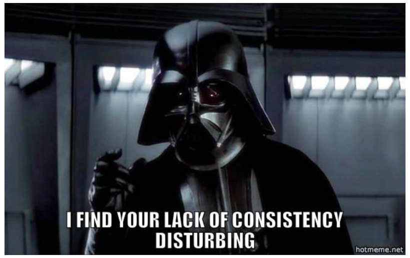
Slide 1

Distributed Systems (COMP9243)

Lecture 5: Replication & Consistency



① Replication
② Consistency
- Models vs Protocols

③ Update propagation

Slide 2

Replication

Make copies of services on multiple machines.

Why?:

➜ Reliability
- Redundancy

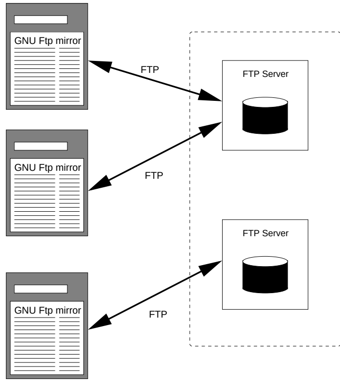
➜ Performance
- Increase processing capacity
- Reduce communication

➜ Scalability (prevent centralisation)
- Prevent overloading of single server (*size* scalability)
- Avoid communication latencies (*geographic* scalability)

Slide 3

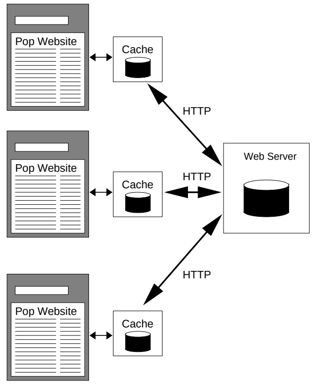
Data vs Control Replication

Data Replication (Server Replication/Mirroring):



Slide 4

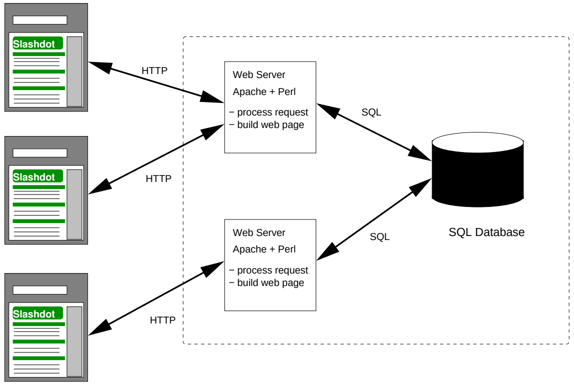
Data Replication (Caching):



What's the difference between mirroring and caching?

Slide 5

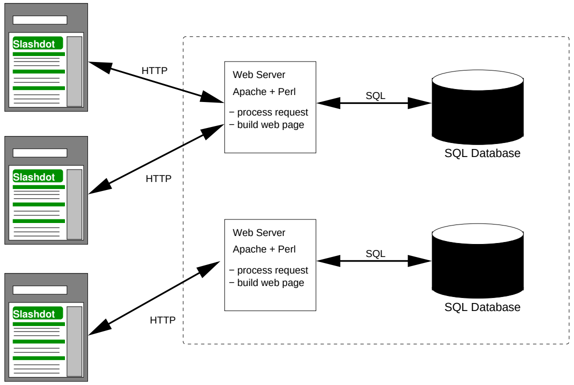
Control Replication:



What are the challenges of doing this?

Slide 6

Data and Control Replication:



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

Slide 7

Replication Issues

Updates

➜ Consistency (how to deal with updated data)

➜ Update propagation

Replica placement

➜ How many replicas?

➜ Where to put them?

Redirection/Routing

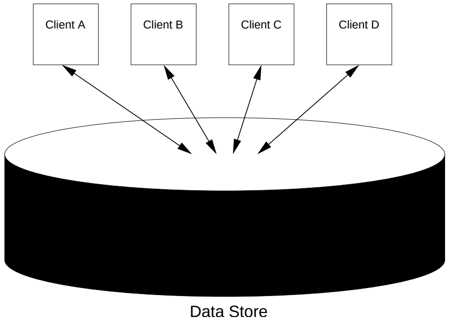
➜ Which replica should clients use?

Slide 8

Distributed Data Store

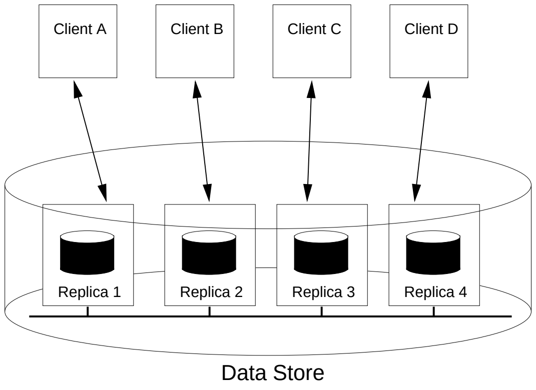
➜ data-store stores data items

Client's Point of View:



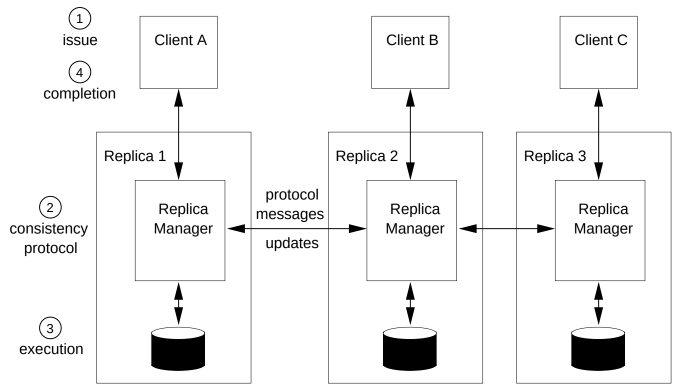
Slide 9

Distributed Data-Store's Point of View:



Slide 11

Replica Managers:



Slide 10

Data Model:

- ➜ data item: simple variable
- ➜ data item values: explicit `(0, 1)`, abstract `(a,b)`
- ➜ data store: collection of data items

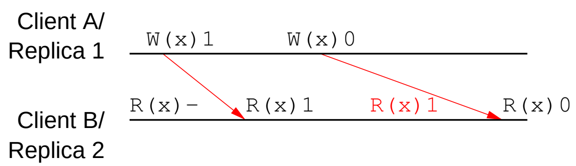
Operations on a Data Store:

- ➜ Read. `Ri(x)b` Client i performs a read for data item `x` and it returns `b`
- ➜ Write. `Wi(x)a` 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 12

Timeline:

- ➜ ClientA/Replica1: `WA(x)1, WA(x)0`
- ➜ ClientB/Replica2: `RB(x)-, RB(x)1, RB(x)1, RB(x)0`



Slide 13

CONSISTENCY

Conflicting Data:

- ➜ Do replicas have exactly the same data?
- ➜ What differences are permitted?

Consistency Dimensions:

- ➜ Time and Order

Time:

- ➜ How old is the data (staleness)?
- ➜ How old is the data allowed to be?
  - Time, Versions

Operation order:

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

Real world examples of inconsistency?

Slide 14

ORDERING

Updates and concurrency result in conflicting operations

Conflicting Operations:

- ➜ Read-write conflict (only 1 write)
- ➜ Write-write conflict (multiple concurrent writes)
- ➜ The order in which conflicting operations are performed affects consistency

Partial vs Total Ordering:

- ➜ partial order: order of a single client's operations
- ➜ total order: interleaving of all conflicting operations

Slide 15

Example:

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

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

Possible results:
--, 11, 10, 00
How about 01?

What are the conflicting ops? What are the partial orders? What are the total orders?

Slide 16

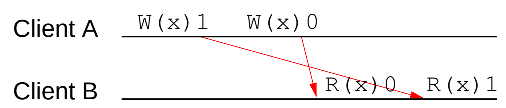
Example:

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

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

Possible results:
--, 11, 10, 00
How about 01?

What are the conflicting ops? What are the partial orders? What are the total orders?



Can you sanely use a system like this?

Slide 17

Consistency Model

Defines which interleavings of operations are valid (admissible)

Consistency Model:

- ➜ 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.

Slide 18

Data Coherence vs Data Consistency:

Data Coherence ordering of operations for single data item

- ➜ e.g. a read of x will return the most recently written value of x

Data Consistency ordering of operations for whole data store

- ➜ implies data coherence
- ➜ includes ordering of operations on other data items too

Non-distributed data store:

- ➜ Data coherence is respected
- ➜ Program order is maintained

Slide 19

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.

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

Slide 20

Strong Ordering vs Weak Ordering

Strong Ordering (tight):

- ➜ All writes must be performed in the order that they are invoked
- ➜ Example: all replicas must see: `W(x)a W(x)b W(x)c`
- ➜ Strict (Linearisable), Sequential, Causal, FIFO (PRAM)

Weak Ordering (loose):

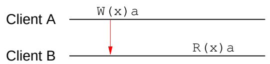
- ➜ Ordering of *groups* of writes, rather than individual writes
- ➜ Series of writes are grouped on a single replica
- ➜ Only results of grouped writes propagated.
- ➜ Example: `{W(x)a W(x)b W(x)c}` == `{W(x)a W(x)c}` == `{W(x)c}`
- ➜ Weak, Release, Entry

Slide 21

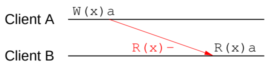
Strict Consistency

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



strictly consistent



not strictly consistent

What is *most recent* in a distributed system?

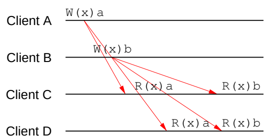
- ➜ Assumes an absolute global time
- ➜ Assumes instant communication (atomic operation)
- ➜ Normal on a uniprocessor
- ✗ Impossible in a distributed system

Slide 22

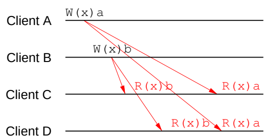
Linearisable Consistency

All operations are performed in a single sequential order

- ➜ Operations ordered according to a global (finite) timestamp.
- ➜ Program order of each client maintained



linearisable



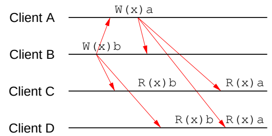
not linearisable

Slide 23

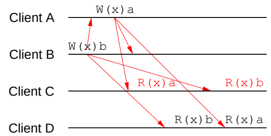
Sequential Consistency

All operations are performed in some sequential order

- ➜ More than one correct sequential order possible
- ➜ All clients see the *same* order
- ➜ Program order of each client maintained
- ➜ Not ordered according to time Why is this good?



sequential



not sequential

Performance:

read time + write time >= minimal packet transfer time

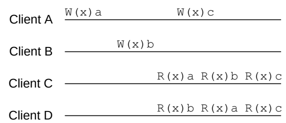
Slide 24

Causal Consistency

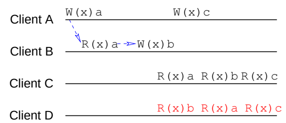
Potentially causally related writes are executed in the same order everywhere

Causally Related Operations:

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



causally consistent



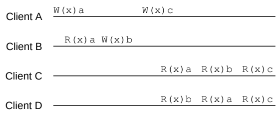
not causally consistent

How could we make this valid?

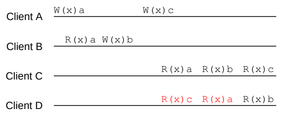
Slide 25

FIFO (PRAM) Consistency

Only partial orderings of writes maintained



FIFO consistent



not FIFO consistent

How could we make this valid?

Slide 26

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

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

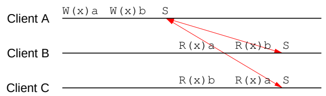
Properties:

- ➜ 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

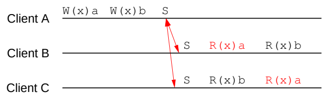
Slide 27

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



weak consistent



not weak consistent

How could we make this valid?

Slide 28

Release Consistency

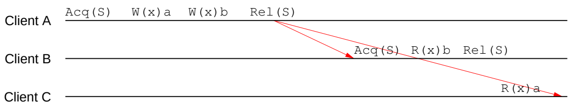
Explicit separation of synchronisation tasks

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

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 29



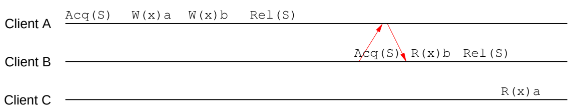
release consistent

What is an example of an invalid ordering?

Slide 30

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)



lazy release consistent

Slide 31

ENTRY CONSISTENCY

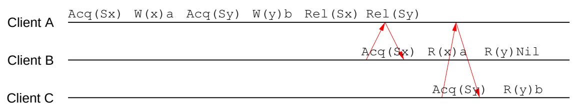
Synchronisation variable associated with specific shared data item (guarded data item)

- ➜ 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 32

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

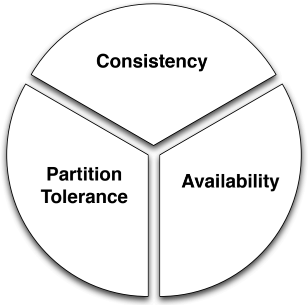


entry consistent

Slide 33

CAP Theory

C: Consistency: Linearisability
A: Availability: Timely response
P: Partition-Tolerance: Functions in the face of a partition

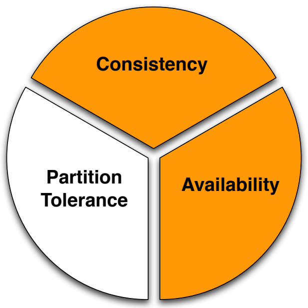


You can only choose two of C A or P

Slide 34

CAP Theory

C: Consistency: Linearisability
A: Availability: Timely response
P: Partition-Tolerance: Functions in the face of a partition

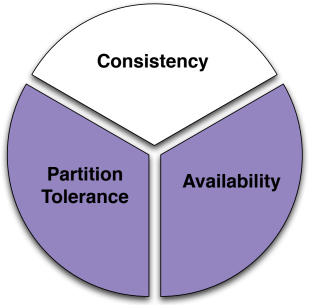


You can only choose two of C A or P

Slide 35

CAP Theory

C: Consistency: Linearisability
A: Availability: Timely response
P: Partition-Tolerance: Functions in the face of a partition

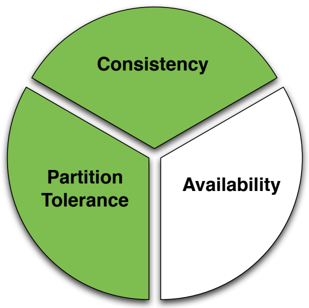


You can only choose two of C A or P

Slide 36

CAP Theory

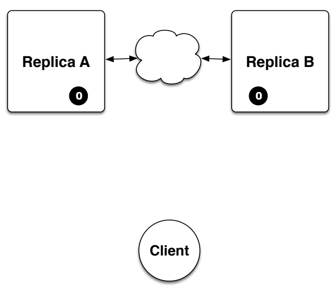
C: Consistency: Linearisability
A: Availability: Timely response
P: Partition-Tolerance: Functions in the face of a partition



You can only choose two of C A or P

CAP Impossibility Proof:

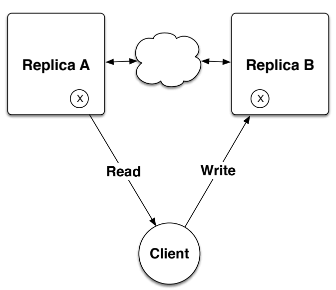
Slide 37



CAP Impossibility Proof:

Goal: Consistency and Availability
No Partition: It works!

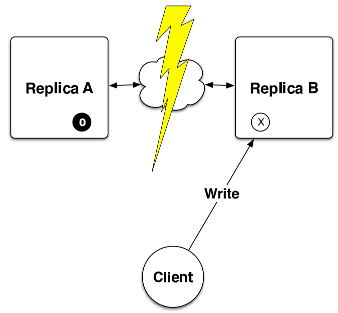
Slide 38



CAP Impossibility Proof:

Partition: no messages between A and B

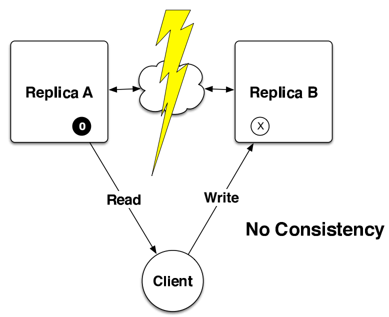
Slide 39



CAP Impossibility Proof:

Assume: Availability and Partition Tolerance

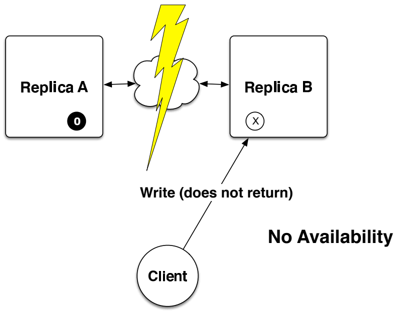
Slide 40



Slide 41

CAP Impossibility Proof:

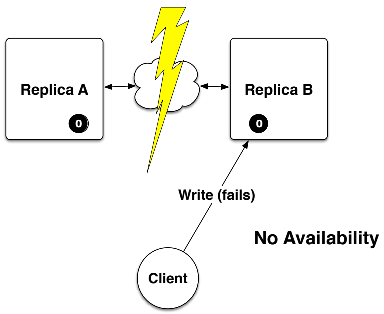
Assume: Consistency and Partition Tolerance



Slide 42

CAP Impossibility Proof:

Assume: Consistency and Partition Tolerance



Slide 43

CAP Consequences

For wide-area systems:

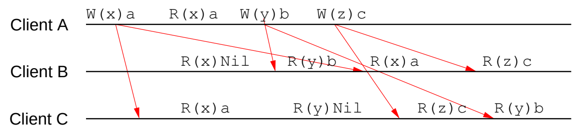
- ➜ Must choose: Consistency or Availability
- ➜ Choosing Availability
  - Give up on consistency?
  - Eventual consistency
- ➜ Choosing Consistency
  - No availability
  - delayed (and potentially failing) operations

Why can't we choose C and A and forget about P?

Slide 44

Eventual Consistency

If no updates take place for a long time, all replicas will gradually become consistent



eventual consistent

Requirements:

- ➜ Few read-write conflicts (R » W)
- ➜ Few write-write conflicts
- ➜ Clients accept time inconsistency (i.e., old data)
- ➜ What about ordering?

Slide 45

Examples:

➜ DNS:
- no write-write conflicts
- updates slowly (1-2 days) propagate to all caches

➜ WWW:
- few write-write conflicts
- mirrors eventually updated
- cached copies (browser or proxy) eventually replaced
- manual merging for write-write conflicts

Slide 46

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

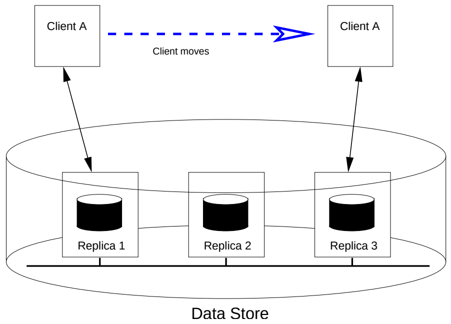
In other words:

➜ The effect of an operation depends on the client performing it

➜ Effect also depends on the history of operations that client has performed.

Slide 47

Data-Store Model for Client-Centric Consistency:



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

Slide 48

Notation and Timeline for Client-Centric Consistency:

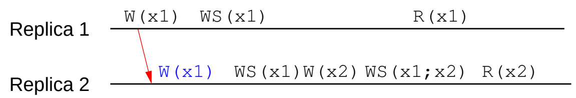
➜ `xi[t]`: version of x at replica i at time t

➜ Write Set: `WS(xi[t])`: set of writes at replica i that led to xi(t)

➜ `WS(xi[t1];xj[t2])`: WS(xj(t2)) contains same operations as WS(xi(t1))

➜ `WS(!xi[t1];xj[t2])`: WS(xj(t2)) does not contain the same operations as WS(xi(t1))

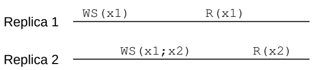
➜ `R(xi[t])`: a read of x returns xi(t)



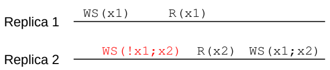
Monotonic Reads

Slide 49

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



monotonic−read consistent



not monotonic−read consistent

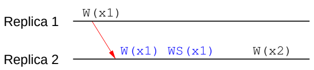
When is Monotonic Reads sufficient?

Monotonic Writes

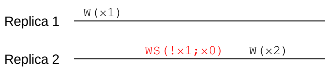
Slide 50

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.



monotonic−write consistent



not monotonic−write consistent

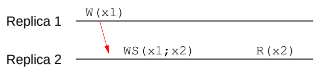
How is this different from FIFO consistency?

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

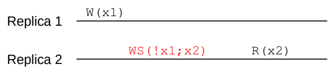
Read Your Writes

Slide 51

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



read−your−writes consistent



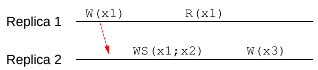
not read−your−writes consistent

When is Read Your Writes sufficient?

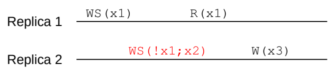
Write Follows Reads

Slide 52

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



writes−follow−reads consistent



not writes−follow−reads consistent

When is Write Follows Reads sufficient?

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 53

Consistency and Performance:
- ➜ Consistency requires extra work and communication
- ✗ Can result in loss of overall performance
- ☑ Weaker consistency possible

Consistency and Scalability:
- ➜ Implementation of consistency must be scalable
  - don't take a centralised approach
  - avoid too much extra communication

Consistency Protocols

Slide 54

Consistency Protocol: implementation of a consistency model (e.g. sequential consistency)

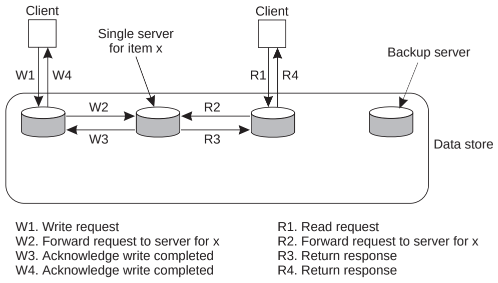
Primary-Based Protocols:
- ➜ Remote-write protocols
- ➜ Local-write protocols

Replicated-Write Protocols:
- ➜ Active Replication
- ➜ Quorum-Based Protocols

Remote-Write Protocols

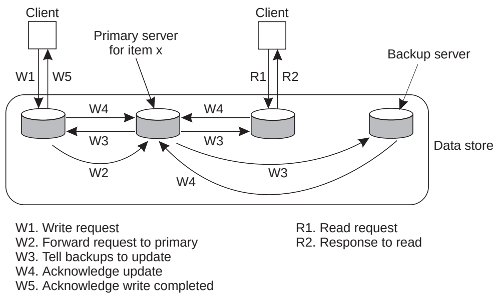
Slide 55

Single Server (single reader/single writer):
- ➜ All writes and reads executed at single server
- ✗ No replication of data



Slide 56

Primary-Backup (multiple reader/single writer):
- ➜ All writes executed at single server, Reads are local
- ➜ Updates block until executed on all backups
- ✗ Performance



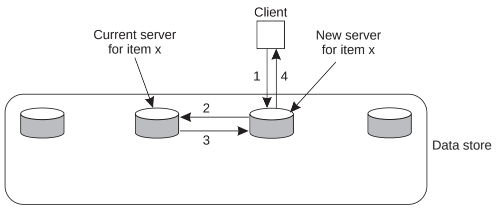
LOCAL-WRITE PROTOCOLS

Migration (single reader/single writer):

➜ Data item migrated to local server on access

☑ Performance (when not sharing data)

Slide 57



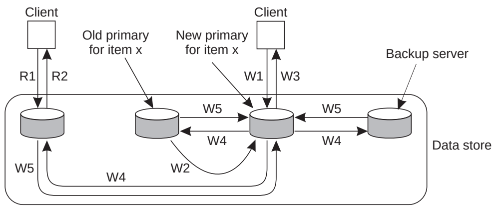
1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

Migrating Primary (multiple reader/single writer):

☑ Performance for concurrent reads

✗ Performance for concurrent writes

Slide 58



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

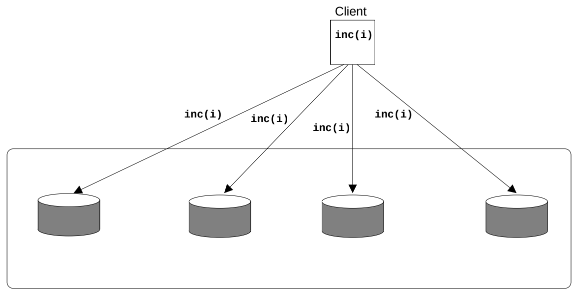
ACTIVE REPLICATION

➜ Updates (write operation) sent to all replicas

➜ Need totally-ordered multicast (for sequential consistency)

➜ e.g. sequencer/coordinator to add sequence numbers

Slide 59



QUORUM-BASED PROTOCOLS

➜ Voting

➜ Versioned data

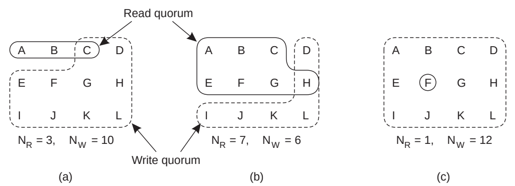
➜ Read Quorum: Nr

➜ Write Quorum: Nw

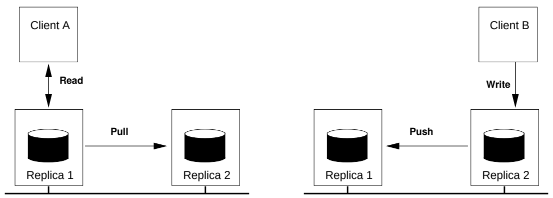
➜ Nr + Nw > N Why?

➜ Nw > N/2 Why?

Slide 60



Push vs Pull



Slide 61

Pull:

- ➜ Updates propagated only on request
- ➜ Also called *client-based*
- ➜ R/W low (W » R)
- ✗ Polling delay

Push:

- ➜ Push updates to replicas
- ➜ Also called *server-based*
- ➜ When low staleness required
- ➜ R/W high (R » W)
- ✗ Have to keep track of all replicas

Slide 62

Push Update Propagation:

What to propagate?

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

Slide 63

Compromise: Leases:

Server promises to push updates until lease expires

Lease length depends on:

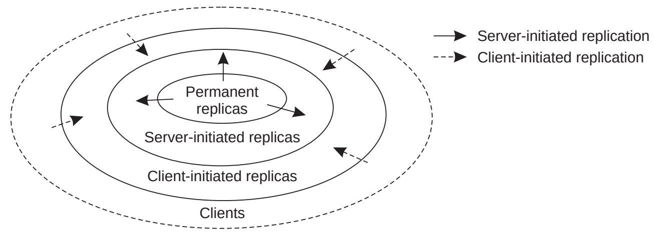
age: Last time item was modified

renewal-frequency: How often replica needs to be updated

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

Replica Placement

Slide 64



Slide 65

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
- ➜ May or may not allow writes
- ➜ Placed close to groups of clients
  - Manually vs Dynamically

Client-Initiated Replicas:

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

Slide 66

Dynamic Replication

Situation changes over time

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

Dynamic Replica Placement:

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

Slide 67

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?

Determining Policy:

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

Keep It Simple, Stupid:

- ➜ Will the programmer understand the consistency model?

Slide 68

Reading List

Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services An overview of the CAP theorem and its proof.

Eventual Consistency An overview of eventual consistency and client-centric consistency models.

Slide 69

Homework

Consistency Models:

- ➜ Research consistency models used in existing Distributed Systems
- ➜ Why are those models being used?
- ➜ In the systems you looked at, could other models have been used? Would that have made the system better?

Hacker's Edition:

- ➜ Find a system that provides Eventual Consistency
  - ➜ (alternatively, implement (possibly in Erlang) a system that provides Eventual Consistency)
- ➜ Replicate some data and perform queries. How often do you get inconsistent results?
- ➜ If you can tweak replication parameters, how do they affect the consistency of results?