Causal consistency in Geo-replicated Systems

Deepthi Akkooorath
AG SoftTech
Geo-Replicated Distributed Application
Geo-Replicated Distributed Application

1. Create meeting

DC1

DC2

DC3
Geo-Replicated Distributed Application

1. Create meeting

DC1

2. Create meeting

DC2

DC3
Geo-Replicated Distributed Application

1. Create meeting
2. Create meeting
3. See meeting
   Delete meeting
Geo-Replicated Distributed Application

1. Create meeting
2. Create meeting
3. See meeting
   Delete meeting
4. Delete meeting
Geo-Replicated Distributed Application

1. Create meeting

2. Create meeting

3. See meeting
   Delete meeting

4. Delete meeting

5. Create meeting
Geo-Replicated Distributed Application

1. Create meeting

2. Create meeting

3. See meeting
   Delete meeting

4. Delete meeting

5. Create meeting

6. Delete meeting
Lamport's Timestamps

- Total order of events satisfying Happened-before relation
- Each process has a Logical clock
- A process increments its clock for each event
- Sends clock with each message it sends
- On receiving a message
  - Sets clock = max(own clock, received clock)
Lamport's Timestamps

- **Total order** of events satisfying Happened-before relation
- Each process has a Logical clock
- A process increments its clock for each event
- Sends clock with each message it sends
- On receiving a message
  - Sets clock = \( \max(\text{own clock}, \text{received clock}) \)
Vectorclocks

- Similar to Lamport's timestamp
- Partial order and detect causality violations
- A system on N processes
  - Vectorclock = array of N logical clocks
  - Each process has a vectorclock
  - Increment its own logical clock for each event
  - On receiving a message
    - Set each entry in vc to be max(local entry, corresponding entry in received vc)
Detect Causality violation using Vectorclocks

DC1
[0,0,0]

DC2
[0,0,0]

DC3
[0,0,0]
Detect Causality violation using Vectorclocks

DC1

[0,0,0]  
Create meeting

DC2

[0,0,0]

DC3

[0,0,0]
Detect Causality violation using Vectorclocks

DC1
[0,0,0]
[1,0,0]
Create meeting

DC2
[0,0,0]

DC3
[0,0,0]
Detect Causality violation using Vectorclocks

Create meeting
Detect Causality violation using Vectorclocks

Create meeting

DC1
[0,0,0]
[1,0,0]

DC2
[0,0,0]
[2,1,0]

DC3
[0,0,0]
Detect Causality violation using Vectorclocks

- DC1
  - [0,0,0]
  - [1,0,0]
  - Create meeting

- DC2
  - [0,0,0]
  - [2,1,0]
  - Delete meeting

- DC3
  - [0,0,0]
Detect Causality violation using Vectorclocks

DC1

[0,0,0]
[1,0,0]

Create meeting

DC2

[0,0,0]
[2,1,0]
[2,2,0]

Delete meeting

DC3

[0,0,0]
Detect Causality violation using Vectorclocks

DC1
[0,0,0]
[1,0,0]
Create meeting

DC2
[2,1,0]
[2,2,0]
Delete meeting

DC3
[0,0,0]
Detect Causality violation using Vectorclocks

- Create meeting at DC1: [1,0,0]
- Delete meeting at DC2: [2,1,0] at [2,2,0]
- Delete meeting at DC3: [2,2,0] at [2,3,1]
Detect Causality violation using Vectorclocks

DC1
[0,0,0]
[1,0,0]
Create meeting

DC2
[2,1,0]
[2,2,0]
Delete meeting

DC3
[0,0,0]

[1,0,0]

[2,3,1]
Detect Causality violation using Vectorclocks

DC1
[0,0,0]
[1,0,0]

Create meeting

DC2
[0,0,0]
[2,1,0]
[2,2,0]

Delete meeting

DC3
[0,0,0]
[2,2,0]
[2,3,1]
[1,0,0]
[2,3,2]
Detect Causality violation using Vectorclocks

Create meeting

Delete meeting

DC1

DC2

DC3

[0,0,0]

[1,0,0]

[2,1,0]

[2,2,0]

[2,2,0]

[2,3,2]

[2,3,1]

[1,0,0]
Version Vectors

- Similar to vector clocks
- Partial order among replicas of an object
- Several mechanisms to keep size of version vector small
  - Bounded Version Vectors
  - Dotted Version Vectors
- Causality across objects cannot be tracked
Partitioned and Geo-Replicated Distributed System
Partitioned and Geo-Replicated Distributed System

1. remove photos
Partitioned and Geo-Replicated Distributed System

1. remove photos
2. addFriend(Bob)
Partitioned and Geo-Replicated Distributed System

1. remove photos
2. addFriend(Bob)
3. addFriend(Bob)
Partitioned and Geo-Replicated Distributed System

1. remove photos
2. addFriend(Bob)
3. addFriend(Bob)
4. See photos of Alice
Partitioned and Geo-Replicated Distributed System

1. remove photos
2. addFriend(Bob)
3. addFriend(Bob)
4. See photos of Alice
5. removePhotos
Orbe: Causal Consistency with Dependency Matrix

<table>
<thead>
<tr>
<th>Clock</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Orbe: Causal Consistency with Dependency Matrix

- Dependency matrices to track causality
- Client updates its DM when ever it reads a new version

<table>
<thead>
<tr>
<th>Clock</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Client has seen first 2 updates at replica 2 of partition 1
Orbe: Causal Consistency with Dependency Matrix

- Each Partition has its own version vector - VV

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1/R1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- P1 at DC1 has
  - 1 local update
  - 2 updates from R2
  - 1 update from R3
Orbe: Causal Consistency with Dependency Matrix

- Client send `put(k,v,DM)` to partition P1 at DC1
- P1 at DC1
  - Increment its own VV[R1]
  - Ts = VV[R1]
  - New entry U<k, v, 2, DM, R1>
  - Replicate U to P1 at DC2 and DC3
- On receiving U<k, v, ts, DM, replicaaid> at Pn
  - Check VV >= DM[n]
  - Check if causality is satisfied at other partitions
  - Update VV[replicaaid] = ts
Total order in a partitioned system

- **Snapshot isolation**
  - Reads a consistent snapshot
- **Consistent Snapshot**
  - Includes all updates committed before snapshot time
- **Transactions commit in total order**
- **Snapshot identified by its commit time**
- **Update A is causally before B if A.commit-time < B.commit-time**
Clock SI – Snapshot Isolation using physical clocks

- Loosely synchronized clocks
- No centralized time-stamp generator
- Distributed protocol
- **Snapshot-time**
  - Time when transaction begins
  - Reads return values committed on or before this time
- **Commit-time decided by transaction coordinator and partitions involved in transaction**
ClockSI – Commit Protocol

Diagram showing the relationship between Txn Coordinator and Partitions.
ClockSI – Commit Protocol

- T.snapshottime = Localclock = 8
- Send prepare to partitions
- Commit-time = max(11,9,10)
- Commit to partitions
ClockSI – Commit Protocol

- $T.$snapshottime = Localclock = 8
- Send prepare to partitions
- Commit-time = max(11,9,10)
- Commit to partitions
ClockSI – Commit Protocol

- **Txn Coordinator**
  - T.snapshottime = Localclock = 8
  - Send prepare to partitions
  - Commit-time = max(11,9,10)
  - Commit to partitions

- **Partitions**
  - Receive Prepare
  - Localclock = 11
  - Reply 11
  - Commit-time = 11

  - Receive Prepare
  - Localclock = 9
  - Reply 9
  - Commit-time = 11

  - Receive Prepare
  - Localclock = 10
  - Reply 10
  - Commit-time = 11
ClockSI – Commit Protocol

Txn Coordinator

- T.snapshottime = Localclock = 8
- Send prepare to partitions
- Commit-time = max(11, 9, 10)
- Commit to partitions

Partitions

- Receive Prepare
  - Localclock = 11
  - Reply 11
  - Commit-time = 11

- Receive Prepare
  - Localclock = 9
  - Reply 9
  - Commit-time = 11

- Receive Prepare
  - Localclock = 10
  - Reply 10
  - Commit-time = 11
ClockSI – Commit Protocol

Txn Coordinator
- \( T.\text{snapshottime} = \text{Localclock} = 8 \)
- Send prepare to partitions
- Commit-time = max(11,9,10)
- Commit to partitions

Partitions
- Receive Prepare
  - Localclock = 11
  - Reply 11
  - Commit-time = 11

- Receive Prepare
  - Localclock = 9
  - Reply 9
  - Commit-time = 11

- Receive Prepare
  - Localclock = 10
  - Reply 10
  - Commit-time = 11
ClockSI – Commit Protocol

- **Txn Coordinator**
  - $T.\text{snapshottime} = \text{Localclock} = 8$
  - Send prepare to partitions
  - Commit-time = max(11,9,10)
  - Commit to partitions

- **Partitions**
  - Receive Prepare
  - Localclock = 11
  - Reply 11
  - Commit-time = 11

  - Receive Prepare
  - Localclock = 9
  - Reply 9
  - Commit-time = 11

  - Receive Prepare
  - Localclock = 10
  - Reply 10
  - Commit-time = 11
ClockSI – Commit Protocol

- **Txn Coordinator**
  - T.snapshottime = Localclock = 8
  - Send prepare to partitions
  - Commit-time = max(11, 9, 10)
  - Commit to partitions

- Receive Prepare
  - Localclock = 11
  - Reply 11
  - Commit-time = 11

- Receive Prepare
  - Localclock = 9
  - Reply 9
  - Commit-time = 11

- Receive Prepare
  - Localclock = 10
  - Reply 10
  - Commit-time = 11
Clock SI – Read protocol

Read(Transaction T, dataitem Obj)

- Wait if T.snapshottime > localclock
- If any pending Transaction T' with possible commit-time < T'.snapshottime
  - wait until T' is committed
- Return latest snapshot before snapshot-time
Extended ClockSI: Partitioned and Replicated System

- Vectorclock per partition

  - P1 at DC1 has seen all updates from DC2 before time 9

- Snapshot-time is Vectorclock of coordinator at the time when transaction begins

- Updates in a transaction depends on Snapshot which it reads from

- Snapshot-time encodes causal dependency
Extended ClockSI: Replication

- P1 at DC1 sends updates to P1 at DC2 in *Commit-time order*
- Send snapshot-time and commit-time with every update
- On receiving an update $U_{<DC, \text{Commit-time, Snapshot-time}>}$ from a partition
  - Apply $U$ if local vectorclock > Snapshot-time
  - Set vectorclock[DC] = Commit-time
Extended ClockSI: Read

• Upon receiving a read request in a partition
  - Wait until local vectorclock $\geq$ snapshot-time
  - Return latest value before snapshot-time

• Causality metadata = $O(N)$
• No communication between partitions
Social Network Application
Social Network Application

1. remove photos
   $S=[0,0], C[2,0]$
Social Network Application

1. remove photos
   S=[0,0], C[2,0]

2. addFriend(Bob)
   S[3,0], C[4,0]
Social Network Application

1. remove photos
   $S=[0,0], C[2,0]$

2. addFriend(Bob)
   $S[3,0], C[4,0]$

3. addFriend(Bob)
   $VV=[4,0]$
Social Network Application

1. remove photos
   $S = [0,0], C[2,0]$

2. addFriend(Bob)
   $S[3,0], C[4,0]$

3. addFriend(Bob)

4. Friends? Yes!!
   $S = [4,0]$

$VV = [4,0]$
1. remove photos
   \[ S = [0,0], C[2,0] \]

2. addFriend(Bob)
   \[ S[3,0], C[4,0] \]

3. addFriend(Bob)

4. Friends? Yes !!
   \[ S = [4,0] \]

SeePhotos? \[ S = [4,0] \]

\[ VV = [4,0] \]
Social Network Application

1. remove photos
   $S = [0, 0], C[2, 0]$

2. addFriend(Bob)
   $S[3, 0], C[4, 0]$

3. addFriend(Bob)

4. Friends? Yes !!
   $S = [4, 0]$

SeePhotos? $S = [4, 0]$

$VV = [4, 0]$

$VV = [0, 0]$
Social Network Application

1. remove photos
   \[ S = [0,0], C[2,0] \]

2. addFriend(Bob)
   \[ S[3,0], C[4,0] \]

3. addFriend(Bob)

4. Friends? Yes !
   \[ S = [4,0] \]

5. removePhotos

\[ VV = [4,0] \]

\[ VV = [2,0] \]

SeePhotos? \[ S = [4,0] \]
Social Network Application

1. remove photos
   $S = [0,0], C[2,0]$

2. addFriend(Bob)
   $S [3,0], C[4,0]$

3. addFriend(Bob)

4. Friends? Yes!!
   $S = [4,0]$

5. removePhotos

SeePhotos? $S = [4,0]$

No photos

VV = [2,0]
Conclusion

- Total ordering using Lamport's timestamp
- Causality tracking using Vectorclocks
- Explicit causality tracking
  - Orbe using dependency matrix
  - ClockSI using physical clock and dependency vector
Reference


