Causal Consistency

Distributed Systems Spring 2020

Lecture 14

Agenda

1. Last time: Consistency models and Sequential Consistency
2. Causal Consistency
3. Eventual Consistency
4. Client-centric Consistency Models
5. Next class: Implementation aspects of Consistency protocols
6. Next week: CRDT's!

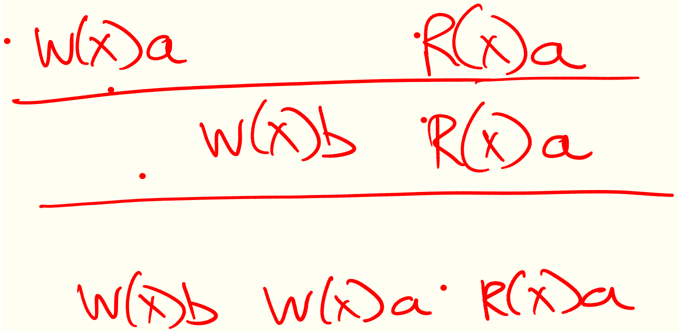
Sequential consistency

Definition

The result of any execution is the same as if the operations of all processes were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program.

All writes are seen in the same order by all processes.

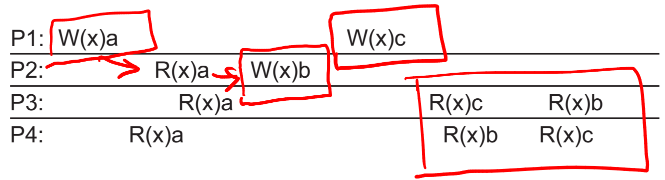
→ Total Order Broadcast



Causal consistency

Definition

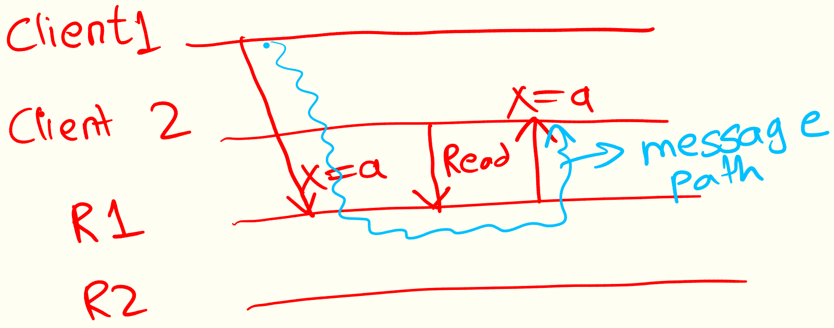
Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order by different processes.



W(x)a < W(x)b

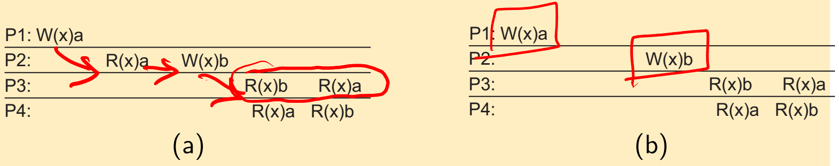
Not sequentially consisteny.

P2:W(x)b and P1:W(x)c are concurrent. P3 and P4 can thus see different orderings.



Causal Consistency Examples

(a) A violation of a causally-consistent store. (b) A correct sequence of events in a causally-consistent store

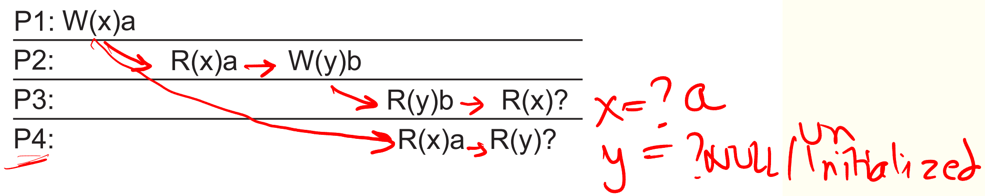


w(x)a < w(x)b < R(x)b

First example: P1:W(x)a $\rightarrow$ P2:R(x)a $\rightarrow$ Wx(b). Thus the two writes are causally related and must take effect in same order.

Second example: writes are not causally related (no interleaved read), and thus can be seen in any order.

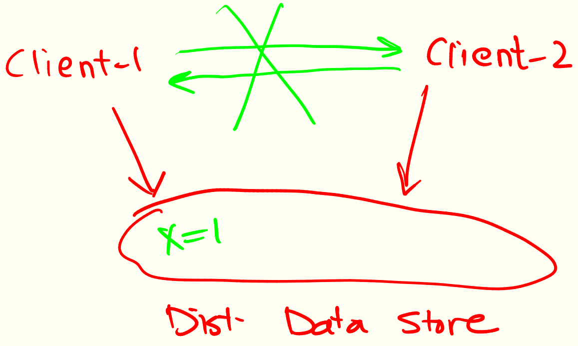
More Causal Consistency Examples



- What should the reads return?
- P3 R(x): a
- Acceptable for P4 to return NULL

Causal Consistency Caveats

- Similar in principle to causal-order broadcast discussed earlier
- Dont want to see a reply before original post
- In causal-order broadcast, a process waits if it receives a message from the "future", based on its vector clock timestamp.
- Same principle can be used to implement causal consistency
- Reads are causally related to writes.
- Out of band causality is not captured
  - Cant "phone" a friend and coordinate
  - Fails to capture causality of writes
  - "When you see x=1, write y=1"



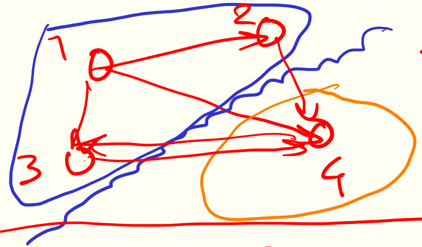
Implementing Causal Consistency

Causal consistency the strongest we can have in presence of partitions

P1: W(x)a				
P2:	R(x)a	W(y)b		
P3:			R(y)b	R(x)?
P4:			R(x)a	R(y)?

- Need to keep track of causal histories
- P3 needs to know about $W(x)a \rightarrow W(y)b$
- Need to keep a dependency graph of operations
- Similar to causal order broadcast
- When reading from a replica, "wait" until replica has applied all causally preceeding writes
- For performance, want to *lazily* propagate writes
- Note: Local-read sequential consistency algorithm has *eager* propagation.

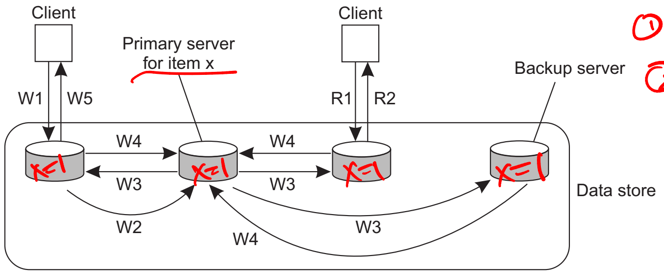
In Total Order Broadcast, what if some process does not reply?
↳ Network partition



If 4 died/failed, then you get this network partition

(1) Total Order Broadcast is NOT necessary for all writes.

Recall Primary-based Replication



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

① Local Reads
② Writes →
Total O[illegible]...

Challenge: Improve Performance

- Lazy propagation of writes (in the background).

Primary-based Implementation

- All writes go through a primary. (For simplicity, assume 1 primary for all objects)
- Writes are thus naturally causally ordered

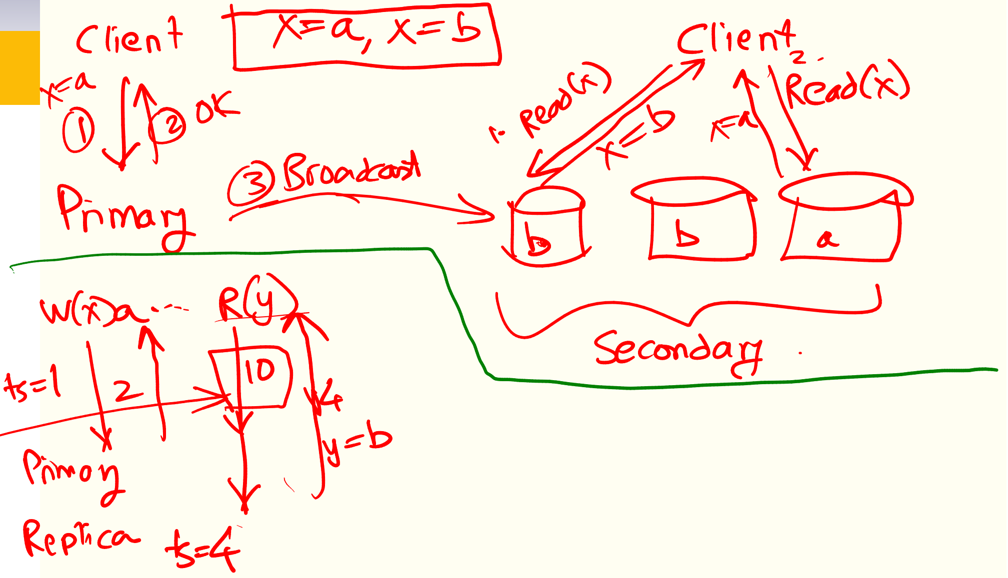
Central Problem

If a client issues reads to two different replicas, how to ensure that the reads are causally ordered?

All read and write operations are logical-clock timestamped:

1. Write operations assigned monotonically increasing timestamps by primary
2. Before a read, compute minimum acceptable timestamp.
   - Max ts across reads over all keys, and writes for that key
3. Each replica maintains M_r: max timestamp of all writes received by that replica
4. Read returns from replica only when $M_r >$ read timestamp

Doug Terry et.al. “Consistency-Based Service Level Agreements for Cloud Storage”



Eventual Consistency

- Concurrent updates are rare
  - Mostly: read-write conflicts
  - Examples: Web-caches, CDN's, DNS

Eventual Consistency

If no updates take place for a long time, all replicas *eventually* become consistent (have the same data stored)

- Easy to implement
- In practice, write-write conflicts handled through some form of leader election
- Inconsistency windows often small ($<$500 ms)

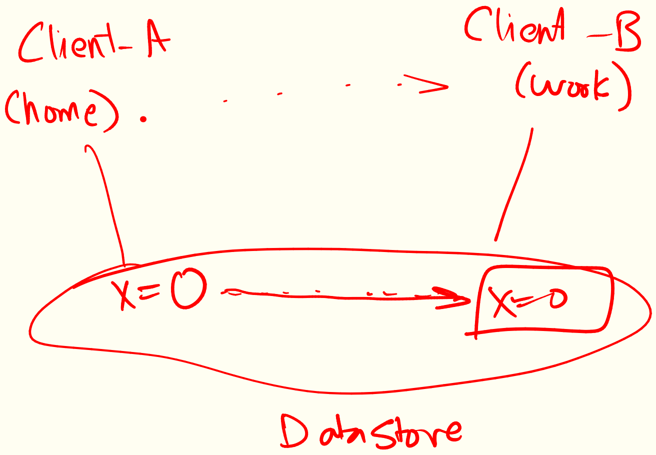
Consistency for mobile users

Example

Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database.

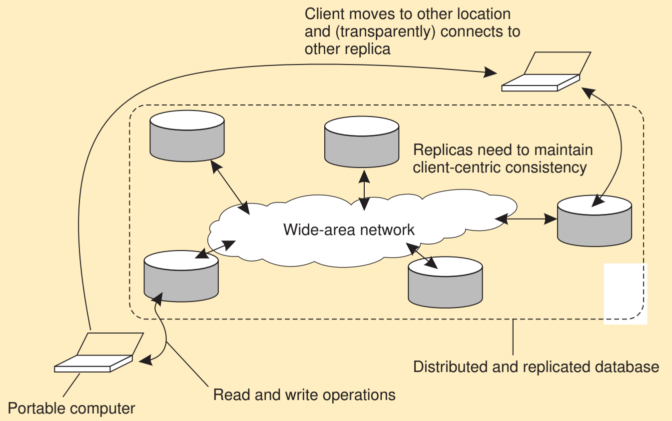
- At location A you access the database doing reads and updates.
- At location B you continue your work, but unless you access the same server as the one at location A, you may detect inconsistencies:
  - your updates at A may not have yet been propagated to B
  - you may be reading newer entries than the ones available at A
  - your updates at B may eventually conflict with those at A

The only thing you really want is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, the database will appear to be consistent **to you**.



Basic architecture

The principle of a mobile user accessing different replicas of a distributed database



END