```
1  ---------- MODULE TCS ----------
   See DISC'2018: Multi-Shot Distributed Transaction Commit
5  EXTENDS Naturals, Integers, FiniteSets, Sequences, Functions, TLC,
6          FiniteSetsExt
7  ------------------------------------
8  CONSTANTS
9      Key,        the set of keys, ranged over by k ∈ Key
10     Tid,        the set of transaction identifiers, ranged over by t ∈ Tid
11     RSet,       RSet[t]: the read set of t ∈ Tid
12     WSet,       WSet[t]: the write set of t ∈ Tid
13     CVer,       CVer[t]: the commit version of t ∈ Tid
14     Shard,      the set of shards, ranged over by s ∈ Shard
15     Coord,      Coord[t]: the coordinator of t ∈ Tid
16     KeySharding KeySharding[k]: the shard that holds k ∈ Key

18 NotTid ≜ CHOOSE t : t ∉ Tid

20 Ver ≜ 0 .. Cardinality(Tid)  with a distinguished minimum version 0
21 Slot ≜ 0 .. Cardinality(Tid) − 1

23 TKey(t) ≜ WSet[t] ∪ {kv[1] : kv ∈ RSet[t]}
24 TSharding(t) ≜ {KeySharding[k] : k ∈ TKey(t)}

26 ASSUME
27     ∧ RSet ∈ [Tid → SUBSET (Key × Ver)]
28     ∧ ∀ t ∈ Tid: RSet[t] \* TODO: one version per object
29     ∧ WSet ∈ [Tid → SUBSET Key]
30     ∧ \* TODO: "no blind update" assumption
31     ∧ CVer ∈ [Tid → Ver]
32     ∧ \* TODO: higher than any of the versions read
33     ∧ Coord ∈ [Tid → Shard]
34     ∧ KeySharding ∈ [Key → Shard]
35 ------------------------------------
36 VARIABLES
37     next,       next[s] ∈ Z points to the last filled slot
38     txn,        txn[s][i] is the transaction (identifier) to certify in the i-th slot
39     vote,       vote[s][i] is the vote for txn[s][i]
40     dec,        dec[s][i] is the decision for txn[s][i]
41     phase,      phase[s][i] is the phase for txn[s][i]
42     msg,        the set of messages in transit
43     submitted   the set of t ∈ Tid that have been submitted to TCS

45 sVars ≜ ⟨next, txn, vote, dec, phase⟩
46 vars ≜ ⟨next, txn, vote, dec, phase, msg, submitted⟩
47 ------------------------------------
48 Message ≜ [type : {"PREPARE"}, t : Tid, s : Shard]
```

```
49      ∪ [type : {"PREPARE_ACK"}, s : Shard, n : Int, t : Tid, v : {"COMMIT", "ABORT"}]
50      ∪ [type : {"DECISION"}, p : Int, d : {"COMMIT", "ABORT"}, s : Shard]

52  Send(m) ≜ msg' = msg ∪ m
53  Delete(m) ≜ msg' = msg \ m
54  SendAndDelete(sm, dm) ≜ msg' = (msg ∪ sm) \ dm
55 ----------------------------------------------------------------------
56  TypeOK ≜
57      ∧   next ∈ [Shard → Int]
58      ∧   txn ∈ [Shard → [Slot → Tid ∪ {NotTid}]]
59      ∧   vote ∈ [Shard → [Slot → {"COMMIT", "ABORT", "NULL"}]]
60      ∧   dec ∈ [Shard → [Slot → {"COMMIT", "ABORT", "NULL"}]]
61      ∧   phase ∈ [Shard → [Slot → {"START", "PREPARED", "DECIDED"}]]
62      ∧   msg ⊆ Message
63      ∧   submitted ⊆ Tid
64 ----------------------------------------------------------------------
65  Init ≜
66      ∧ next = [s ∈ Shard ↦ −1]
67      ∧ txn = [s ∈ Shard ↦ [i ∈ Slot ↦ NotTid]]
68      ∧ vote = [s ∈ Shard ↦ [i ∈ Slot ↦ "NULL"]]
69      ∧ dec = [s ∈ Shard ↦ [i ∈ Slot ↦ "NULL"]]
70      ∧ phase = [s ∈ Shard ↦ [i ∈ Slot ↦ "START"]]
71      ∧ msg = {}
72      ∧ submitted = {}
73 ----------------------------------------------------------------------
74  ComputeVote(t, s, n) ≜ "ABORT"  TODO
75  ComputeDecision(vs) ≜
76      IF ∃ v ∈ vs : v = "ABORT" THEN "ABORT" ELSE "COMMIT"
77 ----------------------------------------------------------------------
78  Certify(t) ≜  Certify t ∈ Tid
79      ∧  t ∈ Tid \ submitted
80      ∧  Send([type : {"PREPARE"}, t : {t}, s : TSharding(t)])
81      ∧  submitted' = submitted ∪ {t}
82      ∧  UNCHANGED sVars

84  Prepare(t, s) ≜  Prepare t ∈ Tid on s ∈ Shard when receive "PREPARE(t)" message
85      ∧  ∃ m ∈ msg :
86          ∧ m = [type ↦ "PREPARE", t ↦ t, s ↦ s]
87          ∧ next' = [next EXCEPT ![s] = @ + 1]
88          ∧ txn' = [txn EXCEPT ![s][next'[s]] = t]
89          ∧ vote' = [vote EXCEPT ![s][next'[s]] = ComputeVote(t, s, next'[s])]  TODO
90          ∧ phase' = [phase EXCEPT ![s][next'[s]] = "PREPARED"]
91          ∧ SendAndDelete({[type ↦ "PREPARE_ACK",
92                            s ↦ s,
93                            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 = TSharding(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"
        ∧ m.s = s
        ∧ dec' = [dec EXCEPT ![s][m.p] = m.d]
        ∧ phase' = [phase EXCEPT ![s][m.p] = "DECIDED"]
        ∧ Delete({m})
    ∧ UNCHANGED ⟨next, txn, vote, submitted⟩
```

```
Next ≜
    ∨ ∃ t ∈ Tid : Certify(t)
    ∨ ∃ t ∈ Tid, s ∈ Shard :
        ∨ Prepare(t, s)
        ∨ PrepareAck(t, s)
    ∨ ∃ s ∈ Shard :
        ∨ Decision(s)

Spec ≜ Init ∧ □[Next]_vars
```

```
\* Modification History
\* Last modified Sun Jun 13 17:13:35 CST 2021 by hengxin
\* Created Sat Jun 12 21:01:57 CST 2021 by hengxin
```