Consistency and Replication

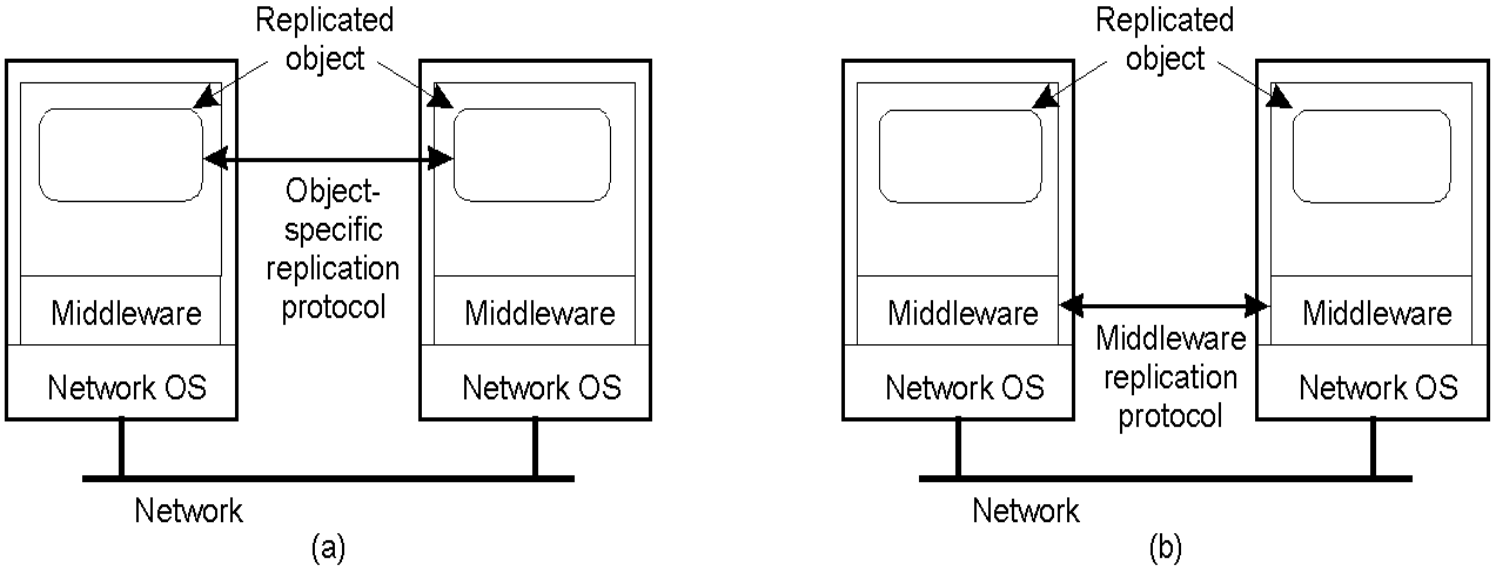
- Introduction
- Consistency models
  - Data-centric consistency models
  - Client-centric consistency models

Why replicate?

- Data replication: common technique in distributed systems
- Reliability
  - If one replica is unavailable or crashes, use another
  - Protect against corrupted data
- Performance
  - Scale with size of the distributed system (replicated web servers)
  - Scale in geographically distributed systems (web proxies)

- Key issue: need to maintain *consistency* of replicated data
  - If one copy is modified, others become inconsistent

Object Replication

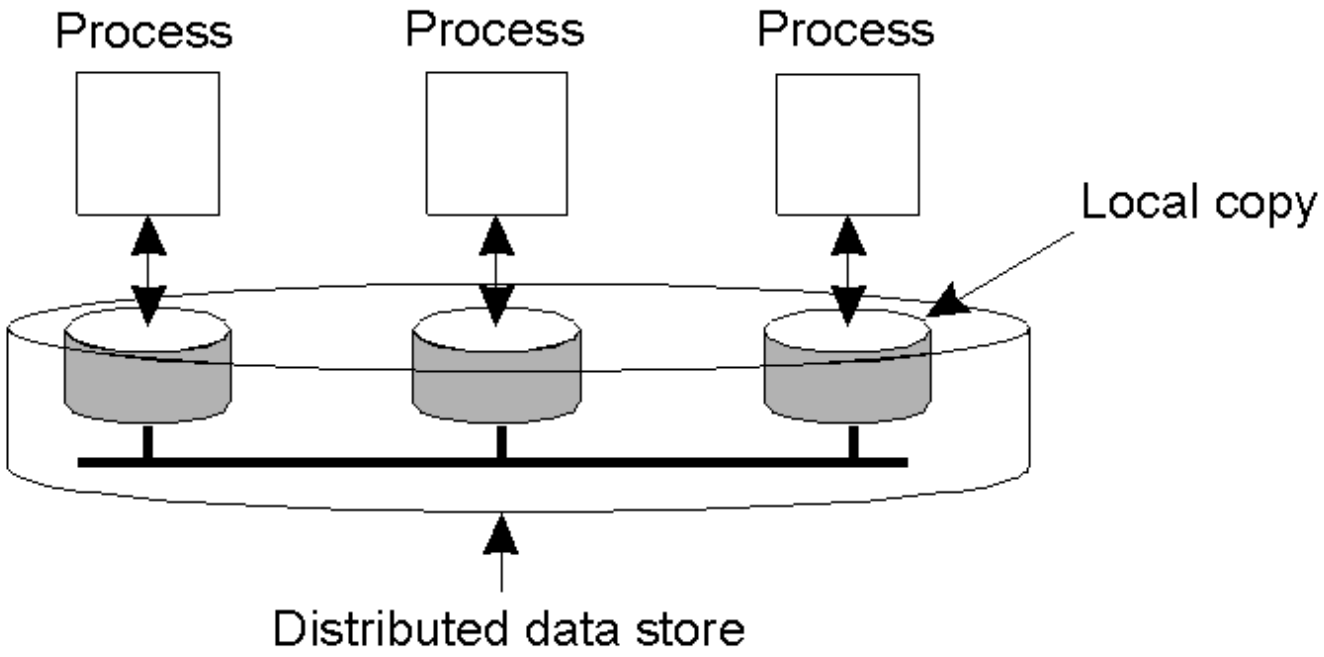


- Approach 1: application is responsible for replication
  - Application needs to handle consistency issues
- Approach 2: system (middleware) handles replication
  - Consistency issues are handled by the middleware
  - Simplifies application development but makes object-specific solutions harder

Replication and Scaling

- Replication and caching used for system scalability
- Multiple copies:
  - Improves performance by reducing access latency
  - But higher network overheads of maintaining consistency
  - Example: object is replicated *N* times
    - Read frequency *R*, write frequency *W*
    - If *R*<<*W*, high consistency overhead and wasted messages
    - Consistency maintenance is itself an issue
      - What semantics to provide?
      - Tight consistency requires globally synchronized clocks!
- Solution: loosen consistency requirements
  - Variety of consistency semantics possible

Data-Centric Consistency Models



- Consistency model (aka *consistency semantics or constraints*)
  - Contract between processes and the data store
    - If processes obey certain rules, data store will work correctly
  - All models attempt to return the results of the last write for a read operation
    - Differ in how "last" write is determined/defined

Continuous Consistency

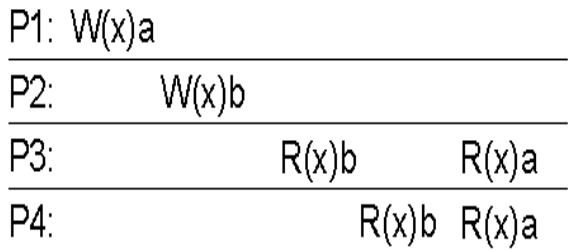
- The only way replication can work is by loosening consistency rules.
- There are different ways for applications to specify what inconsistencies they can tolerate
- Three ways to define inconsistencies
- Deviation in
  - numerical values between replicas(%)
  - staleness between replicas(e.g. weather reports)
  - deviation with respect to the ordering of update operations(tentative changes awaiting global agreement)

Strict Consistency

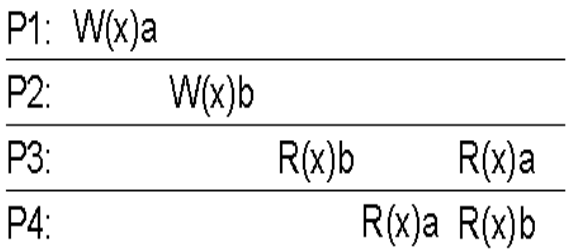
- Any read always returns the result of the most recent write
  - Implicitly assumes the presence of a global clock
  - A write is immediately visible to all processes
    - Difficult to achieve in real systems as network delays can be variable

Sequential Consistency

- Sequential consistency: weaker than strict consistency
  - Assumes all operations are executed in some sequential order and each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order
    - Nothing is said about "most recent write"



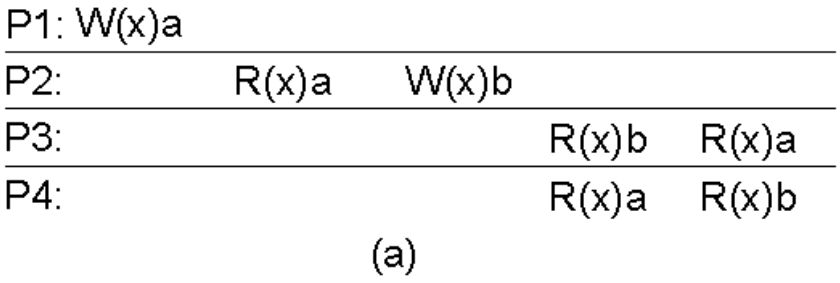
(a)



(b)

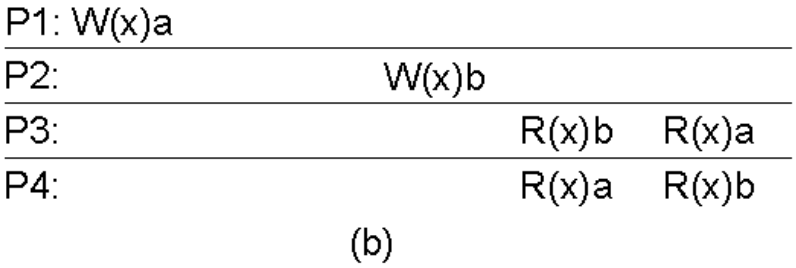
Causal consistency

- Causally related writes must be seen by all processes in the same order.
  - Concurrent writes may be seen in different orders on different machines



(a)

Not permitted



(b)

Permitted

Causal consistency

- Implementing causal consistency requires keeping track of which processes have seen which writes.
- This means that a dependency graph of which operation is dependent on which other operations must be constructed and maintained

Other models

- FIFO consistency: writes from a process are seen by others in the same order. Writes from different processes may be seen in different order (even if causally related)
  - Relaxes causal consistency
  - Simple implementation: tag each write by (Proc ID, seq #)
- Even FIFO consistency may be too strong!
  - Requires all writes from a process be seen in order
- Assume use of critical sections for updates
  - Send final result of critical section everywhere
  - Do not worry about propagating intermediate results
    - Assume presence of synchronization primitives to define semantics

Other Models

- Weak consistency
  - Accesses to synchronization variables associated with a data store are sequentially consistent
  - No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere
  - No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.
- Entry and release consistency
  - Assume shared data are made consistent at entry or exit points of critical sections

Summary of Data-centric Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
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 other in the order they were used. Writes from different processes may not always be seen in that order

(a)

Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization 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.

(b)

Eventual Consistency

- in many data-base systems, most processes hardly ever perform update operations;
  - how fast updates should be made available to only-reading processes?
- Web pages are updated by a single authority

- For the above two examples, if no updates take place for a long time, all replicas will gradually become consistent. This form of consistency is called eventual consistency

Eventual Consistency

- Eventual consistent data stores work fine as long as clients always access the same replica.
- However, problems arise when different replicas are accessed over a short period of time.

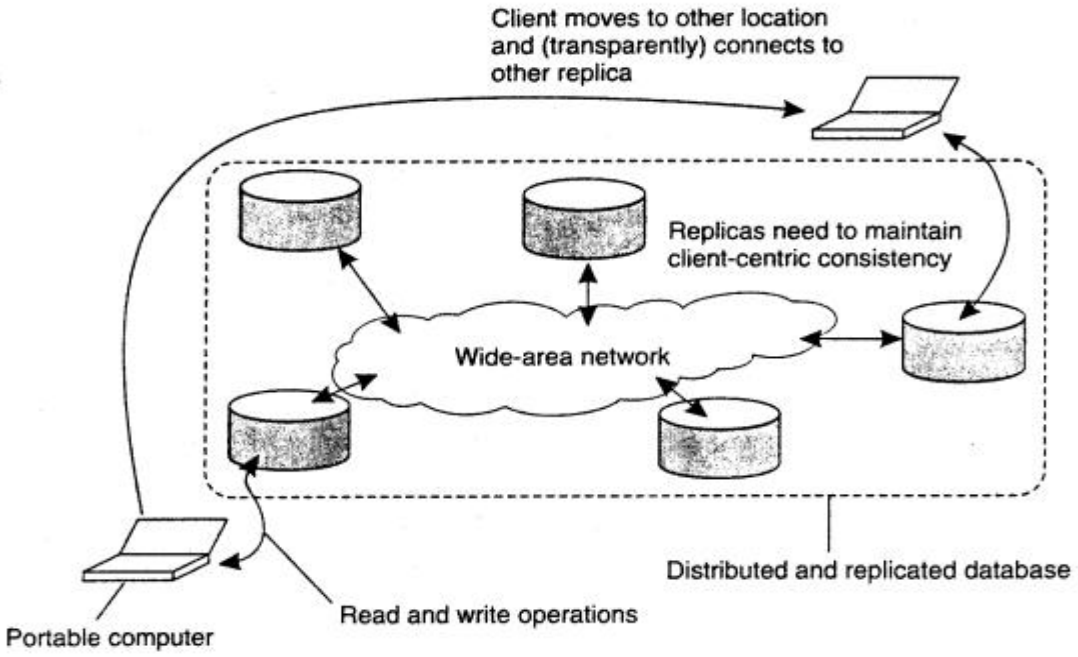


Figure 1-11. The principle of a mobile user accessing different replicas of a distributed database.

Client-centric consistency models

- The example is typical for eventually-consistent data stores and is caused by the fact that users may sometimes operate on different replicas.
- The problem can be alleviated by introducing client-centric consistency
- Client-centric consistency provides guarantees for a single client concerning the consistency of accesses to a data store by that client.
- No guarantees are given concerning concurrent accesses by different clients.

Client-centric consistency models

- Client-centric consistency models originate from the work on Bayou
- Bayou is a database system developed for mobile computing, where it is assumed that network connectivity is unreliable and subject to various performance problems
- Bayou essentially distinguishes four different consistency models
- Assumption
  - data items have an associated owner, which is the only process that is permitted to modify that item

Client-centric consistency models

- The four client-centric consistency models
  - Monotonic Reads
  - Monotonic Writes
  - Read Your Writes
  - Writes Follow Reads

Monotonic Reads

- A data store is said to provide monotonic-read consistency if the following condition holds:
  - *If a process reads the value of a data item x, any successive read operation on x by that process will always return that same value or a more recent value*
- It guarantees that if a process has seen a value of x at time t, it will never see an older version of x at a later time e.g. distributed e-mail database

Monotonic Writes

- Required in situations where write operations are propagated in the correct order to all copies of the data store
- In a monotonic-write consistent store, the following condition holds:
  - *A write operation by a process on a data item x is completed before any successive write operation on x by the same process.*
  - *E.g. updating a software library*
- Note that monotonic-write consistency resembles data-centric FIFO consistency- difference- we are now considering consistency only for a single process instead of for a collection of concurrent processes

Read Your Writes

- A data store is said to provide read-your-writes consistency, if the following condition holds:
  - *The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.*
- A write operation is always completed before a successive read operation by the same process, no matter where that read operation takes place.

Writes Follow Reads

- A data store is said to provide writes-follow-reads consistency, if the following holds:
  - *A write operation by a process on a data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read.*
- Any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by that process