```
1  ---------- MODULE Cure ----------
   See ICDCS2016: "Cure: Strong Semantics Meets High Availability and Low Latency".
5  EXTENDS Naturals, FiniteSets, TLC, SequenceUtils, RelationUtils, MathUtils
6  ----------
7  CONSTANTS
8      Key,            the set of keys, ranged over by k ∈ Key
9      Value,          the set of values, ranged over by v ∈ Value
10     Client,         the set of clients, ranged over by c ∈ Client
11     Partition,      the set of partitions, ranged over by p ∈ Partition
12     Datacenter,     the set of datacenters, ranged over by d ∈ Datacenter
13     KeySharding,        the mapping from Key to Partition
14     ClientAttachment    the mapping from Client to Datacenter

16 NotVal ≜ CHOOSE v : v ∉ Value

18 ASSUME
19     ∧ KeySharding ∈ [Key → Partition]
20     ∧ ClientAttachment ∈ [Client → Datacenter]
21 ----------
22 VARIABLES
23  At the client side:
24     cvc,    cvc[c]: the vector clock of client c ∈ Client
25  At the server side (each for partition p ∈ Partition in d ∈ Datacenter):
26     clock,      clock[p][d]: the current clock
27     pvc,        pvc[p][d]: the vector clock
28     css,        css[p][d]: the stable snapshot
29     store,      store[p][d]: the kv store
30  history:
31     L,   L[c]: local history at client c ∈ Client
32  communication:
33     msgs,   the set of messages in transit
34     incoming    incoming[p][d]: incoming FIFO channel for propagating updates and heartbeats

36 cVars ≜ ⟨cvc⟩
37 sVars ≜ ⟨clock, pvc, css, store, L⟩
38 mVars ≜ ⟨msgs, incoming⟩
39 vars ≜ ⟨cvc, clock, pvc, css, store, L, msgs, incoming⟩
40 ----------
41 VC ≜ [Datacenter → Nat]   vector clock with an entry per datacenter d ∈ Datacenter
42 VCInit ≜ [d ∈ Datacenter ↦ 0]
43 Merge(vc1, vc2) ≜ [d ∈ Datacenter ↦ Max(vc1[d], vc2[d])]

45 DC ≜ Cardinality(Datacenter)
46 DCIndex ≜ CHOOSE f ∈ [1 .. DC → Datacenter] : Injective(f)
47 LTE(vc1, vc2) ≜   less-than-or-equal-to comparator for vector clocks
48     LET RECURSIVE LTEHelper(_, _, _)
```

```
49              LTEHelper(vc1h, vc2h, index) ≜
50                  IF index > DC THEN TRUE  EQ
51                   ELSE  LET d ≜ DCIndex[index]
52                         IN   CASE vc1h[d] < vc2h[d] → TRUE   LT
53                                 □   vc1h[d] > vc2h[d] → FALSE   GT
54                                 □   OTHER  → LTEHelper(vc1h, vc2h, index + 1)
55        IN    LTEHelper(vc1, vc2, 1)

57  KVTuple ≜ [key : Key, val : Value ∪ {NotVal}, vc : VC]
58  OpTuple ≜ [type : {"R", "W"}, kv : KVTuple]

60  Message ≜
61              [type : {"ReadRequest"}, key : Key, vc : VC, c : Client, p : Partition, d : Datacenter]
62      ∪       [type : {"ReadReply"}, val : Value ∪ {NotVal}, vc : VC, c : Client]
63      ∪       [type : {"UpdateRequest"}, key : Key, val : Value, vc : VC, c : Client, p : Partition, d : Datacenter]
64      ∪       [type : {"UpdateReply"}, ts : Nat, c : Client, d : Datacenter]
65      ∪       [type : {"Replicate"}, d  : Datacenter, kv : KVTuple]
66      ∪       [type : {"Heartbeat"}, d : Datacenter, ts  : Nat]

68  Send(m) ≜ msgs' = msgs ∪ {m}
69  SendAndDelete(sm, dm) ≜ msgs' = (msgs ∪ {sm}) \ {dm}

71  TypeOK ≜
72      ∧   cvc ∈ [Client → VC]
73      ∧   clock ∈ [Partition → [Datacenter → Nat]]
74      ∧   pvc ∈ [Partition → [Datacenter → VC]]
75      ∧   css ∈ [Partition → [Datacenter → VC]]
76      ∧   store ∈ [Partition → [Datacenter → SUBSET KVTuple]]
77      ∧   msgs ⊆ Message
78      ∧   incoming ∈ [Partition → [Datacenter → Seq(Message)]]
79      ∧   L ∈ [Client → Seq(OpTuple)]
80  ------------------------------------------------------------------------
81  Init ≜
82      ∧ cvc = [c ∈ Client ↦ VCInit]
83      ∧ clock = [p ∈ Partition ↦ [d ∈ Datacenter ↦ 0]]
84      ∧ pvc = [p ∈ Partition ↦ [d ∈ Datacenter ↦ VCInit]]
85      ∧ css = [p ∈ Partition ↦ [d ∈ Datacenter ↦ VCInit]]
86      ∧ store = [p ∈ Partition ↦ [d ∈ Datacenter ↦
87                    [key : {k ∈ Key : KeySharding[k] = p}, val : {NotVal}, vc : {VCInit}]]]
88      ∧ msgs = {}
89      ∧ incoming = [p ∈ Partition ↦ [d ∈ Datacenter ↦ ⟨⟩]]
90      ∧ L = [c ∈ Client ↦ ⟨⟩]
91  ------------------------------------------------------------------------
92   Client operations at client c ∈ Client.

94  CanIssue(c) ≜ ∀ m ∈ msgs :  to ensure well-formedness of clients
```

```
95      m.type ∈ {"ReadRequest", "ReadReply", "UpdateRequest", "UpdateReply"} ⇒ m.c ≠ c

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

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

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

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

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

137 UpdateRequest(p, d) ≜   handle a "UpdateRequest"
138     ∧ ∃ m ∈ msgs :
139         ∧ m.type = "UpdateRequest" ∧ m.p = p ∧ m.d = d
140         ∧ m.vc[d] < clock[p][d]   waiting condition; (" ≤ " strengthed to " < ")
141         ∧ css′ = [css EXCEPT ![p][d] = Merge(m.vc, @)]
```

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

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

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

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

```
188          ∨ Replicate(p, d)
189          ∨ Heartbeat(p, d)
190          ∨ Tick(p, d)
191          ∨ UpdateCSS(p, d)

193  Spec ≜ Init ∧ □[Next]_vars
194 ├──────────────────────────────────────────────────────────────────────────┤
195  so ≜ UNION {SeqToRel(L[c]) : c ∈ Client}  session order

197  rf ≜  read-from (or called writes-into) relation
198     LET ops ≜ UNION {Range(L[c]) : c ∈ Client}
199         rops  ≜ {op ∈ ops : op.type  = "R"}
200          wops ≜ {op ∈ ops : op.type  = "W"}
201     IN  {⟨w, r⟩ ∈ wops × rops : w.kv.key = r.kv.key ∧ w.kv.vc = r.kv.vc}

203  co ≜ TC(so ∪ rf)  causality order

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

216  CM ≜  causal memory consistency model; see Ahamad@DC'1995
217     LET ops ≜ UNION {Range(L[c]) : c ∈ Client}
218         wops ≜ {op ∈ ops : op.type  = "W"}
219     IN  ∀ c ∈ Client :
220            ∃ sc ∈ PermutationsOf(L[c] ∘ SetToSeq(wops)) :
221               ∧ Valid(sc)  ClassCastException: LetInNode cannot be cast to class OpApplNode
222                ∧ Respect(SeqToRel(sc), co)

224  THEOREM Spec ⇒ □CM
225 └──────────────────────────────────────────────────────────────────────────┘
```