MODULE *OneVotePaxosStore*

Specification of the consensus protocol in *PaxosStore*.

See [*PaxosStore*@*VLDB*2017](*https* : *//www.vldb.org/pvldb/vol*10/*p*1730 − *lin.pdf*) by *Tencent*.

In this version (adopted from "*UniversalPaxosStore.tla*"):

- Use *OneVote* and *IntersectingQuorum* together to replace the Client-restricted config for *Ballot* allocation; that is, no $Bals(p)$ in this version.

- Still no message types or state flags.

EXTENDS *Integers*, *FiniteSets*

$Max(m, n) \triangleq$ IF $m > n$ THEN m ELSE n

CONSTANTS

Participant, the set of partipants

Value the set of possible input values for *Participant* to propose

$None \triangleq$ CHOOSE $b : b \notin Value$

$Quorum \triangleq \{Q \in$ SUBSET $Participant :$
$\quad Cardinality(Q) * 2 = Cardinality(Participant) + 1\}$

ASSUME $QuorumAssumption \triangleq$

$\quad \wedge \quad \forall\, Q \in Quorum : Q \subseteq Participant$

$\quad \wedge \quad \forall\, Q1,\, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

$Ballot \triangleq Nat$

$State \triangleq [maxBal \ : Ballot \cup \{-1\},$
$\quad maxVBal : Ballot \cup \{-1\},\ maxVVal : Value \cup \{None\}]$

$InitState \triangleq [maxBal \mapsto -1,\ maxVBal \mapsto -1,\ maxVVal \mapsto None]$

For simplicity, in this specification, we choose to send the complete state of a participant each time. When receiving such a message, the participant processes only the "partial" state it needs.

$Message \triangleq [from : Participant,\ to :$ SUBSET $Participant,\ state : [Participant \to State]]$

VARIABLES

state, *state*[*p*][*q*]: the state of $q \in$ *Participant* from the view of $p \in$ *Participant*

msgs the set of messages that have been sent

$vars \triangleq \langle state,\ msgs \rangle$

$TypeOK \triangleq$

$\quad \wedge \quad state \in [Participant \to [Participant \to State]]$

$\quad \wedge \quad msgs \subseteq Message$

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

$Init \triangleq$

```
58      ∧ state = [p ∈ Participant ↦ [q ∈ Participant ↦ InitState]]
59      ∧ msgs = {}
        p ∈ Participant starts the prepare phase by issuing a ballot b ∈ Ballot.
63  Prepare(p, b) ≜
64      ∧  state[p][p].maxBal < b
65      ∧  state' = [state EXCEPT ![p][p].maxBal = b]
66      ∧  Send([from ↦ p, to ↦ Participant, state ↦ state'[p]])
        q ∈ Participant updates its own state state[q] according to the actual state pp of p ∈ Participant
        extracted from a message m ∈ Message it receives. This is called by OnMessage(q).

        Note: pp is m.state[p]; it may not be equal to state[p][p] at the time UpdateState is called.
75  UpdateState(q, p, pp) ≜
76      state' = [state EXCEPT
77                  ![q][p].maxBal = Max(@, pp.maxBal),
78                  ![q][p].maxVBal = Max(@, pp.maxVBal),
79                  ![q][p].maxVVal = IF state[q][p].maxVBal < pp.maxVBal
80                                    THEN pp.maxVVal ELSE @,
81                  ![q][q].maxBal = Max(@, pp.maxBal),
82                  ![q][q].maxVBal = IF state[q][q].maxBal ≤ pp.maxVBal
83                                    THEN pp.maxVBal ELSE @,    make promise
84                  ![q][q].maxVVal = IF ∨ state[q][q].maxBal < pp.maxVBal
85                                       OneVote
86                                     ∨ state[q][q].maxBal = pp.maxVBal ∧ @ = None
87                                    THEN pp.maxVVal ELSE @]   accept
        q ∈ Participant receives and processes a message in Message.
91  OnMessage(q) ≜
92      ∃ m ∈ msgs :
93        ∧ q ∈ m.to
94        ∧ LET p ≜ m.from
95          IN  UpdateState(q, p, m.state[p])
96        ∧ IF ∨ m.state[q].maxBal < state'[q][q].maxBal
97             ∨ m.state[q].maxVBal < state'[q][q].maxVBal
98          THEN Send([from ↦ q, to ↦ {m.from}, state ↦ state'[q]])
99          ELSE UNCHANGED msgs
        p ∈ Participant starts the accept phase by issuing the ballot b ∈ Ballot with value v ∈ Value.
104 Accept(p, b, v) ≜
105     ∧ state[p][p].maxVBal ≠ b  for OneVote; TODO: too strong?
106     (i.e., state[p][p].maxVBal = b ⇒ v = state[p][p].maxVVal)
107     (it ensures ∀ p ∈ Participant, b ∈ Ballot : Accept(p, b, _) only once)
108     ∧ ∃ Q ∈ Quorum : ∀ q ∈ Q : state[p][q].maxBal = b
109     ∧ ∨ ∀ q ∈ Participant : state[p][q].maxVBal = −1  free to pick its own value
110       ∨ ∃ q ∈ Participant :   v is the value with the highest maxVBal
111           ∧ state[p][q].maxVVal = v
112           ∧ ∀ r ∈ Participant : state[p][q].maxVBal ≥ state[p][r].maxVBal
```

```
113       ∧ state' = [state EXCEPT ![p][p].maxVBal = b, ![p][p].maxVVal = v]
114       ∧ Send([from ↦ p, to ↦ Participant, state ↦ state'[p]])
115 ----------------------------------------------------------------------
116   Next ≜ ∃ p ∈ Participant : ∨ OnMessage(p)
117                              ∨ ∃ b ∈ Ballot : ∨ Prepare(p, b)
118                                               ∨ ∃ v ∈ Value : Accept(p, b, v)
119   Spec ≜ Init ∧ □[Next]_vars
120 ----------------------------------------------------------------------
121   ChosenP(p) ≜  the set of values chosen by p ∈ Participant
122       {v ∈ Value : ∃ b ∈ Ballot :
123                      ∃ Q ∈ Quorum : ∀ q ∈ Q : ∧ state[p][q].maxVBal = b
124                                               ∧ state[p][q].maxVVal = v}
125   chosen ≜ UNION {ChosenP(p) : p ∈ Participant}

127   Consistency ≜ Cardinality(chosen) ≤ 1
128   THEOREM Spec ⇒ □Consistency
129 ======================================================================
```

\ * Modification History
\ * Last modified *Wed Jul* 31 16:36:13 *CST* 2019 by *hengxin*
\ * Last modified *Mon Jul* 22 13:59:15 *CST* 2019 by *pure_*
\ * Last modified *Mon Jun* 03 21:26:09 *CST* 2019 by *stary*
\ * Last modified *Wed* May 09 21:39:31 *CST* 2018 by dell
\ * Created *Mon Apr* 23 15:47:52 *GMT* + 08:00 2018 by *pure_*