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:
   "PREPARE/PREPARE_ACK/DECISION": using CONSTANTS
60  Message ≜ [type : {"PREPARE"}, t : Tid, s : Shard]
61      ∪ [type : {"PREPARE_ACK"}, s : Shard, n : Int, t : Tid, v : BOOLEAN ]
62      ∪ [type : {"DECISION"}, p : Int, d : BOOLEAN , s : Shard]

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

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

```
                    ∧ vote[s][k]}
        pt  ≜ {txn[s][k] : k ∈ ps}  "prepared to commit" transactions
        gv  ≜ ∀ k ∈ KeyOnShard(s), v ∈ Ver :
                    ∧ ⟨k, v⟩ ∈ RSet[t] ⇒ (∀ p ∈ pt : k ∉ WSet[p])
                    ∧ k ∈ WSet[t] ⇒ (∀ p ∈ pt : ⟨k, v⟩ ∉ RSet[p])
    IN  fv ∧ gv

ComputeDecision(vs) ≜ ∀ v ∈ vs : v
```

```
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⟩

Decision(s) ≜  Decide on shard s ∈ Shard when receive a "DECISION" message
    ∧ ∃ m ∈ msg :
        ∧ m.type = "DECISION"
```

```
145             ∧ m.s = s
146             ∧ dec' = [dec EXCEPT ![s][m.p] = m.d]
147             ∧ phase' = [phase EXCEPT ![s][m.p] = "DECIDED"]
148             ∧ Delete({m})
149         ∧ UNCHANGED ⟨next, txn, vote, submitted⟩
150 ----------------------------------------------------------------
    TODO: adding the two non-deterministic actions
154 ----------------------------------------------------------------
155 Next ≜
156     ∨ ∃ t ∈ Tid : Certify(t)
157     ∨ ∃ t ∈ Tid, s ∈ Shard :
158         ∨ Prepare(t, s)
159         ∨ PrepareAck(t, s)
160     ∨ ∃ s ∈ Shard :
161         ∨ Decision(s)

163 Spec ≜ Init ∧ □[Next]_vars
164 ================================================================
    \* Modification History
    \* Last modified Sun Jun 13 19:39:45 CST 2021 by hengxin
    \* Created Sat Jun 12 21:01:57 CST 2021 by hengxin
```