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

EXTENDS *Naturals*, *Sequences*, *FiniteSets*, *TLC*, *RelationUtils*

$Max(a, b) \triangleq$ IF $a < b$ THEN b ELSE a
$Min(S) \triangleq$ CHOOSE $a \in S : \forall\, b \in S : a \leq b$

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*, $cvc[c]$: 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

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

$cVars \triangleq \langle cvc \rangle$
$sVars \triangleq \langle clock, pvc, css, store, L \rangle$
$mVars \triangleq \langle msgs, incoming \rangle$
$vars \triangleq \langle cvc, clock, pvc, css, store, L, msgs, incoming \rangle$

$VC \triangleq [Datacenter \to Nat]$ 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]

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 → 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 ↦
                  [key : {k ∈ Key : KeySharding[k] = p}, val : {NotVal}, vc : {VCInit}]]]
    ∧ msgs = {}
    ∧ incoming = [p ∈ Partition ↦ [d ∈ Datacenter ↦ ⟨⟩]]
    ∧ L = [c ∈ Client ↦ ⟨⟩]
├──────────────────────────────────────────────────────────────────────────┤
```

```
95   Client operations at client c ∈ Client.

97   CanIssue(c) ≜ ∀ m ∈ msgs :  to ensure well-formedness of clients
98       m.type ∈ {"ReadRequest", "ReadReply", "UpdateRequest", "UpdateReply"} ⇒ m.c ≠ c

100  Read(c, k) ≜  c ∈ Client reads from k ∈ Key
101      ∧ CanIssue(c)
102      ∧ Send([type ↦ "ReadRequest", key ↦ k, vc ↦ cvc[c],
103              c ↦ c, p ↦ KeySharding[k], d ↦ ClientAttachment[c]])
104      ∧ UNCHANGED ⟨cVars, sVars, incoming⟩

106  ReadReply(c) ≜  c ∈ Client handles the reply to its read request
107      ∧ ∃ m ∈ msgs :
108          ∧ m.type = "ReadReply" ∧ m.c = c  such m is unique due to well-formedness
109          ∧ cvc' = [cvc EXCEPT ![c] = Merge(m.vc, @)]
110          ∧ msgs' = msgs \ {m}
111      ∧ UNCHANGED ⟨sVars, incoming⟩

113  Update(c, k, v) ≜  c ∈ Client updates k ∈ Key with v ∈ Value
114      ∧ CanIssue(c)
115      ∧ Send([type ↦ "UpdateRequest", key ↦ k, val ↦ v,
116              vc ↦ cvc[c], c ↦ c, p ↦ KeySharding[k], d ↦ ClientAttachment[c]])
117      ∧ UNCHANGED ⟨cVars, sVars, incoming⟩

119  UpdateReply(c) ≜  c ∈ Client handles the reply to its update request
120      ∧ ∃ m ∈ msgs :
121          ∧ m.type = "UpdateReply" ∧ m.c = c  such m is unique due to well-formedness
122          ∧ cvc' = [cvc EXCEPT ![c][m.d] = m.ts]
123          ∧ msgs' = msgs \ {m}
124      ∧ UNCHANGED ⟨sVars, incoming⟩
125  ├──────────────────────────────────────────────────────────────────────────┤
126  Server operations at partition p ∈ Partition in datacenter d ∈ Datacenter.

128  ReadRequest(p, d) ≜  handle a "ReadRequest"
129      ∧ ∃ m ∈ msgs :
130          ∧ m.type = "ReadRequest" ∧ m.p = p ∧ m.d = d
131          ∧ css' = [css EXCEPT ![p][d] = Merge(m.vc, @)]
132          ∧ LET kvs ≜ {kv ∈ store[p][d] :
133                          ∧ kv.key = m.key
134                          ∧ ∀ dc ∈ Datacenter \ {d} : kv.vc[dc] ≤ css'[p][d][dc]}
135                lkv ≜ CHOOSE kv ∈ kvs : ∀ akv ∈ kvs : LTE(akv.vc, kv.vc)
136            IN  ∧ SendAndDelete([type ↦ "ReadReply", val ↦ lkv.val, vc ↦ lkv.vc, c ↦ m.c], m)
137                ∧ L' = [L EXCEPT ![m.c] = Append(@, [type ↦ "R", kv ↦ lkv])]
138      ∧ UNCHANGED ⟨cVars, clock, pvc, store, incoming⟩

140  UpdateRequest(p, d) ≜  handle a "UpdateRequest"
141      ∧ ∃ m ∈ msgs :
```

```
142         ∧ m.type = "UpdateRequest" ∧ m.p = p ∧ m.d = d
143         ∧ m.vc[d] < clock[p][d]   waiting condition; (" ≤ " strengthed to " < ")
144         ∧ css' = [css EXCEPT ![p][d] = Merge(m.vc, @)]
145         ∧ LET kv ≜ [key ↦ m.key, val ↦ m.val,
146                     vc ↦ [m.vc EXCEPT ![d] = clock[p][d]]]
147           IN  ∧ store' = [store EXCEPT ![p][d] = @ ∪ {kv}]
148               ∧ SendAndDelete([type ↦ "UpdateReply", ts ↦ clock[p][d], c ↦ m.c, d ↦ d], m)
149               ∧ incoming' = [incoming EXCEPT ![p] = [dc ∈ Datacenter ↦
150                   IF dc = d THEN @[dc] ELSE Append(@[dc], [type ↦ "Replicate", d ↦ d, kv ↦ kv])]]
151               ∧ L' = [L EXCEPT ![m.c] = Append(@, [type ↦ "R", kv ↦ kv])]
152     ∧ UNCHANGED ⟨cVars, clock, pvc⟩

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

163 Heartbeat(p, d) ≜   handle a "Heartbeat"
164     ∧ incoming[p][d] ≠ ⟨⟩
165     ∧ LET m ≜ Head(incoming[p][d])
166       IN  ∧ m.type = "Heartbeat"
167           ∧ pvc' = [pvc EXCEPT ![p][d][m.d] = m.ts]
168           ∧ incoming' = [incoming EXCEPT ![p][d] = Tail(@)]
169     ∧ UNCHANGED ⟨cVars, cvc, clock, css, store, L, msgs⟩
170 ──────────────────────────────────────────────────────────────────────
171  Clock management at partition p ∈ Partition in datacenter d ∈ Datacenter
172 Tick(p, d) ≜   clock[p][d] ticks
173     ∧ clock' = [clock EXCEPT ![p][d] = @ + 1]
174     ∧ pvc' = [pvc EXCEPT ![p][d][d] = clock'[p][d]]
175     ∧ incoming' = [incoming EXCEPT ![p] = [dc ∈ Datacenter ↦
176         IF dc = d THEN @[dc] ELSE Append(@[dc], [type ↦ "Heartbeat", d ↦ d, ts ↦ pvc'[p][d][d]])]]
177     ∧ UNCHANGED ⟨cVars, cvc, css, store, L, msgs⟩

179 UpdateCSS(p, d) ≜   update css[p][d]
180     ∧ css' = [css EXCEPT ![p][d] =
181                 [dc ∈ Datacenter ↦ Min({pvc[pp][d][dc] : pp ∈ Partition})]]
182     ∧ UNCHANGED ⟨cVars, mVars, clock, pvc, store, L⟩
183 ──────────────────────────────────────────────────────────────────────
184 Next ≜
185     ∨ ∃ c ∈ Client, k ∈ Key : Read(c, k)
186     ∨ ∃ c ∈ Client, k ∈ Key, v ∈ Value : Update(c, k, v)
187     ∨ ∃ c ∈ Client : ReadReply(c) ∨ UpdateReply(c)
```

```
188       ∨ ∃ p ∈ Partition, d ∈ Datacenter :
189           ∨ ReadRequest(p, d)
190           ∨ UpdateRequest(p, d)
191           ∨ Replicate(p, d)
192           ∨ Heartbeat(p, d)
193           ∨ Tick(p, d)
194           ∨ UpdateCSS(p, d)

196   Spec ≜ Init ∧ □[Next]_vars
197 ├──────────────────────────────────────────────────────────────────────────┤
198   so ≜ UNION {SeqToRel(L[c]) : c ∈ Client}  session order

200   rf ≜  read-from (or called writes-into) relation
201       LET ops ≜ UNION {Range(L[c]) : c ∈ Client}
202           rops ≜ {op ∈ ops : op.type = "R"}
203           keys ≜ {op.kv.key : op ∈ rops}
204           wops ≜ {op ∈ ops : op.type = "W"} ∪  initial writes
205                    [type : {"W"}, kv : [key : keys, val : {NotVal}, vc : {VCInit}]]
206       IN  {⟨w, r⟩ ∈ wops × rops : w.kv.key = r.kv.key ∧ w.kv.vc = r.kv.vc}

208   co ≜ TC(so ∪ rf)  causality order

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"
216                         THEN ValidHelper(Tail(seq), kvs @@ op.kv.key :> op.kv.vc)
217                         ELSE ∧ op.kv.vc = kvs[op.kv.key]
218                              ∧ ValidHelper(Tail(seq), kvs)
219       IN  ValidHelper(s, {})

221   TODO: Handling initial writes
222   CM ≜  causal memory consistency model; see DC'1995
223       LET ops ≜ UNION {Range(L[c]) : c ∈ Client}
224           wops ≜ {op ∈ ops : op.type = "W"}
225       IN  ∀ c ∈ Client :
226               ∃ sc ∈ Seq(Range(L[c]) ∪ wops) :  TODO: performance?
227                   ∧ Valid(sc)
228                   ∧ Respect(sc, co)
229 └──────────────────────────────────────────────────────────────────────────┘
```