Distributed Systems
COMP 212

Lecture 23

Othon Michail

UNIVERSITY OF
LIVERPOOL

Consistency and Replication

Why Replicate Data?

- Enhance reliability
  - While at least one server has not crashed, the service can be supplied
  - Protection against corrupted data (the majority of the copies is expected to be correct)
- Improve performance
  - Increasing the #clients would overload a single server
    - e.g., several web servers can have the same DNS name and the servers are selected in turn to share the load
  - Placing copies of data in the proximity of processes using them
- Replication of read-only data is simple, but replication of changing data has overheads

More on Replication

- Replicas allow remote sites to continue working in the event of local failures
- Possible to protect against data corruption
- Replicas allow data to reside close to where it is used
- This directly supports the distributed systems' goal of enhanced scalability
- Even a large number of replicated "local" systems can improve performance
  - think of clusters

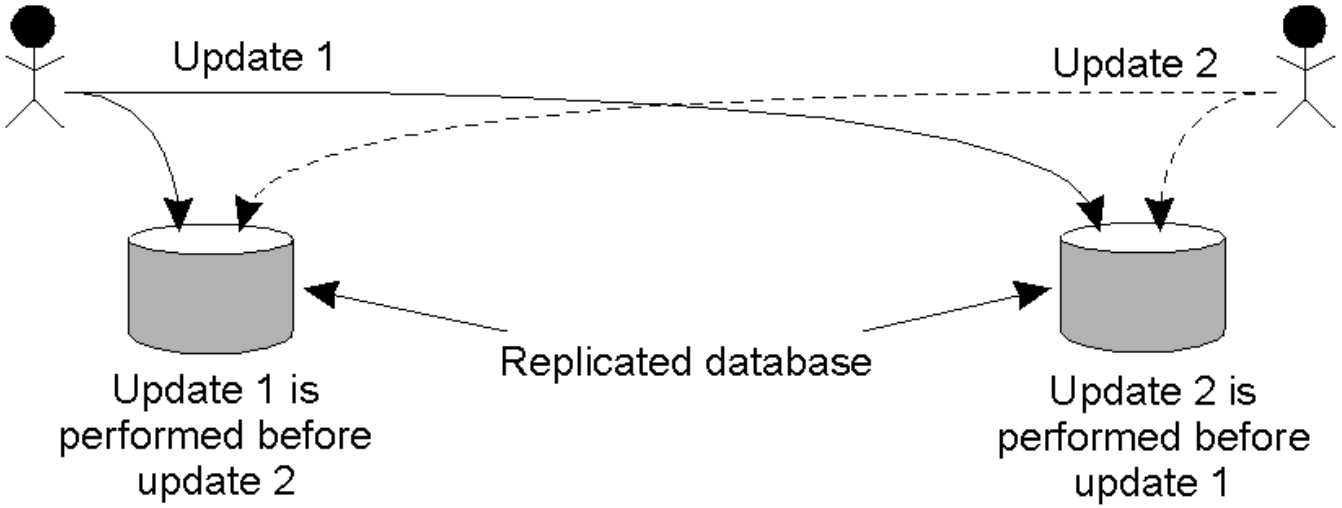
Replication and Scalability

- Replication is a widely-used scalability technique
  - think of web clients and web proxies
- When systems scale, the first problems to surface are those associated with performance
  - as the systems get bigger (e.g., more users), they often get slower
- Replicating the data and moving it closer to where it is needed helps to solve this scalability problem

So, what's the catch?

- There is a price to be paid when data is replicated
- It is not easy to keep all those replicas consistent
- Modifying a copy makes it different than the other copies
- Therefore, a modification should be propagated to all copies
  - The more transparent we want to make this the more we have to pay
  - There is a whole range from weak requirements to strong requirements

What Can Go Wrong



- Updating a replicated database: Update 1 adds £ 100 to an account, Update 2 calculates and adds 1% interest to the same account
- Due to network delays, the updates may come in different order!
- Inconsistent state

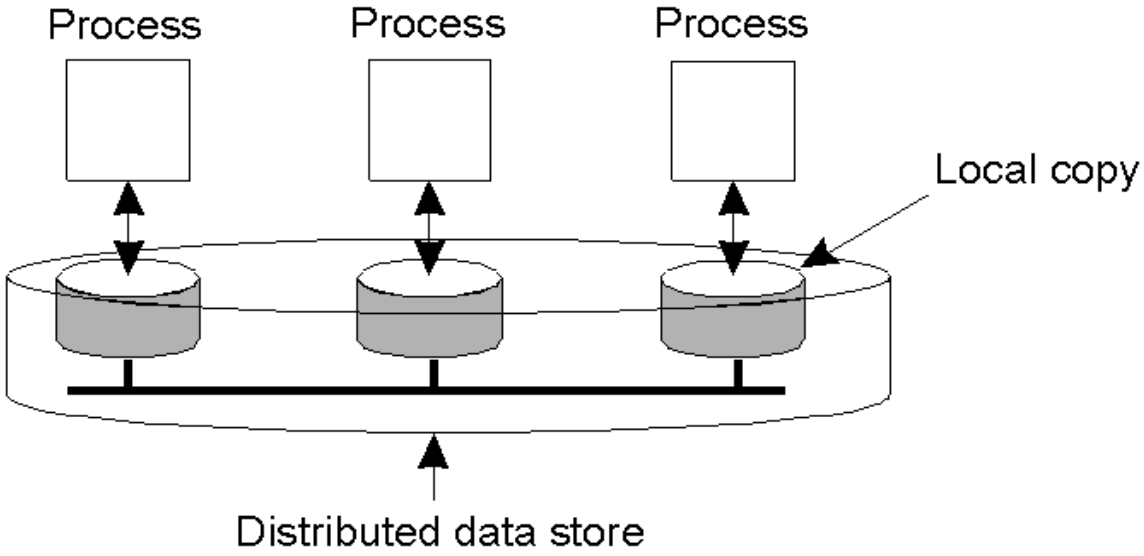
Requirements for replicated data

What is replication transparency?

- Replication transparency
  - clients see logical objects (not several physical copies)
  - they access one logical item and receive a single result
- Consistency
  - specified to suit the application
    - e.g., when a user of a diary disconnects, their local copy may be inconsistent with the others and will need to be reconciled when they connect again. But connected clients using different copies should get consistent results.

Data-Centric Consistency Models

- Data-store: distributed shared data; each process assumed to have access to a local copy of the entire store
- A data-store can be read from or written to by any process in a distributed system
- A local copy of the data-store (replica) can support "fast reads"
- However, a write to a local replica needs to be propagated to *all* remote replicas



- Various consistency models help to understand the various mechanisms used to achieve and enable this

What We Study Here?

- How to express what is happening in a system
  - Guaranteeing global ordering is a costly operation, downgrades scalability
  - Maybe, for some applications *total* ordering is an overkill?
    - Weaker consistency requirements
- How to make replicas consistent
  - Update protocols

What is a Consistency Model?

- A consistency model is a CONTRACT between a DS data-store and its processes
- If the processes agree to the rules, the data-store will perform properly and as advertised
- In the absence of a global clock it is difficult to define precisely which is the most recent value of a shared item
- Each model imposes different restrictions on what ordering of operations observed by the processes is acceptable. E.g.
  - Strict: Anything observed should be the real ordering (practically impossible to implement)
  - …
  - Weak: Wrong observations are always acceptable unless an explicit synchronisation takes place (very typical solution)

Consistency Models

No explicit synchronisation operations

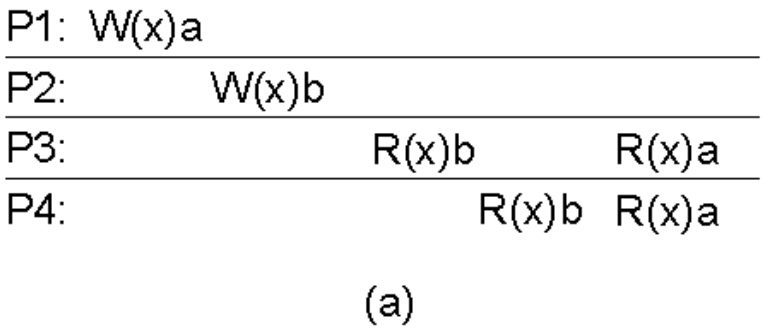
Strict	Absolute time ordering of all shared accesses matters
Linearisability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time.
Causal	All processes see causally-related shared accesses in the same order
FIFO	All processes see writes from each particular process in the order they were performed. Writes from different processes may not always be seen in their real order.

With explicit synchronisation operations

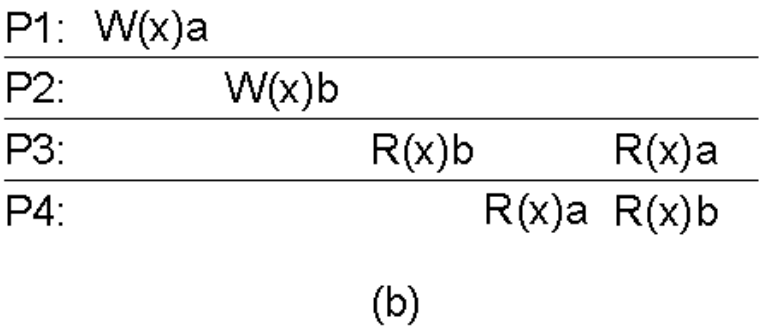
Weak	Shared data can be counted on to be consistent only after a synchronisation is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered

Example: Sequential Consistency

- *All* processes see the same interleaving set of operations, regardless of what that interleaving is



(a)



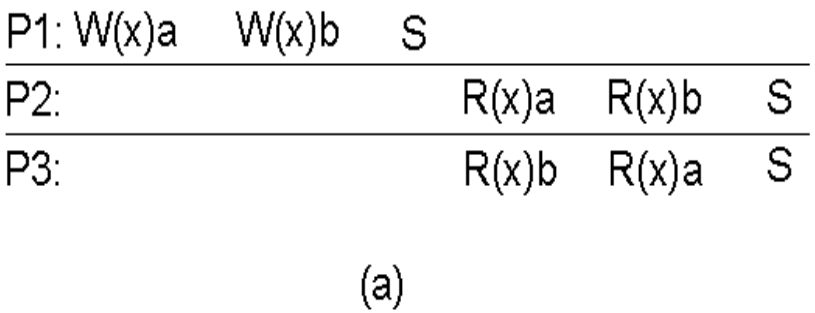
(b)

a) A sequentially consistent data-store
   - the "first" write occurred *after* the "second" on all replicas

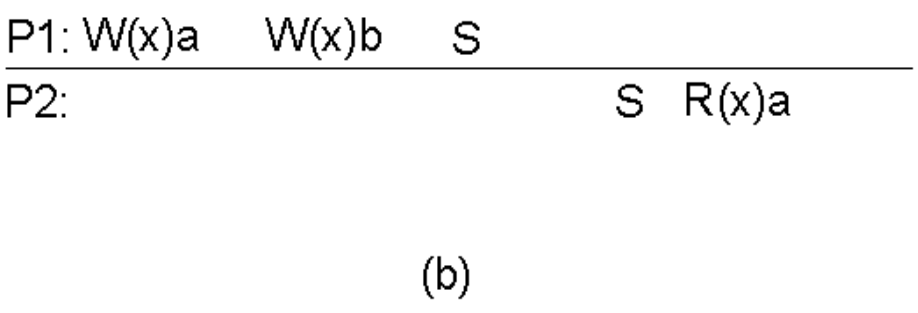
b) A data-store that is not sequentially consistent
   - it appears as if the writes have occurred in a non-sequential order, and this is NOT allowed

Example: Weak Consistency

- In some cases we do not need to synchronise on all data
  - Enforce consistency only on groups of operations
  - Not on individual data
  - Useful when isolated accesses are rare, with most coming in clusters (like transactions)



(a)



(b)

a) A valid sequence of events for weak consistency. This is because P2 and P3 have yet to synchronise, so there are no guarantees about the value in 'x'.

b) An invalid sequence for weak consistency. P2 has synchronised, so it cannot read 'a' from 'x' – it should be getting 'b'.

Client-Centric Consistency Models

- The previously studied consistency models concern themselves with maintaining a consistent (globally accessible) data-store in the presence of concurrent read/write operations

- Another class of a distributed data-store is that which is characterised by the lack of simultaneous updates. Here, the emphasis is more on maintaining a consistent view of things for the individual client process that is currently operating on the data-store.

More Client-Centric Consistency

- How fast should updates (writes) be made available to read-only processes?

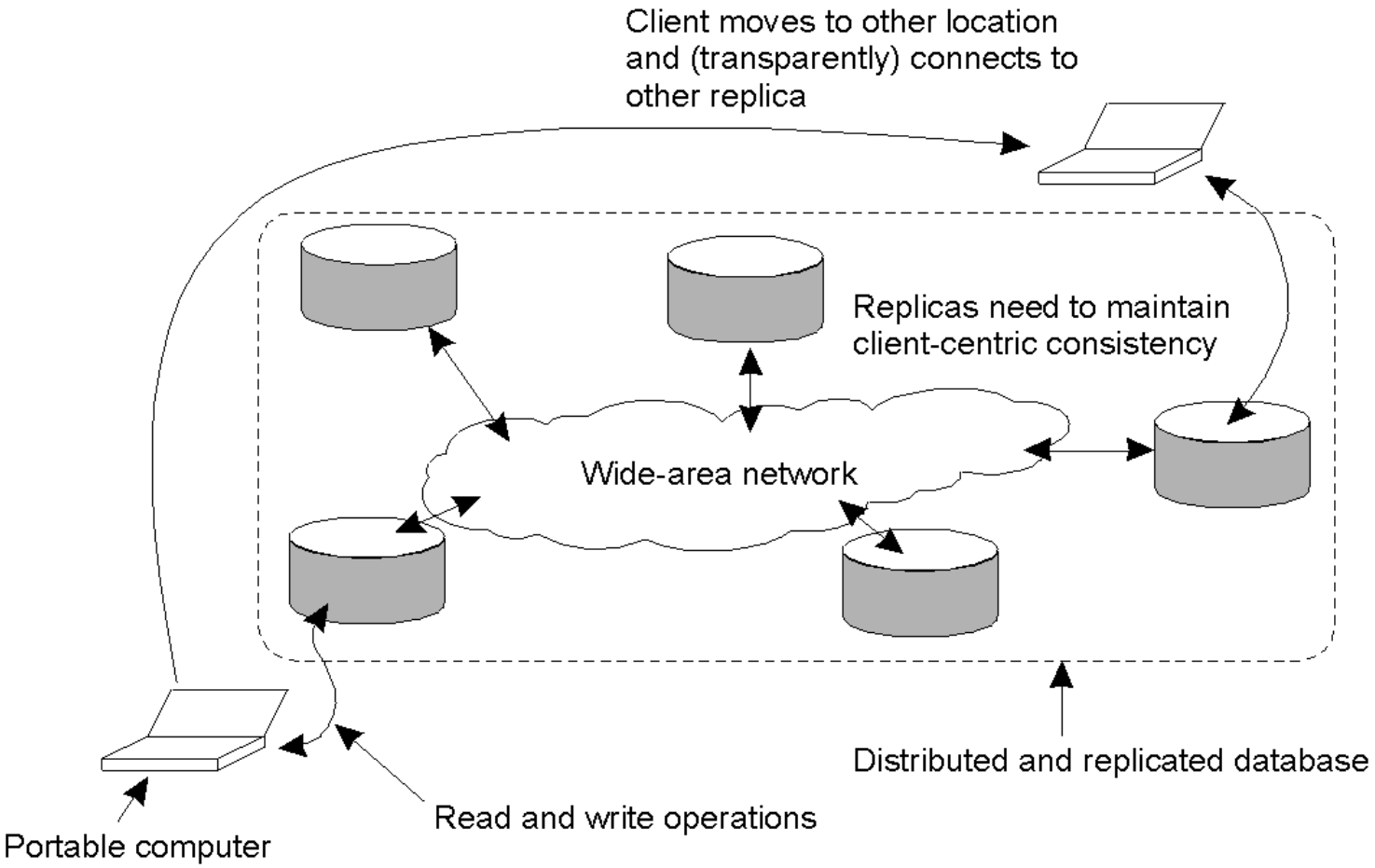
- Think of most database systems:
  - mainly *read*s
- Think of the DNS:
  - *write-write* conflicts do no occur.
- Think of WWW:
  - as with DNS, except that heavy use of client-side caching is present
  - even the return of stale pages is acceptable to most users

- These systems all exhibit a high degree of acceptable inconsistency … with the replicas gradually becoming consistent over time

Toward Eventual Consistency

- The only requirement is that all replicas will eventually be the same.
- All updates must be guaranteed to propagate to all replicas … eventually!
- This works well if every client always updates the same replica
- Things are a little difficult if the clients are mobile

Eventual Consistency: Mobile Problems



- The principle of a mobile user accessing different replicas of a distributed database
- When the system can guarantee that a single client sees accesses to the data-store in a consistent way, we then say that "client-centric consistency" holds

Consistency and Replication: Summary

- Reasons for replication:
  - improved performance, improved reliability
- Replication can lead to inconsistencies …
- How best can we propagate updates so that these inconsistencies are not noticed?
- With “best” meaning “without crippling performance”
- The proposed solutions resolve around the relaxation of any existing consistency constraints

Summary, continued

- Various consistency models have been proposed
- Data-Centric:
  - Strict, Sequential, Causal, FIFO concern themselves with individual reads/writes to data items
  - Weaker models introduce the notion of synchronisation variables
    - Release, Entry concern themselves with a group of reads/writes
- Client-Centric:
  - Concerned with maintaining consistency for a single clients' access to the distributed data-store
    - The Eventual Consistency model is an example