MODULE *cosmos_client*

Microsoft Azure Cosmos *DB* TLA+ specication for the five consistency levels the service offers. The spec focuses on the consistency guarantees Cosmos *DB* provides to the clients, without the details of the protocol implementation.

EXTENDS *Naturals*, *Integers*, *Reals*, *Sequences*, *FiniteSets*, *TLC*

Number of regions

CONSTANT *NumRegions*
CONSTANT *NumWriteRegions*

ASSUME $NumRegions \in Nat$
ASSUME $NumWriteRegions \geq 1 \land NumWriteRegions \leq NumRegions$

Number of clients per region for modeling

CONSTANT *NumClientsPerRegion*

ASSUME $NumClientsPerRegion \in Nat$

MaxNumOp max number of operations from client

CONSTANT *MaxNumOp*

Consistency level
(1) strong (*Linearizability*)
(2) bounded (Bounded *Staleness*)
(3) session
(4) prefix (Consistent *Prefix*)
(5) eventual

CONSTANT *Consistency*

ASSUME $Consistency \in \{\text{“strong”}, \text{“bounded\_staleness”}, \text{“session”}, \text{“consistent\_prefix”}, \text{“eventual”}\}$

The bounded version differences in *Bounded Staleness* consistency

CONSTANT K

ASSUME $K \in Nat$

All regions in topology

$Regions \triangleq 1 \,..\, NumRegions$

All writable regions in topology

$WriteRegions \triangleq 1 \,..\, NumWriteRegions$

All clients with local region

$Clients \triangleq \{\langle r, j\rangle : r \in Regions, j \in 1 \,..\, NumClientsPerRegion\}$

All possible operations in history

$Operations \triangleq [type : \{\text{“write”}\}, data : Nat, region : WriteRegions, client : Clients]$
$\quad \cup [type : \{\text{“read”}\}, data : Nat, region : Regions, client : Clients]$

--algorithm *cosmos_client*

```
{
    variables  Max staleness. Strong is a special case of bounded with K = 1
               Bound = CASE Consistency = "strong" → 1
                         □  Consistency = "bounded_staleness" → K
                         □  Consistency = "session" → MaxNumOp
                         □  Consistency = "consistent_prefix" → MaxNumOp
                         □  Consistency = "eventual" → MaxNumOp ;

                Client operation history
               History = ⟨⟩ ;

                Latest data value in each region
               Data = [r ∈ Regions ↦ 0] ;

                Tentative log in each region
               Database = [r ∈ Regions ↦ ⟨⟩] ;

                Value used by clients
               value = 0 ;

    define
    {
        Removing duplicates from a sequence:
      RECURSIVE RemDupRec(_, _)
      RemDupRec(es, seen) ≜ IF es = ⟨⟩ THEN ⟨⟩
                              ELSE IF es[1] ∈ seen THEN RemDupRec(Tail(es), seen)
                              ELSE ⟨es[1]⟩ ∘ RemDupRec(Tail(es), seen ∪ {es[1]})

      RemoveDuplicates(es) ≜ RemDupRec(es, {})

      SetMax(S) ≜ IF S = {} THEN −1
                  ELSE CHOOSE i ∈ S : ∀ j ∈ S : i ≥ j

      SeqToSet(s) ≜ {s[i] : i ∈ DOMAIN s}

      Last(s) ≜ s[Len(s)]

      MaxLen(c) ≜ LET region ≜ CHOOSE i ∈ Regions : ∀ j ∈ Regions : Len(c[i]) ≥ Len(c[j])
                  IN  Len(c[region])

      MinLen(c) ≜ LET region ≜ CHOOSE i ∈ Regions : ∀ j ∈ Regions : Len(c[i]) ≤ Len(c[j])
                  IN  Len(c[region])
    }

    ------------------------- CLIENT ACTIONS -------------------------
```

```
113     Regular write at local region
114   macro write( v )
115   {
116       if ( self[1] ∈ WriteRegions )
117       {
118           when ∀ i, j ∈ Regions : Data[i] − Data[j] < Bound ;
119           Database[self[1]] := Append(@, v) ;
120           Data[self[1]] := v ;
121           History := Append(History, [type   ↦ "write",
122                                       data   ↦ v,
123                                       region ↦ self[1],
124                                       client ↦ self]) ;
125           session_token := v ;
126       }
127   }

129     Reads with consistency checks
130   macro read( )
131   {
132       We check session token for session consistency
133       when Consistency ≠ "session" ∨ Data[self[1]] ≥ session_token ;
134       We check global value for strong consistency
135       when Consistency ≠ "strong" ∨ ∀ i, j ∈ Regions : Data[i] = Data[j] ;
136       History := Append(History, [type   ↦ "read",
137                                   data   ↦ Data[self[1]],
138                                   region ↦ self[1],
139                                   client ↦ self]) ;
140       session_token := Data[self[1]] ;
141   }

143     ------------------------------------------------------------
144     ---------------------- REGION ACTIONS ----------------------
145     ------------------------------------------------------------

147     Asynchronously replicates from source region to destination region and merges data history
148   macro replicate( )
149   {
150       with ( s ∈ WriteRegions ; d ∈ Regions )
151       {
152           Database[d] := RemoveDuplicates(SortSeq(Database[d] ∘ Database[s], < )) ;
153           if ( Len(Database[d]) > 0 )
154           {
155               Data[d] := Last(Database[d]) ;
156           }
157       }
158   }
```

```
160         ------------------------------------------------------------
161         ---------------------- CLIENT PROCESSES ----------------------
162         ------------------------------------------------------------
163     fair process ( client ∈ Clients )
164     variable session_token = 0 ;
165     numOp = 0 ;
166     {
167         client_actions:
168         while ( numOp < MaxNumOp )
169         {
170             numOp := numOp + 1 ;
171             either
172             {
173                 write:
174                 value := value + 1 ;
175                 write(value) ;
176              }
177             or read: read() ;
178          }
179      }
181         ------------------------------------------------------------
182         ---------------------- SERVER PROCESSES ----------------------
183         ------------------------------------------------------------
184     fair process ( CosmosDB = ⟨0, 0⟩ )
185     {
186         database_action:
187         while ( TRUE )
188         {
189             replicate() ;
190          }
191      }

193  }

195  BEGIN TRANSLATION
196 VARIABLES Bound, History, Data, Database, value, pc

198  define statement
199 RECURSIVE RemDupRec(_, _)
200 RemDupRec(es, seen) ≜ IF es = ⟨⟩ THEN ⟨⟩
201                          ELSE IF es[1] ∈ seen THEN RemDupRec(Tail(es), seen)
202                          ELSE ⟨es[1]⟩ ∘ RemDupRec(Tail(es), seen ∪ {es[1]})

204 RemoveDuplicates(es) ≜ RemDupRec(es, {})

206 SetMax(S) ≜ IF S = {} THEN −1
```

```
                ELSE  CHOOSE i ∈ S : ∀ j ∈ S : i ≥ j

SeqToSet(s) ≜ {s[i] : i ∈ DOMAIN s}

Last(s) ≜ s[Len(s)]

MaxLen(c) ≜ LET region ≜ CHOOSE i ∈ Regions : ∀ j ∈ Regions : Len(c[i]) ≥ Len(c[j])
            IN  Len(c[region])

MinLen(c) ≜ LET region ≜ CHOOSE i ∈ Regions : ∀ j ∈ Regions : Len(c[i]) ≤ Len(c[j])
            IN  Len(c[region])

VARIABLES session_token, numOp

vars ≜ ⟨Bound, History, Data, Database, value, pc, session_token, numOp⟩

ProcSet ≜ (Clients) ∪ {⟨0, 0⟩}

Init ≜  Global variables
        ∧ Bound = (CASE Consistency = "strong" → 1
                     □  Consistency = "bounded_staleness" → K
                     □  Consistency = "session" → MaxNumOp
                     □  Consistency = "consistent_prefix" → MaxNumOp
                     □  Consistency = "eventual" → MaxNumOp)
        ∧ History = ⟨⟩
        ∧ Data = [r ∈ Regions ↦ 0]
        ∧ Database = [r ∈ Regions ↦ ⟨⟩]
        ∧ value = 0
        Process client
        ∧ session_token  = [self ∈ Clients ↦ 0]
        ∧ numOp = [self ∈ Clients ↦ 0]
        ∧ pc = [self ∈ ProcSet ↦ CASE self ∈ Clients → "client_actions"
                                    □  self = ⟨0, 0⟩ → "database_action"]

client_actions(self) ≜  ∧ pc[self] = "client_actions"
                        ∧ IF numOp[self] < MaxNumOp
                             THEN  ∧ numOp' = [numOp EXCEPT ![self] = numOp[self] + 1]
                                   ∧ ∨ ∧ pc' = [pc EXCEPT ![self]      = "write"]
                                     ∨ ∧ pc' = [pc EXCEPT ![self]      = "read"]
                             ELSE  ∧ pc' = [pc EXCEPT ![self] = "Done"]
                                   ∧ numOp' = numOp
                        ∧ UNCHANGED ⟨Bound, History, Data, Database, value,
                                     session_token⟩

write(self) ≜  ∧ pc[self] = "write"
               ∧ value' = value + 1
               ∧ IF self[1] ∈ WriteRegions
                    THEN  ∧ ∀ i, j ∈ Regions : Data[i] − Data[j]      < Bound
```

```
                                    ∧ Database' = [Database EXCEPT ![self[1]] = Append(@, value')]
                                    ∧ Data' = [Data EXCEPT ![self[1]] = value']
                                    ∧ History' = Append(History, [type  ↦ "write",
                                                                  data  ↦ value',
                                                                  region ↦ self[1],
                                                                  client  ↦ self])
                                    ∧ session_token' = [session_token EXCEPT ![self] = value']
                               ELSE ∧ TRUE
                                    ∧ UNCHANGED ⟨History, Data, Database,
                                                 session_token⟩
                          ∧ pc' = [pc EXCEPT ![self] = "client_actions"]
                          ∧ UNCHANGED ⟨Bound, numOp⟩

read(self) ≜ ∧ pc[self] = "read"
             ∧ Consistency ≠ "session" ∨ Data[self[1]] ≥ session_token[self]
             ∧ Consistency ≠ "strong" ∨ ∀ i, j ∈ Regions : Data[i] = Data[j]
             ∧ History' = Append(History, [type  ↦ "read",
                                           data  ↦ Data[self[1]],
                                           region ↦ self[1],
                                           client  ↦ self])
             ∧ session_token' = [session_token EXCEPT ![self] = Data[self[1]]]
             ∧ pc' = [pc EXCEPT ![self] = "client_actions"]
             ∧ UNCHANGED ⟨Bound, Data, Database, value, numOp⟩

client(self) ≜ client_actions(self) ∨ write(self) ∨ read(self)

database_action ≜ ∧ pc[⟨0, 0⟩] = "database_action"
                  ∧ ∃ s ∈ WriteRegions :
                      ∃ d ∈ Regions :
                        ∧ Database' = [Database EXCEPT ![d] = RemoveDuplicates(SortSeq(Database[d] ∘
                        ∧ IF Len(Database'[d]) > 0
                             THEN ∧ Data' = [Data EXCEPT ![d] = Last(Database'[d])]
                             ELSE ∧ TRUE
                                  ∧ Data' = Data
                  ∧ pc' = [pc EXCEPT ![⟨0, 0⟩] = "database_action"]
                  ∧ UNCHANGED ⟨Bound, History, value, session_token, numOp⟩

CosmosDB ≜ database_action

Next ≜ CosmosDB
       ∨ (∃ self ∈ Clients : client(self))

Spec ≜ ∧ Init ∧ □[Next]_vars
       ∧ ∀ self ∈ Clients : WF_vars(client(self))
       ∧ WF_vars(CosmosDB)

END TRANSLATION
```

```
304 ├──────────────────────────────────────────────────────────────────────────────┤

306  enable these invariants in model checker

308  Check elements in History are type of Opertion
309  TypeOK ≜ {History[i] : i ∈ DOMAIN History} ⊆ Operations

311  Read value in any regional database history
312  AnyReadPerRegion(r) ≜ ∀ i ∈ DOMAIN History : ∧ History[i].type = "read"
313                                                ∧ History[i].region = r
314                                                ⇒ History[i].data ∈ SeqToSet(Database[r]) ∪ {0}

316  Operation in history h is monitonic
317  Monotonic(h) ≜ ∀ i, j ∈ DOMAIN h : i ≤ j ⇒ h[i].data ≤ h[j].data

319  Reads in region r are monotonic
320  MonotonicReadPerRegion(r) ≜ LET reads ≜ [i ∈ {j ∈ DOMAIN History : ∧ History[j].type = "read"
321                                                                    ∧ History[j].region = r}
322                                                          ↦ History[i]]
323                              IN  Monotonic(reads)

325  Reads from client c are monotonic
326  MonotonicReadPerClient(c) ≜ LET reads ≜ [i ∈ {j ∈ DOMAIN History : ∧ History[j].type = "read"
327                                                                    ∧ History[j].client = c}
328                                                          ↦ History[i]]
329                              IN  Monotonic(reads)

331  MonotonicWritePerRegion(r) ≜ LET writes ≜ [i ∈ {j ∈ DOMAIN History : ∧ History[j].type = "write"
332                                                                     ∧ History[j].region = r}
333                                                           ↦ History[i]]
334                               IN  Monotonic(writes)

336  Clients read their own writes
337  ReadYourWrite ≜ ∀ i, j ∈ DOMAIN History : ∧ i < j
338                                            ∧ History[i].type = "write"
339                                            ∧ History[j].type = "read"
340                                            ∧ History[i].client = History[j].client
341                                            ⇒ History[j].data ≥ History[i].data

343  Read the latest writes
344  ReadAfterWrite ≜ ∀ i, j ∈ DOMAIN History : ∧ i < j
345                                             ∧ History[i].type  = "write"
346                                             ∧ History[j].type  = "read"
347                                             ⇒ History[j].data ≥ History[i].data

349  Linearizability ≜ ∀ i, j ∈ DOMAIN History : ∧ i < j
350                                              ⇒ History[j].data ≥ History[i].data
```

```
LastOperation(c) ≜ LET i ≜ SetMax({j ∈ DOMAIN History : History[j].client = c})
                   IN  IF i > 0 THEN History[i] ELSE ⟨⟩

BoundedStaleness ≜ ∧ ∀ i, j ∈ Regions : Data[i] − Data[j] ≤ K
                   ∧ ∀ r ∈ Regions : MonotonicReadPerRegion(r)
                   ∧ ReadYourWrite

ConsistentPrefix ≜ ∀ r ∈ Regions : ∧ MonotonicWritePerRegion(r)
                                   ∧ AnyReadPerRegion(r)

Strong ≜ ∧ Linearizability
         ∧ Monotonic(History)
         ∧ ReadAfterWrite

Session ≜ ∧ ∀ c ∈ Clients : MonotonicReadPerClient(c)
          ∧ ReadYourWrite

Eventual ≜ ∀ i ∈ DOMAIN History :
           LET r ≜ History[i].region
           IN  History[i].data ∈ {Database[r][j] : j ∈ DOMAIN Database[r]} ∪ {0}

Invariant ≜ ∧ TypeOK
            ∧ CASE Consistency = "strong" → Strong
                 □ Consistency = "bounded_staleness" → BoundedStaleness
                 □ Consistency = "session" → Session
                 □ Consistency = "consistent_prefix" → ConsistentPrefix
                 □ Consistency = "eventual" → Eventual

Liveness ≜ ◇□(∀ i, j ∈ Regions : Database[i] = Database[j])
```

* Authored by Cosmos *DB*