

DISTRIBUTED SYSTEMS (COMP9243)

Lecture 3a: Replication & Consistency

Slide 1

- ① Replication
- ② Consistency
 - Models
 - Protocols
- ③ Update propagation
- ④ Replica placement

REPLICATION

Make copies of services on multiple machines.

Why?:

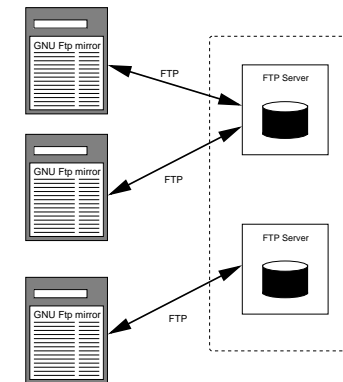
Slide 2

- Reliability
 - Redundancy
- Performance
 - Increase processing capacity
 - Reduce communication
- Scalability (prevent centralisation)
 - Prevent overloading of single server (*size scalability*)
 - Avoid communication latencies (*geographic scalability*)

DATA VS CONTROL REPLICATION

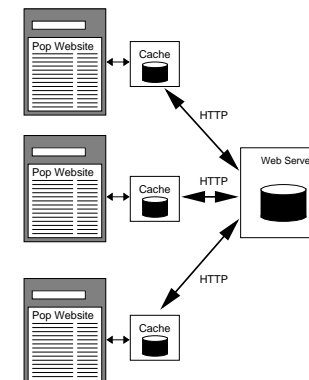
Data Replication (Server Replication):

Slide 3

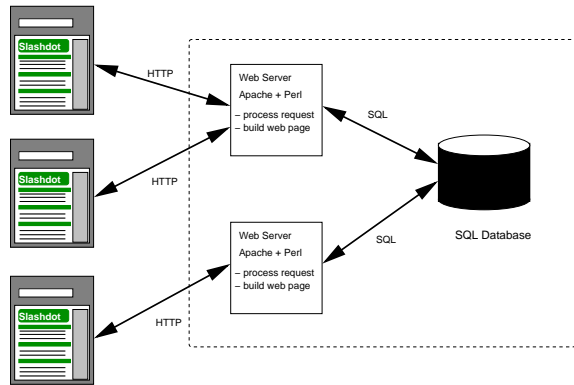


Data Replication (Caching):

Slide 4

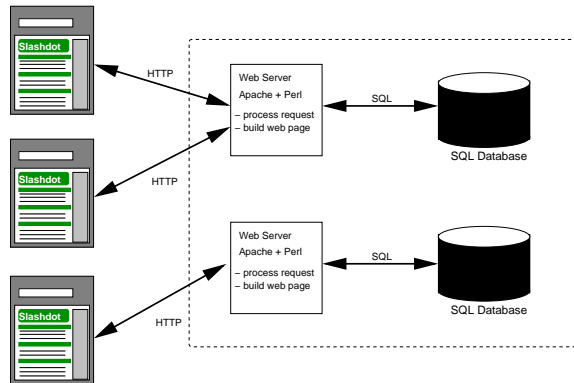


Control Replication:



Slide 5

Data and Control Replication:



Slide 6

Will be looking primarily at data replication (including combined data and control replication).

REPLICATION ISSUES

Updates

- Consistency (how to deal with updated data)
- Update propagation

Slide 7

Replica placement

- How many replicas?
- Where to put them?

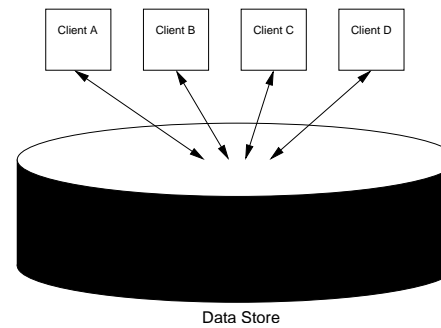
Redirection/Routing

- Which replica should clients use?

DISTRIBUTED DATA STORE

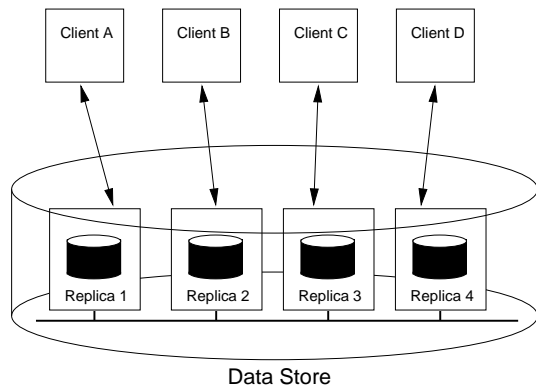
→ data-store stores data items

Client's Point of View:



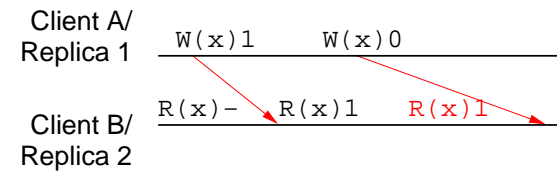
Slide 8

Distributed Data-Store's Point of View:



Slide 9

Timeline:



Slide 11

Operations on a Data Store:

- Read. $R_i(x)$ Client i performs a read for data item x and it returns b
- Write. $W_i(x)$ Client i performs write on data item x setting it to a
- Operations not instantaneous
 - Time of issue (when request is sent by client)
 - Time of execution (when request is executed at a replica)
 - Time of completion (when reply is received by client)
- Coordination among replicas

Slide 10

INCONSISTENCY

Staleness:

- How old is the data?
- How old is the data allowed to be?

Slide 12

- Time
- Versions

Operation order:

- Were operations performed in the right order?
- What orderings are allowed?

CONSISTENCY

Non-distributed data store:

- Program order is maintained
- Data coherence is respected

Updates and concurrency result in conflicting operations

Slide 13

Conflicting Operations:

- Read-write conflict (only 1 write)
- Write-write conflict (multiple concurrent writes)

Consistency:

- The order in which conflicting operations are performed affects consistency
- **partial order**: order of a single client's operations
- **total order**: interleaving of all conflicting operations

Example:

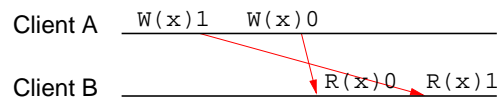
Client A: `x = 1; x = 0;`

Client B: `print(x);`
`print(x);`

Possible results:

--, 11, 10, 00

How about 01?



Slide 14

CONSISTENCY MODEL

Defines which interleavings of operations are valid (admissible)

Consistency Model:

Slide 16

- Concerned with consistency of a data store.
- Specifies characteristics of valid total orderings

A data store that implements a particular model of consistency will provide a total ordering of operations that is valid according to the model.

DATA-CENTRIC CONSISTENCY MODEL

A contract, between a distributed data store and clients, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency.

Slide 17

- Described consistency is experienced by all clients
- Multiple clients accessing the same data store
- Client A, Client B, Client C see same kinds of orderings
- Non-mobile clients (replica used doesn't change)

STRONG ORDERING VS WEAK ORDERING

Strong Ordering (tight):

- All writes must be performed in the order that they are executed
- Example: all clients must see: $W(x)_a W(x)_b W(x)_c$
- Strict (Linearisable) Sequential, Causal, FIFO (PRAM)

Slide 18

Weak Ordering (loose):

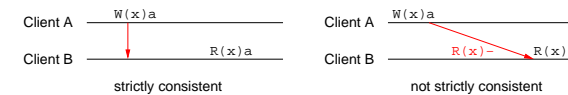
- Ordering of *groups* of writes, rather than individual writes
- Series of writes are grouped on a single replica
- Only results of grouped writes propagated.
- Example: $\{W(x)_a W(x)_b W(x)_c\} == \{W(x)_b W(x)_a W(x)_c\}$
- Weak, Release, Entry

STRICT CONSISTENCY

Any read on a data item x returns a value corresponding to the result of the most recent write on x

Absolute time ordering of all shared accesses

Slide 19



What is *most recent* in a distributed system?

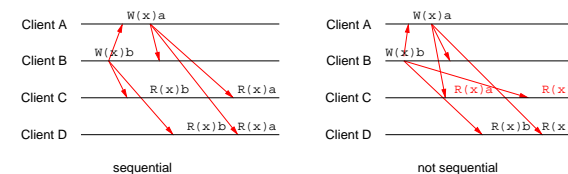
- Assumes an absolute global time
- Assumes instant communication (atomic operation)
- Normal on a uniprocessor
- ✗ Impossible in a distributed system

SEQUENTIAL CONSISTENCY

All operations are performed in some sequential order

- More than one correct sequential order
- All clients see the *same* order
- Program order of each client maintained
- Not ordered according to time

Slide 20



Performance:

read time + write time \geq minimal packet transfer time

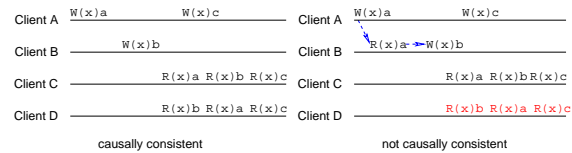
CAUSAL CONSISTENCY

Potentially causally related writes are executed in the same order everywhere

Causally Related Operations:

Slide 21

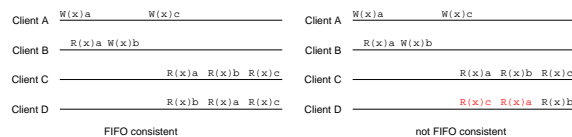
- Read followed by a write (in same client)
- $W(x)$ followed by $R(x)$ (in same or different clients)



FIFO (PRAM) CONSISTENCY

Only partial orderings of writes maintained

Slide 22



WEAK CONSISTENCY

Shared data can be counted on to be consistent only after a synchronisation is done

Enforces consistency on a group of operations, rather than single operations

Slide 23

- Synchronisation variable (S)
- Synchronise operation (`synchronise(S)`)
- Define 'critical section' with synchronise operations

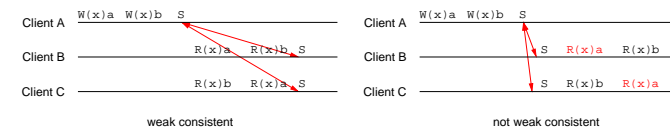
Properties:

- Order of synchronise operations sequentially consistent
- Synchronise operation cannot be performed until all previous writes have completed everywhere
- Read or Write operations cannot be performed until all previous synchronise operations have completed

Example:

- `synchronise(S) W(x)a W(y)b W(x)c synchronise(S)`
- Writes performed locally
- Updates propagated only upon synchronisation
- Only $W(y)b$ and $W(x)c$ have to be propagated

Slide 24



RELEASE CONSISTENCY

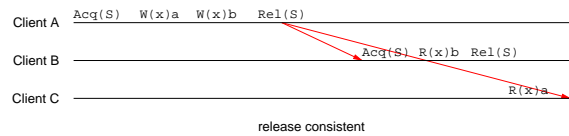
Explicit separation of synchronisation tasks

- `acquire(S)` - bring local state up to date
- `release(S)` - propagate all local updates
- acquire-release pair defines 'critical region'

Slide 25

Properties:

- Order of synchronisation operations are FIFO consistent
- Release cannot be performed until all previous reads and writes done by the client have completed
- Read or Write operations cannot be performed until all previous acquires done by the client have completed

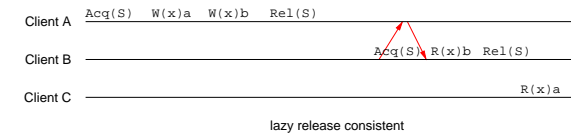


Slide 26

Lazy Release Consistency:

- Don't send updates on release
- Acquire causes client to get newest state
- Added efficiency if acquire-release performed by same client (e.g., in a loop)

Slide 27



ENTRY CONSISTENCY

Synchronisation variable associated with specific shared data item (guarded data item)

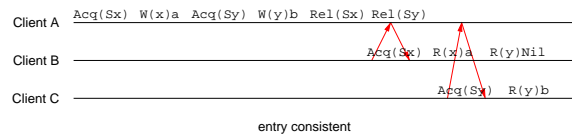
- Each shared data item has own synchronisation variable
- `acquire()`
 - Provides ownership of synchronisation variable
 - Exclusive and nonexclusive access modes
 - Synchronises data
 - Requires communication with current owner
- `release()`
 - Relinquishes exclusive access (but not ownership)

Slide 28

Slide 29

Properties:

- Acquire does not complete until all guarded data is brought up to date locally
- If a client has exclusive access to a synchronisation variable, no other client can have any kind of access to it
- When acquiring nonexclusive access, a client must first get the updated values from the synchronisation variable's current owner



Slide 30

CLIENT-CENTRIC CONSISTENCY MODELS

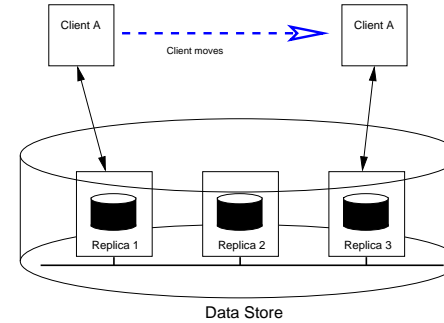
Provides guarantees about ordering of operations for a single client

- Single client accessing data store
- Client accesses different replicas (modified data store model)
- Data isn't shared by clients
- Client A, Client B, Client C may see different kinds of orderings

In other words:

- The effect of an operation depends on the client performing it
- Effect also depends on the history of operations that client has performed.

Data-Store Model for Client-Centric Consistency:



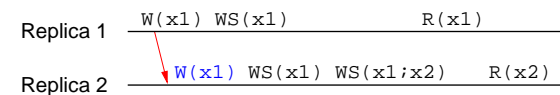
Slide 31

- Data-items have an owner
- No write-write conflicts

Slide 32

Notation and Timeline for Client-Centric Consistency:

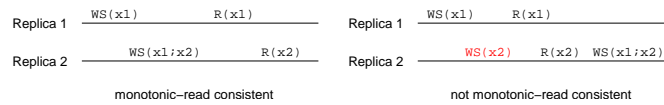
- $x_i[t]$: version of x at replica i at time t
- Write Set: $WS(x_i[t])$: set of writes at replica i that led to $x_i(t)$
- $WS(x_i[t_1]; x_j[t_2])$: $WS(x_j(t_2))$ contains same operations as $WS(x_i(t_1))$
- $R(x_i[t])$: a read of x returns $x_i(t)$



MONOTONIC READS

If a client has seen a value of x at a time t , it will never see an older version of x at a later time

Slide 33



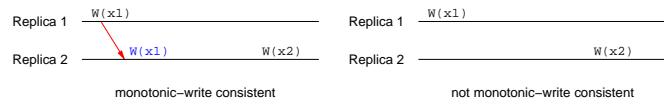
When is Monotonic Reads sufficient?

MONOTONIC WRITES

A write operation on data item x is completed before any successive write on x by the same client

All writes by a single client are sequentially ordered.

Slide 34



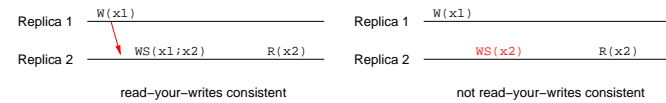
How is this different from FIFO consistency?

- Only applies to write operations of single client.
- Writes from clients not requiring monotonic writes may appear in different orders.

READ YOUR WRITES

The effect of a write on x will always be seen by a successive read of x by the same client

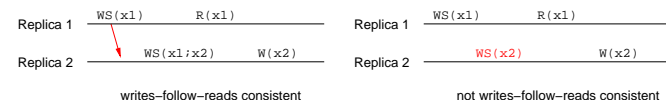
Slide 35



WRITE FOLLOWS READS

A write operation on x will be performed on a copy of x that is up to date with the value most recently read by the same client

Slide 36



CHOOSING THE RIGHT MODEL

Trade-offs

Consistency and Redundancy:

- All copies must be strongly consistent
- All copies must contain full state
- Reduced consistency → reduced reliability

Slide 37

Consistency and Performance:

- Consistency requires extra work
- Consistency requires extra communication
- ✗ Can result in loss of overall performance

Consistency and Scalability:

- Implementation of consistency must be scalable
 - don't take a centralised approach
 - avoid too much extra communication

CONSISTENCY PROTOCOLS

Consistency Protocol: implementation of a consistency model

Primary-Based Protocols:

- Remote-write protocols
- Local-write protocols

Slide 38

Replicated-Write Protocols:

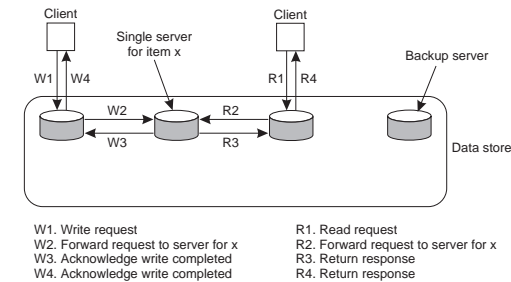
- Active Replication
- Quorum-Based Protocols

REMOTE-WRITE PROTOCOLS

Single Server:

- All writes and reads executed at single server
- No replication of data

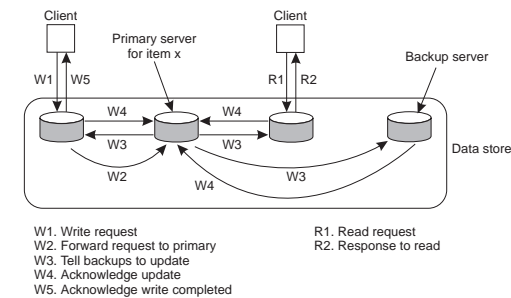
Slide 39



Primary-Backup:

- All writes executed at single server, Reads are local
- Updates block until executed on all backups
- ✗ Performance

Slide 40

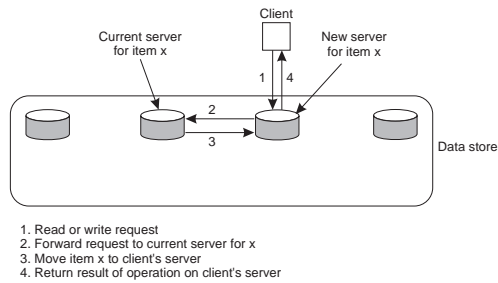


LOCAL-WRITE PROTOCOLS

Migration:

- Data item migrated to local server on access
- Distributed, non-replicated, data store

Slide 41

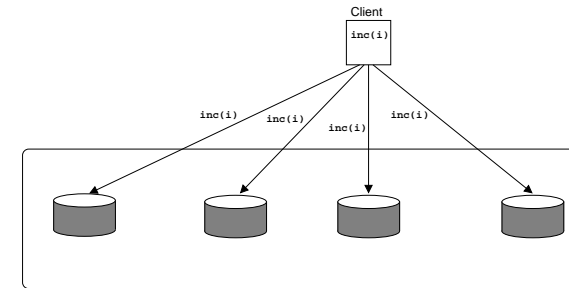


1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

ACTIVE REPLICATION

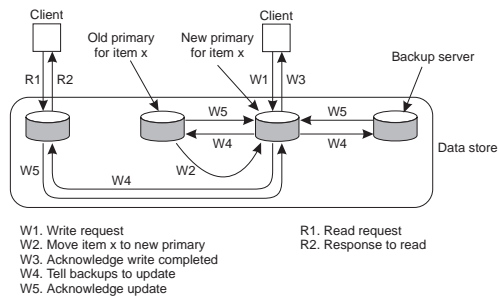
- Updates (write operation) sent to all replicas
- Need totally-ordered multicast
- e.g. sequencer/coordinator to add sequence numbers

Slide 43



Migrating Primary (multiple reader/single writer):

Slide 42

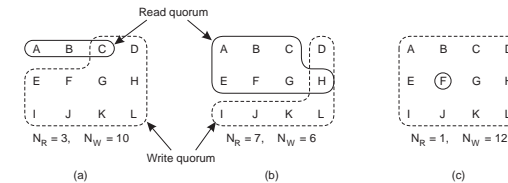


- W1. Write request
 - W2. Move item x to new primary
 - W3. Acknowledge write completed
 - W4. Tell backups to update
 - W5. Acknowledge update
- R1. Read request
 - R2. Response to read

QUORUM-BASED PROTOCOLS

- Voting
- Versioned data
- Read Quorum: N_r
- Write Quorum: N_w
- $N_r + N_w > N$
- $N_w > N/2$

Slide 44



UPDATE PROPAGATION

What to propagate?

- Data
 - R/W high
- Update operation
 - low bandwidth costs
- Notification/Invalidation
 - R/W low

Slide 45

Leases:

Server promises to push updates until lease expires

Lease length depends on:

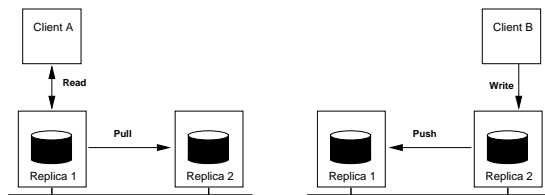
Slide 47

age: Last time item was modified

renewal-frequency: How often replica needs to be updated

state-space overhead: lower expiration time to reduce bookkeeping when many clients

PUSH VS PULL



Slide 46

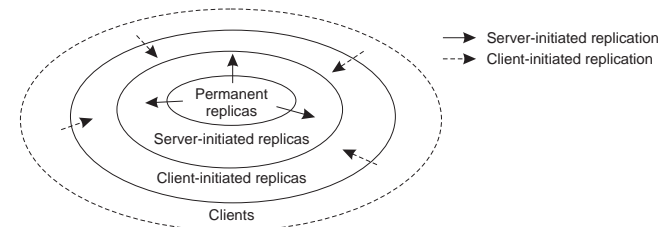
Pull:

- Updates propagated only on request
- Also called *client-based*
- R/W low
- Polling delay

Push:

- Push updates to replicas
- Also called *server-based*
- When low staleness required
- $R \gg W$
- ✗ Have to keep track of all replicas

REPLICA PLACEMENT



Slide 48

Permanent Replicas:

- Initial set of replicas
- Created and maintained by data-store owner(s)
- Allow writes

Server-Initiated Replicas:

- Enhance performance
- Not maintained by owner
- Placed close to groups of clients
 - Manually
 - Dynamically

Slide 49

Client-Initiated Replicas:

- Client caches
 - Temporary
 - Owner not aware of replica
 - Placed close to client
 - Maintained by host (often client)
-

DYNAMIC REPLICATION

Situation changes over time

- Number of users, Amount of data
- Flash crowds
- R/W ratio

Slide 50 Dynamic Replica Placement:

- Network of replica servers
 - Keep track of data item requests at each replica
 - Deletion threshold
 - Replication threshold
 - Migration threshold
 - Clients always send requests to nearest server
-

MISCELLANEOUS IMPLEMENTATION AND DESIGN ISSUES

End-to-End argument:

- Where to implement replication mechanisms?
- Application? Middleware? OS?

Policy vs Mechanism:

- Consistency models built into middleware?
- One-size-fits-all?

Slide 51

Determining Policy:

- Who determines the consistency model used?
 - Application
 - Middleware
 - Client
 - Server
-