MODULE *CureKV*

See *ICDCS*2016: "Cure: Strong Semantics Meets High Availability and Low Latency".

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

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

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

$KeysOnPartition \triangleq [p \in Partition \mapsto \{k \in Key : KeySharding[k] = p\}]$

VARIABLES

At the client side:

- *cvc*, $cvc[c]$: the vector clock of client $c \in Client$

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

- *clock*, $clock[p][d]$: the current clock
- *pvc*, $pvc[p][d]$: the vector clock
- *css*, $css[p][d]$: the stable snapshot
- *store*, $store[p][d]$: the *kv* store (represented by a map)
- *remote*, $remote[p][d]$: the set of remote updates that have not been applied yet

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

$cVars \triangleq \langle cvc \rangle$

$sVars \triangleq \langle clock, pvc, css, store, remote, L \rangle$

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

$vars \triangleq \langle cvc, clock, pvc, css, store, remote, L, msgs, incoming \rangle$

$VC \triangleq [Datacenter \to Nat]$ vector clock with an entry per datacenter $d \in Datacenter$

$VCInit \triangleq [d \in Datacenter \mapsto 0]$

$Merge(vc1, vc2) \triangleq [d \in Datacenter \mapsto Max(vc1[d], vc2[d])]$

$DC \triangleq Cardinality(Datacenter)$

```
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 : {"ReadRequest"}, key : Key, vc : VC, c : Client, p : Partition, d : Datacenter]
    ∪     [type : {"ReadReply"}, val : Value ∪ {NotVal}, vc : VC, c : Client]
    ∪     [type : {"UpdateRequest"}, key : Key, val : Value, vc : VC, c : Client, p : Partition, d : Datacenter]
    ∪     [type : {"UpdateReply"}, ts : Nat, c : Client, d : Datacenter]
    ∪     [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]]
    ∧  pvc ∈ [Partition → [Datacenter → VC]]
    ∧  css ∈ [Partition → [Datacenter → VC]]
    ∧  store ∈ [Partition → [Datacenter → [Key → KVTuple]]]
    ∧  remote ∈ [Partition → [Datacenter → SUBSET KVTuple]]
    ∧  msgs ⊆ Message
    ∧  incoming ∈ [Partition → [Datacenter → Seq(Message)]]
    ∧  L ∈ [Client → Seq(OpTuple)]
⊢————————————————————————————————————————————————————————————————————————
Init ≜
    ∧ cvc = [c ∈ Client ↦ VCInit]
    ∧ clock = [p ∈ Partition ↦ [d ∈ Datacenter ↦ 0]]
    ∧ pvc = [p ∈ Partition ↦ [d ∈ Datacenter ↦ VCInit]]
    ∧ css = [p ∈ Partition ↦ [d ∈ Datacenter ↦ VCInit]]
    ∧ store = [p ∈ Partition ↦ [d ∈ Datacenter ↦
                  [k ∈ KeysOnPartition[p] ↦ [key ↦ k, val ↦ NotVal, vc ↦ VCInit]]]]
    ∧ remote = [p ∈ Partition ↦ [d ∈ Datacenter ↦ {}]]
    ∧ msgs = {}
    ∧ incoming = [p ∈ Partition ↦ [d ∈ Datacenter ↦ ⟨⟩]]
```

```
95      ∧ L = [c ∈ Client ↦ ⟨⟩]
96  ├──────────────────────────────────────────────────────────────────────┤
97   Client operations at client c ∈ Client.
99  CanIssue(c) ≜ ∀ m ∈ msgs :  to ensure well-formedness of clients
100     m.type ∈ {"ReadRequest", "ReadReply", "UpdateRequest", "UpdateReply"} ⇒ m.c ≠ c
102 Read(c, k) ≜   c ∈ Client reads from k ∈ Key
103     ∧ CanIssue(c)
104     ∧ Send([type ↦ "ReadRequest", key ↦ k, vc ↦ cvc[c],
105             c ↦ c, p ↦ KeySharding[k], d ↦ ClientAttachment[c]])
106     ∧ UNCHANGED ⟨cVars, sVars, incoming⟩
108 ReadReply(c) ≜   c ∈ Client handles the reply to its read request
109     ∧ ∃ m ∈ msgs :
110         ∧ m.type = "ReadReply" ∧ m.c = c   such m is unique due to well-formedness
111         ∧ cvc' = [cvc EXCEPT ![c] = Merge(m.vc, @)]
112         ∧ msgs' = msgs \ {m}
113     ∧ UNCHANGED ⟨sVars, incoming⟩
115 Update(c, k, v) ≜   c ∈ Client updates k ∈ Key with v ∈ Value
116     ∧ CanIssue(c)
117     ∧ Send([type ↦ "UpdateRequest", key ↦ k, val ↦ v,
118             vc ↦ cvc[c], c ↦ c, p ↦ KeySharding[k], d ↦ ClientAttachment[c]])
119     ∧ UNCHANGED ⟨cVars, sVars, incoming⟩
121 UpdateReply(c) ≜   c ∈ Client handles the reply to its update request
122     ∧ ∃ m ∈ msgs :
123         ∧ m.type = "UpdateReply" ∧ m.c = c   such m is unique due to well-formedness
124         ∧ cvc' = [cvc EXCEPT ![c][m.d] = m.ts]
125         ∧ msgs' = msgs \ {m}
126     ∧ UNCHANGED ⟨sVars, incoming⟩
127 ├──────────────────────────────────────────────────────────────────────┤
128  Server operations at partition p ∈ Partition in datacenter d ∈ Datacenter.
130 Apply(p, d) ≜   apply remote updates which become stable due to up-to-date css[p][d]
131     LET keys ≜ {kv.key : kv ∈ remote[p][d]}
132         kvs ≜ [k ∈ keys ↦ {kv ∈ remote[p][d] :
133                              ∧ kv.key = k
134                              ∧ ∀ dc ∈ Datacenter \ {d} : kv.vc[dc] ≤ css'[p][d][dc]}]
135     IN  ∧ store' = [store EXCEPT ![p][d] = [k ∈ KeysOnPartition[p] ↦
136                       IF k ∈ keys ∧ kvs[k] ≠ {}
137                       THEN CHOOSE kv ∈ kvs[k] : ∀ akv ∈ kvs[k] : LTE(akv.vc, kv.vc)
138                       ELSE @[k]]]
139         ∧ remote' = [remote EXCEPT ![p][d] = @ \ (UNION (Range(kvs)))]
141 ReadRequest(p, d) ≜   handle a "ReadRequest"
```

```
    ∧ ∃ m ∈ msgs :
        ∧ m.type = "ReadRequest" ∧ m.p = p ∧ m.d = d
        ∧ css' = [css EXCEPT ![p][d] = Merge(m.vc, @)]
        ∧ LET kv ≜ store[p][d][m.key]
          IN  ∧ SendAndDelete([type ↦ "ReadReply", val ↦ kv.val, vc ↦ kv.vc, c ↦ m.c], m)
              ∧ Apply(p, d)
              ∧ L' = [L EXCEPT ![m.c] = Append(@, [type ↦ "R", kv ↦ kv, c ↦ m.c, cnt ↦ Len(@) + 1])]
    ∧ UNCHANGED ⟨cVars, clock, pvc, incoming⟩

UpdateRequest(p, d) ≜  handle a "UpdateRequest"
    ∧ ∃ m ∈ msgs :
        ∧ m.type = "UpdateRequest" ∧ m.p = p ∧ m.d = d
        ∧ m.vc[d] < clock[p][d]   waiting condition; (" ≤ " strengthed to " < ")
        ∧ css' = [css EXCEPT ![p][d] = Merge(m.vc, @)]
        ∧ LET kv ≜ [key ↦ m.key, val ↦ m.val,
                    vc ↦ [m.vc EXCEPT ![d] = clock[p][d]]]
          IN  ∧ store' = [store EXCEPT ![p][d][m.key] = kv]  TODO: Apply???
              ∧ SendAndDelete([type ↦ "UpdateReply", ts ↦ clock[p][d], c ↦ m.c, d ↦ d], m)
              ∧ incoming' = [incoming EXCEPT ![p] = [dc ∈ Datacenter ↦
                  IF dc = d THEN @[dc] ELSE Append(@[dc], [type ↦ "Replicate", d ↦ d, kv ↦ kv])]]
              ∧ L' = [L EXCEPT ![m.c] = Append(@, [type ↦ "W", kv ↦ kv, c ↦ m.c, cnt ↦ Len(@) + 1])]
    ∧ UNCHANGED ⟨cVars, clock, pvc, remote⟩

Replicate(p, d) ≜  handle a "Replicate"
    ∧ incoming[p][d] ≠ ⟨⟩
    ∧ LET m ≜ Head(incoming[p][d])
      IN  ∧ m.type = "Replicate"
          ∧ remote' = [remote EXCEPT ![p][d] = @ ∪ {m.kv}]
          ∧ pvc' = [pvc EXCEPT ![p][d][m.d] = m.kv.vc[m.d]]
          ∧ incoming' = [incoming EXCEPT ![p][d] = Tail(@)]
    ∧ UNCHANGED ⟨cVars, cvc, clock, css, store, L, msgs⟩

Heartbeat(p, d) ≜  handle a "Heartbeat"
    ∧ incoming[p][d] ≠ ⟨⟩
    ∧ LET m ≜ Head(incoming[p][d])
      IN  ∧ m.type = "Heartbeat"
          ∧ pvc' = [pvc EXCEPT ![p][d][m.d] = m.ts]
          ∧ incoming' = [incoming EXCEPT ![p][d] = Tail(@)]
    ∧ UNCHANGED ⟨cVars, cvc, clock, css, store, remote, L, msgs⟩
├──────────────────────────────────────────────────────────────────────────┤
 Clock management at partition p ∈ Partition in datacenter d ∈ Datacenter
Tick(p, d) ≜  clock[p][d] ticks
    ∧ clock' = [clock EXCEPT ![p][d] = @ + 1]
    ∧ pvc' = [pvc EXCEPT ![p][d][d] = clock'[p][d]]
    ∧ incoming' = [incoming EXCEPT ![p] = [dc ∈ Datacenter ↦
        IF dc = d THEN @[dc] ELSE Append(@[dc], [type ↦ "Heartbeat", d ↦ d, ts ↦ pvc'[p][d][d]])]]
```

```
188       ∧ UNCHANGED ⟨cVars, cvc, css, store, remote, L, msgs⟩

190  UpdateCSS(p, d) ≜   update css[p][d]
191      ∧ css′ = [css EXCEPT ![p][d] =
192                  [dc ∈ Datacenter ↦ SetMin({pvc[pp][d][dc] : pp ∈ Partition})]]
193      ∧ Apply(p, d)
194      ∧ UNCHANGED ⟨cVars, mVars, clock, pvc, L⟩
195  ──────────────────────────────────────────────────────────────────────────
196  Next ≜
197      ∨ ∃ c ∈ Client, k ∈ Key : Read(c, k)
198      ∨ ∃ c ∈ Client, k ∈ Key, v ∈ Value : Update(c, k, v)
199      ∨ ∃ c ∈ Client : ReadReply(c) ∨ UpdateReply(c)
200      ∨ ∃ p ∈ Partition, d ∈ Datacenter :
201          ∨ ReadRequest(p, d)
202          ∨ UpdateRequest(p, d)
203          ∨ Replicate(p, d)
204          ∨ Heartbeat(p, d)
205          ∨ Tick(p, d)
206          ∨ UpdateCSS(p, d)

208  Spec ≜ Init ∧ □[Next]_vars
209  ──────────────────────────────────────────────────────────────────────────
210  Valid(s) ≜   Is s a valid serialization?
211      LET RECURSIVE ValidHelper(_, _)
212          ValidHelper(seq, kvs) ≜
213              IF seq = ⟨⟩ THEN TRUE
214               ELSE  LET op ≜ Head(seq)
215                     IN   IF op.type = "W"                                  overwritten
216                          THEN  ValidHelper(Tail(seq), op.kv.key :> op.kv.vc @@ kvs)
217                          ELSE   ∧ op.kv.vc = kvs[op.kv.key]
218                                 ∧ ValidHelper(Tail(seq), kvs)
219      IN     ValidHelper(s, [k ∈ Key ↦ VCInit])   with initial values

221  CM ≜   causal memory consistency model; see Ahamad@DC′1995
222       LET ops ≜ UNION {Range(L[c]) : c ∈ Client}
223           rops  ≜ {op ∈ ops : op.type    = "R"}
224           wops ≜ {op ∈ ops : op.type    = "W"}
225           so   ≜ UNION {SeqToRel(L[c]) : c ∈ Client}   session order
226           rf   ≜ {⟨w, r⟩ ∈ wops × rops : w.kv.key = r.kv.key ∧ w.kv.vc = r.kv.vc}
227            co  ≜ TC(so ∪ rf)   causality order
228       IN    ∀ c ∈ Client :
229               ∃ sc ∈ PermutationsOf(L[c] ∘ SetToSeq(wops)) :
230                   ∧ Valid(sc)
231                   ∧ Respect(sc, co)

233  THEOREM Spec ⇒ □CM
```

```
234 └──────────────────────────────────────────────────────────────────────┘
    \ * Modification History
    \ * Last modified Thu Jan 30 23:55:36 CST 2020 by hengxin
    \ * Created Tue Jan 28 21:28:25 CST 2020 by hengxin
```