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: 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 $k \in Key$

$NotTid \triangleq$ 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*: one version per object

$\land\ WSet \in [Tid \to \text{SUBSET } Key]$

$\land \setminus *$ *TODO*: "no blind update" assumption

$\land\ CVer \in [Tid \to Ver]$

$\land \setminus *$ *TODO*: 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
- $submitted$ the set of $t \in Tid$ that have been submitted to *TCS*

$sVars \triangleq \langle next, txn, vote, dec, phase \rangle$

$vars \triangleq \langle next, txn, vote, dec, phase, msg, submitted \rangle$

```
52 ├──────────────────────────────────────────────────────────────────────┤
   TODO:
   "COMMIT/ABORT": using TRUE/FALSE (initially, FALSE???)
   "PREPARE/PREPARE_ACK/DECISION": using CONSTANTS
57 Message ≜ [type : {"PREPARE"}, t : Tid, s : Shard]
58     ∪ [type : {"PREPARE_ACK"}, s : Shard, n : Int, t : Tid, v : {"COMMIT", "ABORT"}]
59     ∪ [type : {"DECISION"}, p : Int, d : {"COMMIT", "ABORT"}, s : Shard]

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

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

```
98          pt  ≜ {txn[s][k] : k ∈ ps}  "prepared to commit" transactions
99          gv  ≜ IF ∀ k ∈ KeyOnShard(s), v ∈ Ver :
100                    ∧ ⟨k, v⟩ ∈ RSet[t] ⇒ (∀ p ∈ pt : k ∉ WSet[p])
101                    ∧ k ∈ WSet[t] ⇒ (∀ p ∈ pt : ⟨k, v⟩ ∉ RSet[p])
102                THEN "COMMIT" ELSE "ABORT"
103     IN  IF fv = "COMMIT" ∧ gv = "COMMIT" THEN "COMMIT" ELSE "ABORT"

105 ComputeDecision(vs) ≜
106     IF ∀ v ∈ vs : v = "COMMIT" THEN "COMMIT" ELSE "ABORT"
107 ----------------------------------------------------------------------
108 Certify(t) ≜  Certify t ∈ Tid
109     ∧ t ∈ Tid \ submitted
110     ∧ Send([type : {"PREPARE"}, t : {t}, s : TShard(t)])
111     ∧ submitted' = submitted ∪ {t}
112     ∧ UNCHANGED sVars

114 Prepare(t, s) ≜  Prepare t ∈ Tid on s ∈ Shard when receive "PREPARE(t)" message
115     ∧ ∃ m ∈ msg :
116         ∧ m = [type ↦ "PREPARE", t ↦ t, s ↦ s]
117         ∧ next' = [next EXCEPT ![s] = @ + 1]
118         ∧ txn' = [txn EXCEPT ![s][next'[s]] = t]
119         ∧ vote' = [vote EXCEPT ![s][next'[s]] = ComputeVote(t, s, next'[s])]
120         ∧ phase' = [phase EXCEPT ![s][next'[s]] = "PREPARED"]
121         ∧ SendAndDelete({[type ↦ "PREPARE_ACK",
122                           s ↦ s,
123                           n ↦ next'[s],
124                           t ↦ t,
125                           v ↦ vote'[s][next'[s]]]},
126                         {m})
127     ∧ UNCHANGED ⟨dec, submitted⟩

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

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

```
144             ∧ 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 19:22:37 CST 2021 by hengxin
    \* Created Sat Jun 12 21:01:57 CST 2021 by hengxin
```