

## CSSE4004-Lecture 8

### Consistency and replication

## Learning Objectives Consistency and replication

- Trade-offs
  - performance
  - fault tolerance
- Consistency models
  - definitions and differences
  - which to use when
- Data-centric vs. client-centric model
  - design choices and issues

2

## Outline

- Introduction (what's it all about?)
- Data-centric consistency
  - Processes access replicated data
- Client-centric consistency
  - Mobile users view of consistency of their replicated data
- Replica Management
- Consistency protocols

3

## Rationale

- Replication of services
  - Increase availability
  - Enhance reliability (switch to another on crash)
  - Improve performance (load balance, data closer, support scaling in numbers and area)
- Management of replicated data:
  - Scalability of keeping replicas consistent

4

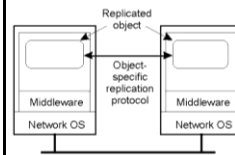
## 2 models for controlling replication

- Object aware of replication
  - Object is responsible for managing consistency
- Distributed system manages replication
  - the more common model
    - example: Piranha (fault-tolerant, totally-ordered, and causally-ordered object invocations in CORBA)
    - easier for developers
    - may be harder to develop application-specific solutions

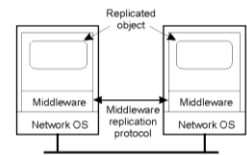
5

## Object Replication

Object-specific management or general approach?



(a) Distributed system for replication-aware distributed objects.



(b) Distributed system responsible for replication management

6

## Performance and Scalability

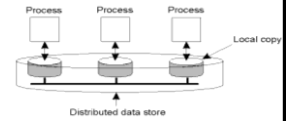
- Main issue: to keep replicas consistent, good idea to ensure that all conflicting operations done in same order everywhere
- Conflicting operations (transactions)
  - read-write / write-write conflict
- But: guaranteeing global ordering on conflicting operations may be costly, downgrading scalability

*weaken consistency requirements to reduce global synchronization*

7

## Data-Centric Consistency Models

**Data store:** distributed collection of storages accessible to clients



**Consistency model:** contract between (distributed) data store and processes, in which data store specifies precisely what results of read and write operations are in presence of concurrency.

8

## Data-Centric Consistency Models

**Strong consistency:** operations on shared data *coherent*

- Strict consistency (related to time)
- Sequential consistency (what we are used to)
- Causal consistency (maintains only causal relations)

**Weak consistency:** coherence only when shared data locked, unlocked

- Entry consistency

*the weaker the consistency model, the easier to scale*

9

## Underlying Problem

*time in distributed systems is hard to pin down – so anything relying on a strong model of ordering is not practical*

10

## Strict Consistency

**Def:** Any read to a shared data item X returns the value stored by the *most recent* write operation on X.

Does it make sense to talk about “most recent” in distributed environment?

All data items have been initialized to NIL

- $W(x)a$ : value  $a$  is written to  $x$
- $R(x)a$ : reading  $x$  returns the value  $a$

Strict consistency is as in “normal” sequential code, where your program does not interfere with any other – as in (a), not (b):

P1: $W(x)a$	P1: $W(x)a$
P2: $R(x)a$	P2: $R(x)NIL$ $R(x)a$
(a)	(b)

11

## Sequential Consistency...

**Def:** Result of any execution same as if operations of all processes executed in some sequential order, and operations of each individual process appear in this sequence in order specified by its program

*actual order doesn't matter but all processes see same order*

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

sequentially consistent data store

data store not sequentially consistent

12

## ...Sequential Consistency

Process P1	Process P2	Process P3
x = 1; print (y, z);	y = 1; print (x, z);	z = 1; print (x, y);

- Three concurrently executing processes
  - how many different orders can the statements complete in (each assumed atomic)?
  - in sequential order as long as ordering of each process maintained
    - 720 execution sequences (6!) but some of them violate program order
    - 90 different orders (though here < 64 distinct allowed results)

13

## Causal Consistency...

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

Assume that W(x)a and W(x)b causally related as W(x)b is a result of R(x)a

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

allowed with causally-consistent store, but not with sequentially or strictly consistent store

14

## ...Causal Consistency

Assume that W(x)a in P1 and W(x)b in P2 are causally related

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

(a)

violation of causally-consistent store

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

(b)

correct sequence of events in causally-consistent store

15

## Grouping operations – weaker consistency

- Most applications use transactions or enter critical sections
- Consistency of whole set of operations is of interest not single read/writes
  - During transactions or operations in critical sections (CS) concurrent access to data used in CS is not allowed
  - Entering and leaving CS can be modeled by shared synchronisation variables
    - When process enters CS it should acquire synchronisation variables
    - When it leaves it releases these variables

16

## Grouping operations - weaker consistency

- Basic idea:
  - don't care that reads and writes in series of operations immediately known to other processes: just want *effect of series* overall to be known
  - access to shared variables not protected by synchronization a bug so inconsistency not a problem
- Synchronisation variable is owned by process which acquired it
  - other processes have to request the variable from it

17

## ...Entry Consistency...

- Conditions:
  - acquire may not complete until all guarded shared data brought up to date
  - before updating shared data, process must enter critical region in exclusive mode
  - if process wants to enter critical region in nonexclusive mode, must check with owner of synchronization variables that it has the most recent copies of guarded shared data

18

## ...Entry Consistency

P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)  
P2: Acq(Lx) R(x)a R(y)NIL  
P3: Acq(Ly) R(y)b

Valid event sequence for *entry consistency* –  
different locks for different variables

19

## Summary of Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses.
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.

(a) Consistency models not using synchronization variables

Consistency	Description
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

(b) Models with synchronization variables

20

## Consistency in well known distributed systems

Most of large-scale distributed systems apply replication for scalability, but tolerate high degree of inconsistency:

- [DNS:] Updates propagated slowly, inserts may not be immediately visible
- [NEWS:] Articles and reactions are pushed and pulled throughout Internet: reactions can be seen before postings
- [WWW:] Caches all over, no guarantee you're reading most recent version of page

21

## Eventual consistency

- For very large databases
  - with tolerance for high degree of inconsistency
  - updates guaranteed to propagate to all replicas
  - if no updates for long time, all replicas gradually become consistent (identical copies)
- Examples
  - WWW
  - DNS
- Eventual consistency works well if always the same replica is accessed

22

## Client-Centric Coherence Models

- Goal: avoid system-wide consistency
- Concentrate on single client's needs
  - mobile clients
  - clients may have a distributed store but at a time work only on a local replica
    - No simultaneous updates
    - Most operations involve reading data

23

## Client-Centric Coherence Models

- Mobile user example
- Monotonic reads
- Monotonic writes
- Read-your-writes
- Write-follows-reads

24

## Example of consistency for mobile users...

Consider distributed replicated database with access through your notebook = front end

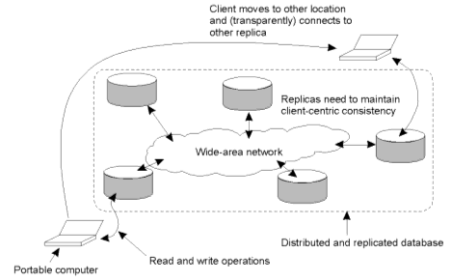
- at  $A$  do reads and updates
- at location  $B$  continue work, but unless use same server as at  $A$ , may detect inconsistencies:
  - your updates at  $A$  may not have yet been propagated to  $B$
  - you may read newer entries than available at  $A$
  - your updates at  $B$  may eventually conflict with those at  $A$

Only thing **you** really want is entries **you** updated/read at  $A$  are in  $B$  as you left them in  $A$ . If so, DB appears consistent **to you**.

25

## ...Example of consistency for mobile users

Mobile user, different replicas of distributed DB



26

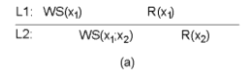
## Notation

- Adapt previous graphical notation
  - except now instead of each line a process, each line a location
- Some client-centric notation
  - $x_i[t]$  is version of data item  $x$  at local copy  $L_i$  at time  $t$
  - $WS(x_i[t])$  is set of write operations (at  $L_i$ ) that lead to version  $x_i[t]$
  - $WS(x_i[t_1]; x_j[t_2])$  indicates it's known that operations in  $WS(x_i[t_1])$  performed at local copy  $L_j$  at later time  $t_2$
- If timing clear, drop the  $t_k$

27

## Monotonic Reads

Def: If process reads value of data item  $x$ , any successive read operation on  $x$  by that process will always return that same or more recent value



A monotonic-read consistent data store

$WS(x_i[t_1]; x_j[t_2])$  implies that the operations at time  $t_2$  include all those in  $WS(x_i[t_1])$



these writes don't include those at L1

A data store that does not provide monotonic reads.

28

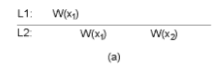
## Example Monotonic reads

- Automatically reading personal calendar updates from different servers. Monotonic Reads guarantees user sees all updates, no matter from which server automatic reading takes place
- Reading (not modifying) incoming mail while on the move. Each time connect to different e-mail server, server fetches (at least) all updates from server previously visited

29

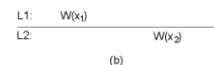
## Monotonic Writes

Write operation by process on data item  $x$  completed before any successive write on  $x$  by same process



A monotonic-write consistent data store

Example: Update program at server  $S_1$ , and ensure that all components on which compilation and linking depends, are also placed at  $S_1$



Example: Maintain versions of replicated files in correct order everywhere (propagate previous version to server where newest version installed)

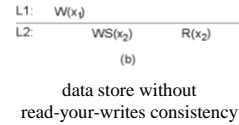
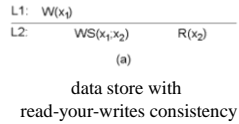
A data store that does not provide monotonic-write consistency

30

## Read Your Writes

Effect of write operation by process on data item  $x$ , always seen by successive read operation on  $x$  by same process

Example: Update your Web page and guarantee your browser shows newest version instead of cached copy

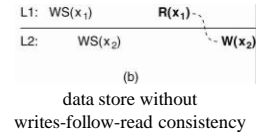
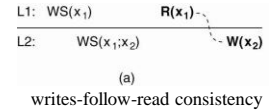


31

## Writes Follow Reads

Write operation by process on data item  $x$  following previous read operation on  $x$  by same process is guaranteed to take place on same or more recent value of  $x$  that was read.

Example: response to newsgroup article always written to any news server after original article



32

## Replica management

- Content Replication and placement
- Update Propagation

33

## Replica Placement

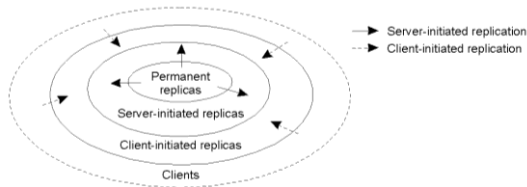
**Model:** consider objects (don't care if data or code)

**Distinguish different processes:** process capable of hosting replica of object or data

- Permanent replicas: process/machine always has replica
- Server-initiated replica: Process can dynamically host replica on request of server in data store (push cache)
  - Used to improve performance
- Client-initiated replica: Process can dynamically host replica on request of client
  - Client caches

34

## Replica Placement – data store replication

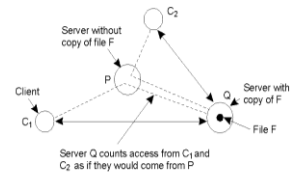


logical organization of different kinds of copies of data store into 3 concentric rings

35

## Server-Initiated Replicas

Keep track of access counts per file, aggregated by considering server closest to requesting clients



- Number of accesses drops below threshold  $D \Rightarrow$  drop file
- Number of accesses exceeds threshold  $R \Rightarrow$  replicate file
- Number of access between  $D$  and  $R \Rightarrow$  may only migrate file

36

## Update propagation

There are three possibilities:

- Propagate only a notification of an update
  - invalidation protocols
- Transfer data from one copy to another
  - used for high read-to-write ratio
- Propagate the update operation to other copies
  - active replication

37

## Pull versus Push Protocols for updates

Issue	Push-based (server-based protocols)	Pull-based (client-based protocols)
State at server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

comparison between push-based and pull-based protocols in case of multiple client, single server systems

38

## Consistency protocols

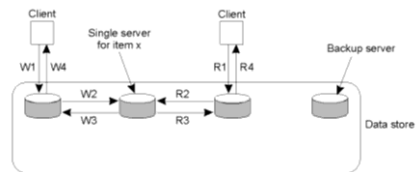
Implementation of specific consistency model

- Primary-based protocols
- Replicated-write protocols

39

## Primary-Based Protocols...

Fixed server to which all read and write operations forwarded  
Example: traditional client-server systems without replication

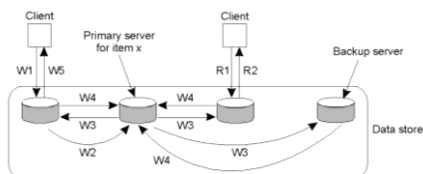


W1. Write request  
W2. Forward request to server for x  
W3. Acknowledge write completed  
W4. Acknowledge write completed  
R1. Read request  
R2. Forward request to server for x  
R3. Return response  
R4. Return response

40

## Primary-backup protocol

Example: Distributed Databases that require absolute correctness  
- update becomes blocking operation.



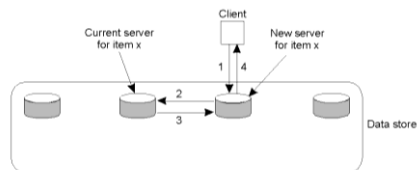
W1. Write request  
W2. Forward request to primary  
W3. Tell backups to update  
W4. Acknowledge update  
W5. Acknowledge write completed  
R1. Read request  
R2. Response to read

...

## Primary-Based Local-Write Protocols

Single copy migrated between processes

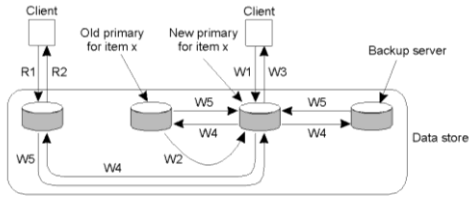
Example: Establishes only fully distributed, nonreplicated data store.  
Useful when writes expected in series from same client (e.g., mobile computing without replication)



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

42

## Primary-backup Local-Write Protocol



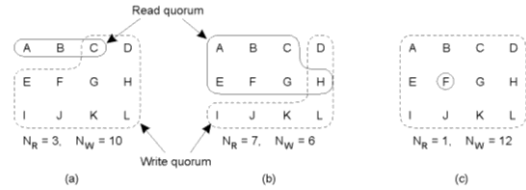
- W1. Write request
- W2. Move item x to new primary
- W3. Acknowledge write completed
- W4. Tell backups to update
- W5. Acknowledge update
- R1. Read request
- R2. Response to read

*primary migrates to process wanting to update*

43

## Quorum-Based Protocols

Ensure that each operation carried out in such a way that majority vote established: distinguish **read quorum** and **write quorum** [Gifford 1979]:



- a) correct choice of read and write set
- b) choice that may lead to write-write conflicts
- c) correct choice, known as ROWA (read one, write all)

44

## Implications of Gifford's Scheme

- General rule ( $N_R$  read quorum,  $N_W$  write quorum):
  - get  $N_R$  replicas to agree before read,  $N_W$  for write
  - meet conditions
    - $N_R + N_W > N$  (prevents read-write conflicts)
    - $N_W > N/2$  (prevents write-write conflicts)
- Specific cases?
  - $N_R = 0, N_W = N/2 + 1$  ?
  - $N_R = 1, N_W = \dots$  ?

45

## Consistency and replication - Summary

- Goals:
  - Improving reliability
  - Improving performance
- Introduces consistency problems
  - Degrade performance
- Tradeoffs:
  - Consistency model, what is propagated, where to propagate, by whom propagation is initiated
- Reading: Chapter 7 (7.1 - 7.4, 7.5 (7.5.2, 7.5.3))

46