS *G_NODES*, *G_MAX_VERSION*

VARIABLES *msgs*, *nodeTS*, *nodeState*, *nodeRcvedAcks*, *nodeLastWriteTS*

all *Galene*(+ environment) variables

$gvars \triangleq \langle msgs,\ nodeTS,\ nodeState,\ nodeRcvedAcks,\ nodeLastWriteTS \rangle$

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

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

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

$\land\ v1 = v2$

$\land\ tb1 = tb2$

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

$\lor\ v1 > v2$

$\lor\ \land\ v1 = v2$

$\land\ tb1 > tb2$

$GMessage \triangleq$ Messages exchanged by the Protocol

$[type : \{\text{“INV”},\ \text{“ACK”}\},\ sender : G\_NODES,\ version : 0\ ..\ G\_MAX\_VERSION,\ tieBreaker : G\_NODES]$

$\cup$

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

$[type : \{\text{“VAL”}\},\ version : 0\ ..\ G\_MAX\_VERSION,\ tieBreaker : G\_NODES]$

$GTypeOK \triangleq$ The type correctness invariant

$\land\ msgs \subseteq GMessage$

$$
\begin{array}{l}
\quad \land \forall\, n \in G\_NODES : nodeRcvedAcks[n] \subseteq (G\_NODES \setminus \{n\}) \\
\quad \land \ nodeLastWriteTS \in [G\_NODES \to [version \quad : 0 \,..\, G\_MAX\_VERSION, \\
\qquad\qquad\qquad\qquad\qquad\qquad tieBreaker : G\_NODES \qquad]] \\
\quad \land \ nodeTS \qquad\quad \in [G\_NODES \to [version \quad : 0 \,..\, G\_MAX\_VERSION, \\
\qquad\qquad\qquad\qquad\qquad\qquad tieBreaker : G\_NODES \qquad]] \\
\quad \land \ nodeState \qquad \in [G\_NODES \to \{\text{“valid”}, \ \text{“invalid”}, \ \text{“invalid\_write”}, \ \text{“write”}\}]
\end{array}
$$

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

$$
\begin{array}{l}
GConsistent \triangleq \\
\quad \forall\, k,\, s \in G\_NODES \ : \ \lor\, nodeState[k] \neq \text{“valid”} \\
\qquad\qquad\qquad\qquad\quad \lor\, nodeState[s] \neq \text{“valid”} \\
\qquad\qquad\qquad\qquad\quad \lor\, nodeTS[k] = nodeTS[s]
\end{array}
$$

$$
\begin{array}{l}
GInit \triangleq \quad \text{The initial predicate} \\
\quad \land \ msgs \qquad\quad = \{\} \\
\quad \land \ nodeRcvedAcks \quad = [n \in G\_NODES \mapsto \{\}] \\
\quad \land \ nodeState \qquad = [n \in G\_NODES \mapsto \text{“valid”}] \\
\quad \land \ nodeTS \qquad\quad = [n \in G\_NODES \mapsto [version \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad tieBreaker \mapsto \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{CHOOSE } k \in G\_NODES : \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \forall\, m \in G\_NODES : k \leq m]] \\
\quad \land \ nodeLastWriteTS = [n \in G\_NODES \mapsto [version \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad tieBreaker \mapsto \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{CHOOSE } k \in G\_NODES : \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \forall\, m \in G\_NODES : k \leq m]]
\end{array}
$$

$$
\begin{array}{l}
g\_upd\_state(n,\, newVersion,\, newTieBreaker,\, newState,\, newAcks) \triangleq \\
\quad \land \ nodeRcvedAcks' \quad = [nodeRcvedAcks \quad \text{EXCEPT } ![n] = newAcks] \\
\quad \land \ nodeState' \qquad = [nodeState \qquad \text{EXCEPT } ![n] \quad = newState] \\
\quad \land \ nodeTS' \qquad\quad = [nodeTS \qquad\quad \text{EXCEPT } ![n].version \quad = newVersion, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![n].tieBreaker \ = newTieBreaker] \\
\quad \land \ nodeLastWriteTS' = [nodeLastWriteTS \ \text{EXCEPT } ![n].version \quad = newVersion, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![n].tieBreaker = newTieBreaker]
\end{array}
$$

$$
\begin{array}{l}
g\_send\_inv\_or\_ack(n,\, newVersion,\, newTieBreaker,\, msgType) \triangleq \\
\quad \land \ send([type \qquad \mapsto msgType, \\
\qquad\qquad sender \quad\ \mapsto n, \\
\qquad\qquad version \quad \mapsto newVersion, \\
\qquad\qquad tieBreaker \ \mapsto newTieBreaker])
\end{array}
$$

$$
\begin{array}{l}
g\_actions\_for\_upd(n,\, newVersion,\, newTieBreaker,\, newState,\, newAcks) \triangleq \\
\quad \land \ g\_upd\_state(n,\, newVersion,\, newTieBreaker,\, newState,\, newAcks) \\
\quad \land \ g\_send\_inv\_or\_ack(n,\, newVersion,\, newTieBreaker,\, \text{“INV”})
\end{array}
$$

$GRead(n) \triangleq$ Execute a read
$\land\ nodeState[n] = \text{“valid”}$
$\land$ UNCHANGED $\langle msgs,\ nodeTS,\ nodeState,\ nodeRcvedAcks,\ nodeLastWriteTS \rangle$

$GWrite(n) \triangleq$ Execute a write
writes in invalid state are also supported as an optimization
$\land\ nodeState[n] \in \{\text{“valid”}\}$
$\land\ nodeTS[n].version < G\_MAX\_VERSION$ Only to configurably terminate the model checking
$\land\ g\_actions\_for\_upd(n,\ nodeTS[n].version + 1,\ n,\ \text{“write”},\ \{\})$

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

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

$GCoordinatorActions(n) \triangleq$ Coordinator actions for reads or writes
$\lor\ GRead(n)$
$\lor\ GWrite(n)$
$\lor\ GRcvAck(n)$
$\lor\ GSendVals(n)$

$GRcvInv(n) \triangleq$ Process received invalidation
$\exists\, m \in msgs :$
$\land\ m.type = \text{“INV”}$
$\land\ m.sender \neq n$

```
        always acknowledge a received invalidation (irrelevant to the timestamp)
    ∧ send([type        ↦ "ACK",
            sender      ↦ n,
            version     ↦ m.version,
            tieBreaker ↦ m.tieBreaker])
    ∧ IF greaterTS(m.version, m.tieBreaker,
                   nodeTS[n].version, nodeTS[n].tieBreaker)
      THEN  ∧ nodeTS' = [nodeTS EXCEPT ![n].version    = m.version,
                                       ![n].tieBreaker = m.tieBreaker]
            ∧ IF nodeState[n] ∈ {"valid", "invalid"}
              THEN
                nodeState' = [nodeState EXCEPT ![n] = "invalid"]
              ELSE
                nodeState' = [nodeState EXCEPT ![n] = "invalid_write"]
      ELSE
            UNCHANGED ⟨nodeState, nodeTS⟩
    ∧ UNCHANGED ⟨nodeLastWriteTS, nodeRcvedAcks⟩

GRcvVal(n) ≜    Process received validation
  ∃ m ∈ msgs :
    ∧ nodeState[n] ≠ "valid"
    ∧ m.type = "VAL"
    ∧ equalTS(m.version, m.tieBreaker,
              nodeTS[n].version,
              nodeTS[n].tieBreaker)
    ∧ nodeState' = [nodeState EXCEPT ![n] = "valid"]
    ∧ UNCHANGED ⟨msgs, nodeTS, nodeLastWriteTS, nodeRcvedAcks⟩

GFollowerActions(n) ≜    Follower actions for writes
  ∨ GRcvInv(n)
  ∨ GRcvVal(n)

---

GNext ≜    Coordinator and Follower actions
  ∃ n ∈ G_NODES :
      ∨ GFollowerActions(n)
      ∨ GCoordinatorActions(n)

G_Spec ≜ GInit ∧ □[GNext]_gvars

THEOREM G_Spec ⇒ (□GTypeOK) ∧ (□GConsistent)
```