MODULE *TCS*

See *DISC'*2018: Multi-Shot Distributed Transaction Commit

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

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

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

$TSharding(t) \triangleq \{KeySharding[k] : k \in TKey(t)\}$

ASSUME

$\land\ RSet \in [Tid \to \text{SUBSET } (Key \times Ver)]$

$\land\ \forall\, t \in Tid$: $RSet[t]$ * *TODO*: one version per object

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

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

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

$\land$ * *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*

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

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

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

Send(m) ≜ msg' = msg ∪ m
Delete(m) ≜ msg' = msg \ m
SendAndDelete(sm, dm) ≜ msg' = (msg ∪ sm) \ dm
----------------------------------------------------------------------------
TypeOK ≜
    ∧ next ∈ [Shard → Int]
    ∧ txn ∈ [Shard → [Slot → Tid ∪ {NotTid}]]
    ∧ vote ∈ [Shard → [Slot → {"COMMIT", "ABORT", "NULL"}]]
    ∧ dec ∈ [Shard → [Slot → {"COMMIT", "ABORT", "NULL"}]]
    ∧ phase ∈ [Shard → [Slot → {"START", "PREPARED", "DECIDED"}]]
    ∧ msg ⊆ Message
    ∧ submitted ⊆ Tid
----------------------------------------------------------------------------
Init ≜
    ∧ next = [s ∈ Shard ↦ −1]
    ∧ txn = [s ∈ Shard ↦ [i ∈ Slot ↦ NotTid]]
    ∧ vote = [s ∈ Shard ↦ [i ∈ Slot ↦ "NULL"]]
    ∧ dec = [s ∈ Shard ↦ [i ∈ Slot ↦ "NULL"]]
    ∧ phase = [s ∈ Shard ↦ [i ∈ Slot ↦ "START"]]
    ∧ msg = {}
    ∧ submitted = {}
----------------------------------------------------------------------------
KeyOnShard(s) ≜ {k ∈ Key : KeySharding[k] = s}

ComputeVote(t, s, n) ≜
  LET cs ≜ {k ∈ Slot :  committed slots before position n
                ∧ k < n
                ∧ phase[s][k] = "DECIDED"
                ∧ dec[s][k] = "COMMIT"}
      ct ≜ {txn[s][k] : k ∈ cs}  committed transactions
      fv ≜ IF ∀ k ∈ KeyOnShard(s), v ∈ Ver :
                ⟨k, v⟩ ∈ RSet[t] ⇒ (∀ c ∈ ct : k ∈ WSet[c] ⇒ CVer[c] ≤ v)
           THEN "COMMIT" ELSE "ABORT"
      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
```

```
94          gv ≜ "ABORT"
95      gv ≜ IF ∀ k ∈ KeyOnShard(s), v ∈ Ver:
96              ∧ ⟨k, v⟩ ∈ RSet[t] ⇒ (∀ p ∈ pt : k ∉ WSet[p])
97              ∧ k ∈ WSet[t] ⇒ (∀ p ∈ pt : ⟨k, v⟩ ∉ RSet[p])
98           THEN "COMMIT" ELSE "ABORT"
99      IN  IF fv = "COMMIT" ∧ gv = "COMMIT" THEN "COMMIT" ELSE "ABORT"

101 ComputeDecision(vs) ≜
102     IF ∀ v ∈ vs : v = "COMMIT" THEN "COMMIT" ELSE "ABORT"
103 ----------------------------------------------------------------------------
104 Certify(t) ≜  Certify t ∈ Tid
105     ∧ t ∈ Tid \ submitted
106     ∧ Send([type : {"PREPARE"}, t : {t}, s : TSharding(t)])
107     ∧ submitted' = submitted ∪ {t}
108     ∧ UNCHANGED sVars

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

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

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

```
140             ∧ m.type = "DECISION"
141             ∧ m.s = s
142             ∧ dec' = [dec EXCEPT ![s][m.p] = m.d]
143             ∧ phase' = [phase EXCEPT ![s][m.p] = "DECIDED"]
144             ∧ Delete({m})
145         ∧ UNCHANGED ⟨next, txn, vote, submitted⟩
146 ----------------------------------------------------------------------------
147 Next ≜
148         ∨ ∃ t ∈ Tid : Certify(t)
149         ∨ ∃ t ∈ Tid, s ∈ Shard :
150             ∨ Prepare(t, s)
151             ∨ PrepareAck(t, s)
152         ∨ ∃ s ∈ Shard :
153             ∨ Decision(s)
154
155 Spec ≜ Init ∧ □[Next]_vars
156 ============================================================================
    \* Modification History
    \* Last modified Sun Jun 13 18:09:18 CST 2021 by hengxin
    \* Created Sat Jun 12 21:01:57 CST 2021 by hengxin
```