MODULE *TxCure*

Transactional Cure Protocol without Strong Transactions.

TODO:

- Values are irrelevant.

EXTENDS *Naturals*, *FiniteSets*, *TLC*, *SequenceUtils*, *RelationUtils*, *MathUtils*

CONSTANTS

Key,	the set of keys, ranged over by $k \in Key$
Value,	the set of values, ranged over by $v \in Value$
Client,	the set of clients, ranged over by $c \in Client$
Partition,	the set of partitions, ranged over by $p \in Partition$
Datacenter,	the set of datacenters, ranged over by $d \in Datacenter$
KeySharding,	the mapping from *Key* to *Partition*
ClientAttachment	the mapping from *Client* to *Datacenter*

$NotVal \triangleq$ CHOOSE $v : v \notin Value$

ASSUME

$\land\ KeySharding \in [Key \to Partition]$

$\land\ ClientAttachment \in [Client \to Datacenter]$

VARIABLES

At the client side:

cvc,	$cvc[c]$: vector clock of client $c \in Client$
tid,	$tid[c]$: transaction identifier of the current ongoing transaction of client $c \in Client$
coord,	$coord[c]$: coordinator (partition) of the current ongoing transaction of client $c \in Client$

At the server side (each for partition $p \in Partition$ in $d \in Datacenter$):

rs,	$rs[p][d][tid][i]$: read set of transaction *tid*, indexed by partition $i \in Partition$
ws,	$ws[p][d][tid][i]$: write set of transaction *tid*, indexed by partition $i \in Partition$
seq,	$seq[p][d]$: *seq* number
opLog,	$opLog[p][d]$: log
clock,	$clock[p][d]$: current clock
knownVC,	$knownVC[p][d]$: vector clock
stableVC,	$stableVC[p][d]$: stable snapshot
uniformVC,	$uniformVC[p][d]$: uniform snapshot
snapshotVC,	$snapshotVC[p][d][t]$: snapshot vector clock of transaction t

history:

L,	$L[c]$: local history at client $c \in Client$

communication:

msgs,	the set of messages in transit
incoming	$incoming[p][d]$: incoming *FIFO* channel for propagating updates and heartbeats

$cVars \triangleq \langle cvc, tid, coord \rangle$

$sVars \triangleq \langle opLog, clock, knownVC, stableVC, uniformVC, snapshotVC \rangle$

$mVars \triangleq \langle msgs, incoming \rangle$

```
hVars ≜ ⟨L⟩
vars ≜ ⟨cVars, sVars, mVars, hVars⟩
```

```
Tid ≜ [seq : Nat, p : Partition, d : Datacenter]  transaction identifier

VC ≜ [Datacenter → Nat]  vector clock with an entry per datacenter d ∈ Datacenter
VCInit ≜ [d ∈ Datacenter ↦ 0]
Merge(vc1, vc2) ≜ [d ∈ Datacenter ↦ Max(vc1[d], vc2[d])]  TODO: except d?

DC ≜ Cardinality(Datacenter)
DCIndex ≜ CHOOSE f ∈ [1 .. DC → Datacenter] : Injective(f)
LTE(vc1, vc2) ≜  less-than-or-equal-to comparator for vector clocks
    LET RECURSIVE LTEHelper(_, _, _)
        LTEHelper(vc1h, vc2h, index) ≜
            IF index > DC THEN TRUE  EQ
            ELSE LET d ≜ DCIndex[index]
                 IN  CASE vc1h[d] < vc2h[d] → TRUE   LT
                       □  vc1h[d] > vc2h[d] → FALSE  GT
                       □  OTHER → LTEHelper(vc1h, vc2h, index + 1)
    IN  LTEHelper(vc1, vc2, 1)

KVTuple ≜ [key : Key, val : Value ∪ {NotVal}, vc : VC]
OpTuple ≜ [type : {"R", "W"}, kv : KVTuple, c : Client, cnt : Nat]

Message ≜
        [type : {"StartRequest"}, vc : VC, c : Client, p : Partition, d : Datacenter]
    ∪   [type : {"StartReply"}, tid : Tid, vc : VC, c : Client]
    ∪   [type : {"ReadRequest"}, tid : Tid, key : Key, c : Client, p : Partition, d : Datacenter]
    ∪   [type : {"ReadReply"}, val : Value ∪ {NotVal}, c : Client]  val is irrelevant
    ∪   [type : {"UpdateRequest"}, tid : Tid, key : Key, val : Value, c : Client, p : Partition, d : Datacenter]
    ∪   [type : {"UpdateReply"}, c : Client]
    ∪   [type : {"CommitRequest"}, tid : Tid, c : Client, p : Partition, d : Datacenter]  val is irrelevant
    ∪   [type : {"CommitReply"}, vc : VC, c : Client]
    ∪   [type : {"Replicate"}, d : Datacenter, kv : KVTuple]
    ∪   [type : {"Heartbeat"}, d : Datacenter, ts : Nat]

Send(m) ≜ msgs′ = msgs ∪ {m}
SendAndDelete(sm, dm) ≜ msgs′ = (msgs ∪ {sm}) \ {dm}

TypeOK ≜
    ∧  cvc ∈ [Client → VC]
    ∧  clock ∈ [Partition → [Datacenter → Nat]]
    ∧  knownVC ∈ [Partition → [Datacenter → VC]]
    ∧  stableVC ∈ [Partition → [Datacenter → VC]]
    ∧  uniformVC ∈ [Partition → [Datacenter → VC]]
    ∧  snapshotVC ∈ [Partition → [Datacenter → [Tid → VC]]]
    ∧  opLog ∈ [Partition → [Datacenter → SUBSET KVTuple]]
```

```
97      ∧   msgs ⊆ Message
98      ∧   incoming ∈ [Partition → [Datacenter → Seq(Message)]]
99      ∧   L ∈ [Client → Seq(OpTuple)]
100 ├──────────────────────────────────────────────────────────────────────┤
101 Init ≜
102     ∧ cvc = [c ∈ Client ↦ VCInit]
103     ∧ clock = [p ∈ Partition ↦ [d ∈ Datacenter ↦ 0]]
104     ∧ knownVC = [p ∈ Partition ↦ [d ∈ Datacenter ↦ VCInit]]
105     ∧ stableVC = [p ∈ Partition ↦ [d ∈ Datacenter ↦ VCInit]]
106     ∧ uniformVC = [p ∈ Partition ↦ [d ∈ Datacenter ↦ VCInit]]
107     ∧ snapshotVC = [p ∈ Partition ↦ [d ∈ Datacenter ↦ [t ∈ Tid ↦ VCInit]]]
108     ∧ opLog = [p   ∈ Partition ↦ [d ∈ Datacenter ↦
109                    [key : {k ∈ Key : KeySharding[k] = p}, val : {NotVal}, vc : {VCInit}]]]
110     ∧ msgs = {}
111     ∧ incoming = [p ∈ Partition ↦ [d ∈ Datacenter ↦ ⟨⟩]]
112     ∧ L = [c ∈ Client ↦ ⟨⟩]
113 ├──────────────────────────────────────────────────────────────────────┤
114  Client operations at client c ∈ Client.

116 CanIssue(c) ≜ ∀ m ∈ msgs :  to ensure well-formedness of clients
117     m.type ∈ {"StartRequest", "StartReply",
118           "ReadRequest", "ReadReply",
119           "UpdateRequest", "UpdateReply"
120           "CommitRequest", "CommitReply"} ⇒ m.c ≠ c

122 Start(c) ≜   c ∈ Client starts a transaction
123     ∧ CanIssue(c)
124     ∧ ∃ p ∈ Partition :
125         ∧ coord′ = [coord EXCEPT ![c] = p]
126         ∧ Send([type ↦ "StartRequest", vc ↦ cvc[c],
127                 c ↦ c, p ↦ p, d ↦ ClientAttachment[c]])
128     ∧ UNCHANGED ⟨cvc, tid, sVars, incoming, hVars⟩

130 StartReply(c) ≜   c ∈ Client handles the reply to its start request
131     ∧ ∃ m ∈ msgs :
132         ∧ m.type = "StartReply" ∧ m.c = c  such m is unique due to well-formedness
133         ∧ cvc′ = [cvc EXCEPT ![c] = m.snapshotVC]
134         ∧ tid′ = [tid EXCEPT ![c] = m.tid]
135         ∧ msgs′ = msgs \ {m}
136     ∧ UNCHANGED ⟨coord, sVars, incoming, hVars⟩

138 Read(c, k) ≜   c ∈ Client reads from k ∈ Key
139       ∧ CanIssue(c)
140       ∧ Send([type ↦ "ReadRequest", tid ↦ tid[c], key ↦ k,
141               c ↦ c, p ↦ coord[c], d ↦ ClientAttachment[c]])
142       ∧ UNCHANGED ⟨cVars, sVars, incoming, hVars⟩
```

```
144 ReadReply(c) ≜  c ∈ Client handles the reply to its read request
145     ∧ ∃ m ∈ msgs :
146         ∧ m.type = "ReadReply" ∧ m.c = c   such m is unique due to well-formedness
147         ∧ msgs′ = msgs \ {m}
148     ∧ UNCHANGED ⟨cVars, sVars, incoming, hVars⟩

150 Update(c, k, v) ≜  c ∈ Client updates k ∈ Key with v ∈ Value
151     ∧ CanIssue(c)
152     ∧ Send([type ↦ "UpdateRequest", tid ↦ tid[c], key ↦ k, val ↦ v,
153                 c ↦ c, p ↦ coord[c], d    ↦ ClientAttachment[c]])
154     ∧ UNCHANGED ⟨cVars, sVars, incoming, hVars⟩

156 UpdateReply(c) ≜  c ∈ Client handles the reply to its update request
157     ∧ ∃ m ∈ msgs :
158         ∧ m.type = "UpdateReply" ∧ m.c = c  such m is unique due to well-formedness
159         ∧ msgs′ = msgs \ {m}
160     ∧ UNCHANGED ⟨cVars, sVars, incoming, hVars⟩

162 Commit(c) ≜  c ∈ Client commits the ongoing transaction tid[c]
163     ∧ CanIssue(c)
164     ∧ Send([type ↦ "CommitRequest", tid ↦ tid[c],
165                 c ↦ c, p ↦ coord[c], d    ↦ ClientAttachment[c]])
166     ∧ UNCHANGED ⟨cVars, sVars, incoming, hVars⟩

168 CommitReply(c) ≜  c ∈ Client handles the reply to its commit request
169     ∧ ∃ m ∈ msgs :
170         ∧ m.type = "CommitReply" ∧ m.c = c  such m is unique due to well-formedness
171         ∧ cvc′ = [cvc EXCEPT !c = [m.vc]]
172         ∧ msgs′ = msgs \ {m}
173     ∧ UNCHANGED ⟨tid, coord, sVars, incoming, hVars⟩
174 ├──────────────────────────────────────────────────────────────────────┤
175  Server operations at partition p ∈ Partition in datacenter d ∈ Datacenter.

177 StartRequest(p, d) ≜  handle a "StartRequest"
178     ∧ ∃ m ∈ msgs :
179         ∧ m.type = "StartRequest" ∧ m.p = p ∧ m.d = d
180         ∧ uniformVC′ = [uniformVC EXCEPT ![p][d] = Merge(m.vc, @)]
181         ∧ seq′ = [seq EXCEPT ![p][d] = @ + 1]
182         ∧ LET t ≜ [seq ↦ seq, p ↦ p, d ↦ d]
183           IN  ∧ snapshotVC′ = [snapshotVC EXCEPT
184                     ![p][d][t] = [dc ∈ Datacenter ↦
185                                   IF dc = d
186                                   THEN Max(m.vc[d], uniformVC[p][d][d])
187                                   ELSE uniformVC′[p][d][d]]]
188               ∧ SendAndDelete([type ↦ "StartReply", tid ↦ t,
189                                vc ↦ snapshotVC′[p][d][t], c ↦ m.c], m)
```

```
190     ∧ UNCHANGED ⟨cVars, clock, knownVC, stableVC, opLog, incoming⟩

192 ReadRequest(p, d) ≜  handle a "ReadRequest"
193     ∧ ∃ m ∈ msgs :
194         ∧ m.type = "ReadRequest" ∧ m.p = p ∧ m.d = d
195         ∧ SendAndDelete([type ↦ "ReadReply", val ↦ lkv.val, vc ↦ lkv.vc, c ↦ m.c], m)
196           ∧ L' = [L EXCEPT ![m.c] = Append(@, [type ↦ "R", kv ↦ lkv, c ↦ m.c, cnt ↦ Len(@) + 1])]
197     ∧ UNCHANGED ⟨cVars, clock, knownVC, opLog, incoming⟩

199 UpdateRequest(p, d) ≜  handle a "UpdateRequest"
200     ∧ ∃ m ∈ msgs :
201         ∧ m.type = "UpdateRequest" ∧ m.p = p ∧ m.d = d
202         ∧ ws' = [ws EXCEPT ![p][d][m.tid][KeySharding[m.key]][m.key] = m.val]
203         ∧ SendAndDelete([type ↦ "UpdateReply", c ↦ m.c], m)
204           ∧ L' = [L EXCEPT ![m.c] = Append(@, [type ↦ "W", kv ↦ kv, c ↦ m.c, cnt ↦ Len(@) + 1])]
205     ∧ UNCHANGED ⟨cVars, clock, knownVC⟩

207 Replicate(p, d) ≜  handle a "Replicate"
208     ∧ incoming[p][d] ≠ ⟨⟩
209     ∧ LET m ≜ Head(incoming[p][d])
210       IN  ∧ m.type = "Replicate"
211           ∧ opLog' = [opLog EXCEPT ![p][d] = @ ∪ {m.kv}]
212           ∧ knownVC' = [knownVC EXCEPT ![p][d][m.d] = m.kv.vc[m.d]]
213           ∧ incoming' = [incoming EXCEPT ![p][d] = Tail(@)]
214     ∧ UNCHANGED ⟨cVars, cvc, clock, stableVC, L, msgs⟩

216 Heartbeat(p, d) ≜  handle a "Heartbeat"
217     ∧ incoming[p][d] ≠ ⟨⟩
218     ∧ LET m ≜ Head(incoming[p][d])
219       IN  ∧ m.type = "Heartbeat"
220           ∧ knownVC' = [knownVC EXCEPT ![p][d][m.d] = m.ts]
221           ∧ incoming' = [incoming EXCEPT ![p][d] = Tail(@)]
222     ∧ UNCHANGED ⟨cVars, cvc, clock, stableVC, opLog, L, msgs⟩
223 ├──────────────────────────────────────────────────────────────────────────────┤
224  Clock management at partition p ∈ Partition in datacenter d ∈ Datacenter
225 Tick(p, d) ≜  clock[p][d] ticks
226     ∧ clock' = [clock EXCEPT ![p][d] = @ + 1]
227     ∧ knownVC' = [knownVC EXCEPT ![p][d][d] = clock'[p][d]]
228     ∧ incoming' = [incoming EXCEPT ![p] = [dc ∈ Datacenter ↦
229         IF dc = d THEN @[dc] ELSE Append(@[dc], [type ↦ "Heartbeat", d ↦ d, ts ↦ knownVC'[p][d][d]])]]
230     ∧ UNCHANGED ⟨cVars, cvc, stableVC, opLog, L, msgs⟩

232 UpdateCSS(p, d) ≜  update stableVC[p][d]
233     ∧ stableVC' = [stableVC EXCEPT ![p][d] =
234                 [dc ∈ Datacenter ↦ SetMin({knownVC[pp][d][dc] : pp ∈ Partition})]]
235     ∧ UNCHANGED ⟨cVars, mVars, clock, knownVC, opLog, L⟩
```

```
Next ≜
    ∨ ∃ c ∈ Client, k ∈ Key : Read(c, k)
    ∨ ∃ c ∈ Client, k ∈ Key, v ∈ Value : Update(c, k, v)
    ∨ ∃ c ∈ Client : ReadReply(c) ∨ UpdateReply(c)
    ∨ ∃ p ∈ Partition, d ∈ Datacenter :
        ∨ ReadRequest(p, d)
        ∨ UpdateRequest(p, d)
        ∨ Replicate(p, d)
        ∨ Heartbeat(p, d)
        ∨ Tick(p, d)
        ∨ UpdateCSS(p, d)

Spec ≜ Init ∧ □[Next]_vars
```

* Modification History
* Last modified *Mon Nov* 23 16:11:35 *CST* 2020 by *hengxin*
* Created *Fri Nov* 20 18:51:11 *CST* 2020 by *hengxin*