MODULE *TCS*

The specification of the Transaction Certification Service (*TCS*) in *DISC′*2018 "Multi-Shot Distributed Transaction Commit" by *Gregory Chockler* and *Alexey Gotsman*.

We have specified the multi-shot $2PC$ protocol in Figure 1 of *DISC′*2018.

TODO: We plan
- to test *SER* using the Serializability Theorem
- to integrate *TCS* into a real distributed transaction protocol
- to implement certification functions for other isolation levels
- to specify the fault-tolerant commit protocol in Figure 5 of *DISC′*2018.

EXTENDS *Naturals*, *Integers*, *FiniteSets*, *Sequences*, *Functions*, *TLC*, *FiniteSetsExt*

CONSTANTS

Key, the set of keys, ranged over by $k \in Key$

Tid, the set of transaction identifiers, ranged over by $t \in Tid$

RSet, $RSet[t]$: the read set of $t \in Tid$

WSet, $WSet[t]$: the write set of $t \in Tid$

CVer, $CVer[t]$: the commit version of $t \in Tid$

Shard, the set of *shards*, ranged over by $s \in Shard$

Coord, $Coord[t]$: the coordinator of $t \in Tid$

KeySharding $KeySharding[k]$: the shard that holds $k \in Key$

$$NotTid \triangleq \text{CHOOSE } t : t \notin Tid$$

$$Ver \triangleq 0 \,..\, Cardinality(Tid)$$ with a distinguished minimum version 0

$$Slot \triangleq 0 \,..\, Cardinality(Tid) - 1$$

$$TShard(t) \triangleq \{KeySharding[k] : k \in (WSet[t] \cup \{kv[1] : kv \in RSet[t]\})\}$$

ASSUME *TODO*: See Section 2 of *DISC′*2018

$$\land \quad RSet \in [Tid \to \text{SUBSET } (Key \times Ver)]$$
$$\land \forall\, t \in Tid: RSet[t] \setminus^* TODO: \text{ one version per object}$$
$$\land\ WSet \in [Tid \to \text{SUBSET } Key]$$
$$\land \setminus^* TODO: \text{ "no blind update" assumption}$$
$$\land\ CVer \in [Tid \to Ver]$$
$$\land \setminus^* TODO: \text{ higher than any of the versions read}$$
$$\land\ Coord \in [Tid \to Shard]$$
$$\land\ KeySharding \in [Key \to Shard]$$

VARIABLES

next, $next[s] \in Z$ points to the last filled slot

txn, $txn[s][i]$ is the transaction (identifier) to certify in the i-th slot

vote, $vote[s][i]$ is the vote for $txn[s][i]$

dec, $dec[s][i]$ is the decision for $txn[s][i]$

phase, $phase[s][i]$ is the phase for $txn[s][i]$

msg, the set of messages in transit

```
52      submitted      the set of t ∈ Tid that have been submitted to TCS

54  sVars ≜ ⟨next, txn, vote, dec, phase⟩
55  vars ≜ ⟨next, txn, vote, dec, phase, msg, submitted⟩
56 ├──────────────────────────────────────────────────────────────────────┤
     TODO:
    "COMMIT/ABORT": using TRUE/FALSE (initially, FALSE???)
    "PREPARE/PREPARE_ACK/DECISION": using CONSTANTS
61  Message ≜ [type : {"PREPARE"}, t : Tid, s : Shard]
62      ∪ [type : {"PREPARE_ACK"}, s : Shard, n : Int, t : Tid, v : {"COMMIT", "ABORT"}]
63      ∪ [type : {"DECISION"}, p : Int, d : {"COMMIT", "ABORT"}, s : Shard]

65  Send(m) ≜ msg′ = msg ∪ m
66  Delete(m) ≜ msg′ = msg \ m
67  SendAndDelete(sm, dm) ≜ msg′ = (msg ∪ sm) \ dm
68 ├──────────────────────────────────────────────────────────────────────┤
69  TypeOK ≜
70      ∧   next ∈ [Shard → Int]
71      ∧   txn ∈ [Shard → [Slot → Tid ∪ {NotTid}]]
72      ∧   vote ∈ [Shard → [Slot → {"COMMIT", "ABORT", "NULL"}]]
73      ∧   dec ∈ [Shard → [Slot → {"COMMIT", "ABORT", "NULL"}]]
74      ∧   phase ∈ [Shard → [Slot → {"START", "PREPARED", "DECIDED"}]]
75      ∧   msg ⊆ Message
76      ∧   submitted ⊆ Tid
77 ├──────────────────────────────────────────────────────────────────────┤
78  Init ≜
79      ∧ next = [s ∈ Shard ↦ −1]
80      ∧ txn = [s ∈ Shard ↦ [i ∈ Slot ↦ NotTid]]
81      ∧ vote = [s ∈ Shard ↦ [i ∈ Slot ↦ "NULL"]]
82      ∧ dec = [s ∈ Shard ↦ [i ∈ Slot ↦ "NULL"]]
83      ∧ phase = [s ∈ Shard ↦ [i ∈ Slot ↦ "START"]]
84      ∧ msg = {}
85      ∧ submitted = {}
86 ├──────────────────────────────────────────────────────────────────────┤
87  KeyOnShard(s) ≜ {k ∈ Key : KeySharding[k] = s}

89  ComputeVote(t, s, n) ≜
90      LET cs ≜ {k ∈ Slot :   committed slots before position n
91                  ∧ k < n
92                  ∧ phase[s][k] = "DECIDED"
93                  ∧ dec[s][k] = "COMMIT"}
94          ct ≜ {txn[s][k] : k ∈ cs}   committed transactions
95          fv ≜ IF ∀ k ∈ KeyOnShard(s), v ∈ Ver :
96                  ⟨k, v⟩ ∈ RSet[t] ⇒ (∀ c ∈ ct : k ∈ WSet[c] ⇒ CVer[c] ≤ v)
97             THEN "COMMIT" ELSE "ABORT"
98          ps ≜ {k ∈ Slot :   "prepared to commit" slots before position n
```

```
                    ∧ k < n
                    ∧ phase[s][k] = "PREPARED"
                    ∧ vote[s][k] = "COMMIT"}
          pt  ≜ {txn[s][k] : k ∈ ps}  "prepared to commit" transactions
          gv  ≜ IF ∀ k ∈ KeyOnShard(s), v ∈ Ver :
                    ∧ ⟨k, v⟩ ∈ RSet[t] ⇒ (∀ p ∈ pt : k ∉ WSet[p])
                    ∧ k ∈ WSet[t] ⇒ (∀ p ∈ pt : ⟨k, v⟩ ∉ RSet[p])
                 THEN "COMMIT" ELSE "ABORT"
      IN  IF fv = "COMMIT" ∧ gv = "COMMIT" THEN "COMMIT" ELSE "ABORT"

ComputeDecision(vs) ≜
    IF ∀ v ∈ vs : v = "COMMIT" THEN "COMMIT" ELSE "ABORT"
├──────────────────────────────────────────────────────────────────────────────┤
Certify(t) ≜  Certify t ∈ Tid
    ∧ t ∈ Tid \ submitted
    ∧ Send([type : {"PREPARE"}, t : {t}, s : TShard(t)])
    ∧ submitted' = submitted ∪ {t}
    ∧ UNCHANGED sVars

Prepare(t, s) ≜  Prepare t ∈ Tid on s ∈ Shard when receive "PREPARE(t)" message
    ∧  ∃ m ∈ msg :
         ∧ m = [type ↦ "PREPARE", t ↦ t, s ↦ s]
         ∧ next' = [next EXCEPT ![s] = @ + 1]
         ∧ txn' = [txn EXCEPT ![s][next'[s]] = t]
         ∧ vote' = [vote EXCEPT ![s][next'[s]] = ComputeVote(t, s, next'[s])]
         ∧ phase' = [phase EXCEPT ![s][next'[s]] = "PREPARED"]
         ∧ SendAndDelete({[type ↦ "PREPARE_ACK",
                           s ↦ s,
                           n ↦ next'[s],
                           t ↦ t,
                           v ↦ vote'[s][next'[s]]]},
                         {m})
    ∧  UNCHANGED ⟨dec, submitted⟩

PrepareAck(t, s) ≜  PrepareAck for t ∈ Tid on shard s ∈ Shard when receive all "PREPARE_ACK" messages for t
    ∧ s = Coord[t]
    ∧ LET ms ≜ {m ∈ msg : m.type = "PREPARE_ACK" ∧ m.t = t}
          vs ≜ {m.v : m ∈ ms}
          ss ≜ {m.s : m ∈ ms}
      IN   ∧ ss = TShard(t)
           ∧ SendAndDelete({[type ↦ "DECISION",
                             p ↦ ChooseUnique(ms, LAMBDA m : m.s = shard).n,
                             d ↦ ComputeDecision(vs),
                             s ↦ shard] : shard ∈ ss},
                           ms)
    ∧ UNCHANGED ⟨sVars, submitted⟩
```

```
146 Decision(s) ≜  Decide on shard s ∈ Shard when receive a "DECISION" message
147     ∧ ∃ m ∈ msg :
148         ∧ m.type = "DECISION"
149         ∧ m.s = s
150         ∧ dec' = [dec EXCEPT ![s][m.p] = m.d]
151         ∧ phase' = [phase EXCEPT ![s][m.p] = "DECIDED"]
152         ∧ Delete({m})
153     ∧ UNCHANGED ⟨next, txn, vote, submitted⟩
154 ----------------------------------------------------------------------------
    TODO: adding the two non-deterministic actions
158 ----------------------------------------------------------------------------
159 Next ≜
160     ∨ ∃ t ∈ Tid : Certify(t)
161     ∨ ∃ t ∈ Tid, s ∈ Shard :
162         ∨ Prepare(t, s)
163         ∨ PrepareAck(t, s)
164     ∨ ∃ s ∈ Shard :
165         ∨ Decision(s)

167 Spec ≜ Init ∧ □[Next]_vars
168 ============================================================================
    \* Modification History
    \* Last modified Sun Jun 13 19:27:55 CST 2021 by hengxin
    \* Created Sat Jun 12 21:01:57 CST 2021 by hengxin
```