CprE 450/550x
Distributed Systems and Middleware

Consistency and Replication

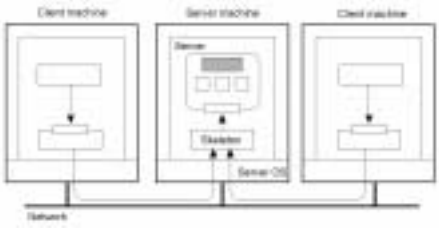
Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu
April 13 & 15 & 20, 2004

Readings for Today's Lecture

- References
  - Chapter 6 of "Distributed Systems: Principles and Paradigms"

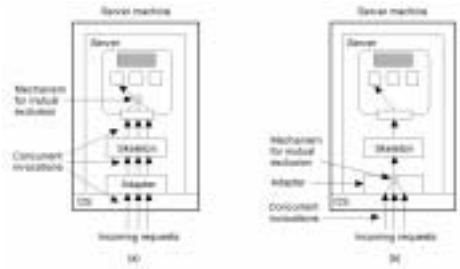
Introduction to Consistency & Replication

Object Replication



Organization of a distributed remote object shared by two different clients.

Object Replication (2)

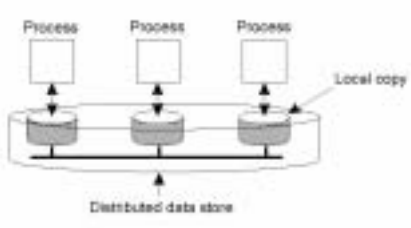


a) A remote object capable of handling concurrent invocations on its own.
b) A remote object for which an object adapter is required to handle concurrent invocations

Object Replication (3)

a) A distributed system for replication-aware distributed objects.
b) A distributed system responsible for replica management

Data-Centric Consistency Models



The general organization of a logical data store, physically distributed and replicated across multiple processes.

7

Strict Consistency

- Behavior of two processes, operating on the same data item.
- A strictly consistent store.
- A store that is not strictly consistent.

8

Linearizability and Sequential Consistency (1)

a) A sequentially consistent data store.
b) A data store that is not sequentially consistent.

9

Linearizability and Sequential Consistency (2)

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.

10

Linearizability and Sequential Consistency (3)

x = 1;
print ((y, z);
y = 1;
print (x, z);
z = 1;
print (x, y);
	x = 1;
	y = 1;
	print (x,z);
	print(y, z);
	z = 1;
	print (x, y);
		y = 1;
		z = 1;
		print (x, y);
		print (x, z);
		x = 1;
		print (y, z);
			y = 1;
			x = 1;
			z = 1;
			print (x, z);
			print (y, z);
			print (x, y);

Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011 (a)	Signature: 101011 (b)	Signature: 110101 (c)	Signature: 111111 (d)

Four valid execution sequences for the processes of the previous slide. The vertical axis is time.

11

Casual Consistency (1)

- Necessary condition:
  Writes that are potentially casually related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.

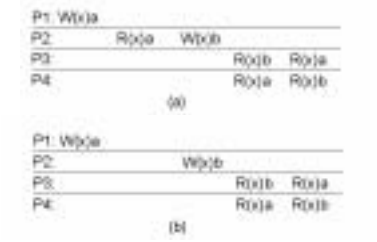
12

Casual Consistency (2)

This sequence is allowed with a casually-consistent store, but not with sequentially or strictly consistent store.

13

Casual Consistency (3)



a) A violation of a casually-consistent store.
b) A correct sequence of events in a casually-consistent store.

14

FIFO Consistency (1)

- Necessary Condition:
  Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.

15

FIFO Consistency (2)

A valid sequence of events of FIFO consistency

16

FIFO Consistency (3)

x = 1;	x = 1;	y = 1;
print (y, z);	y = 1;	print (x, z);
y = 1;	**print(x, z);**	z = 1;
print(x, z);	print (y, z);	**print (x, y);**
z = 1;	z = 1;	x = 1;
print (x, y);	print (x, y);	print (y, z);
Prints: 00	Prints: 10	Prints: 01
(a)	(b)	(c)

Statement execution as seen by the three processes from the previous slide. The statements in bold are the ones that generate the output shown.

17

FIFO Consistency (4)

Process P1	Process P2
x = 1;	y = 1;
if (y == 0) kill (P2);	if (x == 0) kill (P1);

Two concurrent processes.

18

Weak Consistency (1)

- Properties:
- 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.

19

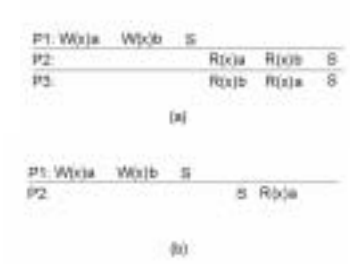
Weak Consistency (2)

```
int a, b, c, d, e, x, y;          /* variables */
int *p, *q;                       /* pointers */
int f( int *p, int *q);           /* function prototype */

a = x * x;                        /* a stored in register */
b = y * y;                        /* b as well */
c = a*a*a + b*b + a * b;          /* used later */
d = a * a * c;                    /* used later */
p = &a;                           /* p gets address of a */
q = &b                            /* q gets address of b */
e = f(p, q)                       /* function call */
```

A program fragment in which some variables may be kept in registers.

20

Weak Consistency (3)



a) A valid sequence of events for weak consistency.
b) An invalid sequence for weak consistency.

21

Release Consistency (1)

A valid event sequence for release consistency.

22

Release Consistency (2)

- Rules:
- Before a read or write operation on shared data is performed, all previous acquires done by the process must have completed successfully.
- Before a release is allowed to be performed, all previous reads and writes by the process must have completed
- Accesses to synchronization variables are FIFO consistent (sequential consistency is not required).

23

Entry Consistency (1)

- Conditions:
- An acquire access of a synchronization variable is not allowed to perform with respect to a process until all updates to the guarded shared data have been performed with respect to that process.
- Before an exclusive mode access to a synchronization variable by a process is allowed to perform with respect to that process, no other process may hold the synchronization variable, not even in nonexclusive mode.
- After an exclusive mode access to a synchronization variable has been performed, any other process's next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.

24

Entry Consistency (2)

- A valid event sequence for entry consistency.

25

Summary of Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	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 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)

a) Consistency models not using synchronization operations.
b) Models with synchronization operations.

26

27

Client-Centric Consistency Models

- Data-centric consistency models
  - **Multiple concurrent processes may simultaneously update the data store**
- Today, we are focusing on a special class of distributed data stores.
  - **There are no or very few simultaneous updates on the data store.**
  - **When such concurrent updates happen, they can be easily resoved.**
  - **Most operations are reading.**
  - **We will introduce a very weak consistency model – eventual consistency.**

28

Client-Centric Consistency Models

- Examples in which concurrency happens in a restricted manner:
  - **Database systems: read-only**
  - **DNS**
  - **WWW**
  - **They are in common that they can tolerate a relatively high degree of inconsistency.**
- Eventual consistency: If no updates take place for a long time, all replicas will gradually and eventually become consistent.

29

Eventual Consistency: Issue

Will work fine if client always access the same replica.

What about when different replicas are accessed?

The principle of a mobile user accessing different replicas of a distributed database.

30

The previous problem can be alleviated

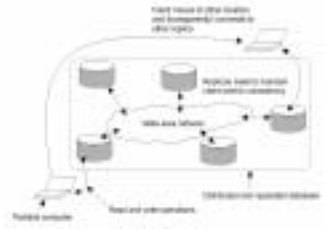
- By Using Client-centric consistency:
  - **Client-centric consistency provides guarantees for a single client concerning the consistency of accesses to a data store by that client**
  - **No guarantees are given concerning concurrent accesses by different clients.**
  - **Originated from the work Bayou.**
  - **In this model, we assume there is only one process that is permitted to update the data store.**

31

Monotonic Reads

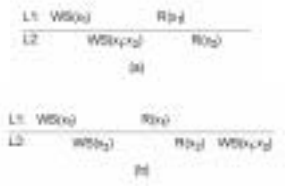
- Condition:

  If a process reads the value of a data item x, any successive read operations on x by that process will always return that same value or a more recent value.



32

Monotonic Reads



The read operations performed by a single process *P* at two different local copies of the same data store.

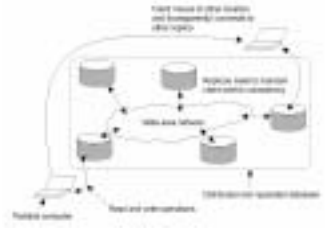
a) A monotonic-read consistent data store
b) A data store that does not provide monotonic reads.

33

Monotonic Writes

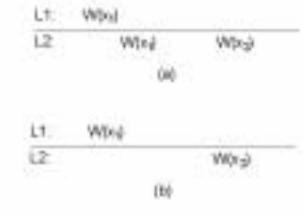
- Condition:

  A write operation by a process on a data item x is completed before any successive write operation on x by the same process.



34

Monotonic Writes



- The write operations performed by a single process *P* at two different local copies of the same data store

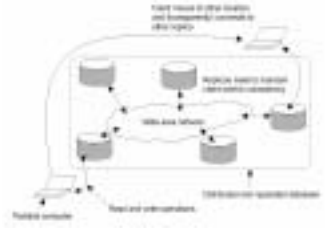
a) A monotonic-write consistent data store.
b) A data store that does not provide monotonic-write consistency.

35

Read your Writes

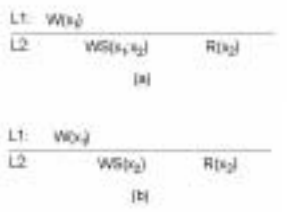
- Condition:

  The effect of a write operation by a process on a data item x will always be seen by a successive read operation on x by the same process.



36

Read Your Writes



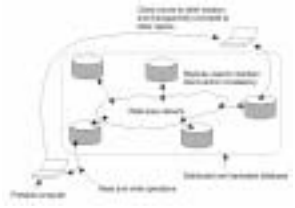
a) A data store that provides read-your-writes consistency.
b) A data store that does not.

37

Writes Follow Reads

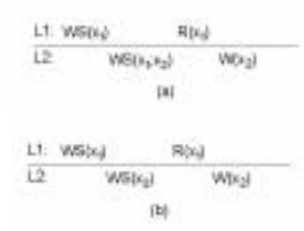
- Condition:

  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.



38

Writes Follow Reads



a) A writes-follow-reads consistent data store
b) A data store that does not provide writes-follow-reads consistency

39

Implementations Issues

- Relatively straightforward without considering performance issues
- Each write operation is assigned a globally unique identifier.

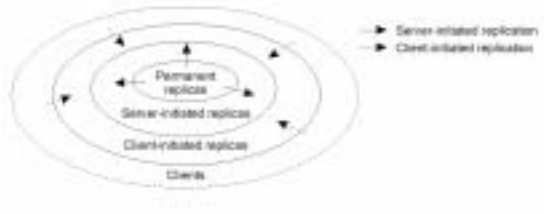
40

Distributed Protocols

- Replica Placement
- Update Propagation
- Epidemic Protocols

41

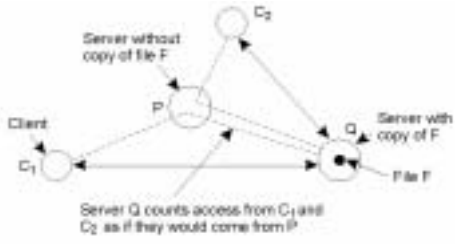
Replica Placement



The logical organization of different kinds of copies of a data store into three concentric rings.

42

Server-Initiated Replicas



Counting access requests from different clients.

43

Client-Initiated Replicas

- Client cache
- Placement of client cache

44

Update Propagaation

- State versus operations
- Pull versus pull protocols
- Unicast versus multicast

45

Pull versus Push Protocols

Issue	Push-based	Pull-based
State of 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

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

46

Epidemic Protocols

- EP does not solve update conflicts. Propagate updates to all replicas in as few messages as possible.
- Update Propagation Models
  - **Infective if it holds an update that it is willing to spread to other servers**
  - **Susceptible if a server has not been updated yet.**
  - **Removed if an updated server that is not willing to or able to spread its update**
- Anti-entropy model:
  - **Server P chooses Q randomly and then exchanges updates with Q:**
    - » **P pushes its own update to Q**
    - » **P pulls in new updates from Q**
    - » **P and Q send updates to each other.**

47

Epidemic Protocols

- Variant: Rumor Spreading/gossiping

48

49

Consistency Protocols

- We have studied various consistency models.
- Today, we will focus on issues of implementation of consistency models:
  - Whether or not there is a primary copy of the data to which all write operations should be forwarded.
  - When no such primary copy exists, a write operation can be initiated at any replica.
- Primary-based protocols
- Replicated-write protocols
- Cache-coherence protocols

50

Primary-based protocols

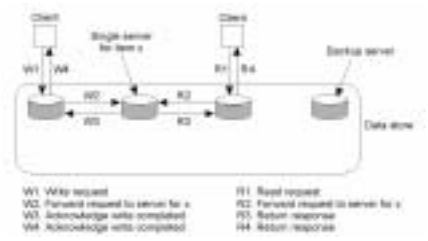
Each date item x has an associated primary for coordinating write operations on x.

Depend on whether primary is fixed or movable.

- Remote-write protocols
  - No replication
  - All read and write operations are carried out at a (remote) single server.
- Local-write protocols
  - Fully-migrating approaches: keeping track of data item
  - Primary-based approaches

51

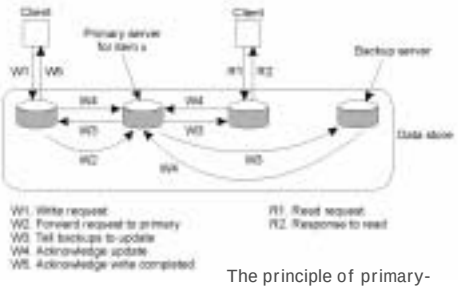
Remote-Write Protocols (1)



Primary-based remote-write protocol with a fixed server to which all read and write operations are forwarded.

52

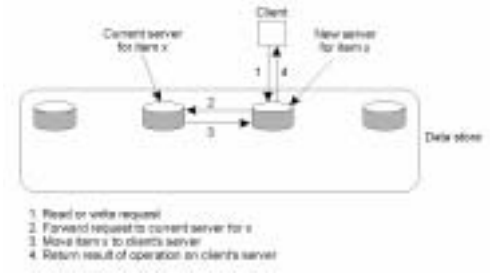
Remote-Write Protocols (2)



The principle of primary-backup protocol.

53

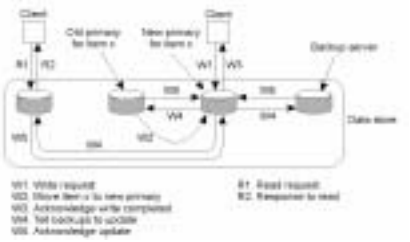
Local-Write Protocols (1)



Primary-based local-write protocol in which a single copy is migrated between processes.

54

Local-Write Protocols (2)



Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

55

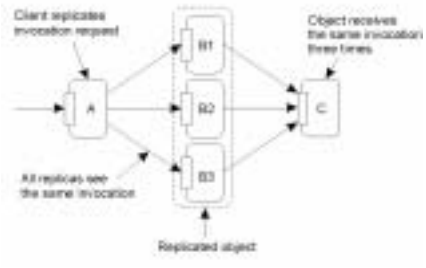
Replicated-write protocols

Write operations can be carried out at multiple replicas instead of only one.

- Active replications
  - An operation is forwarded to all replicas
- Consistency protocols based on majority voting

56

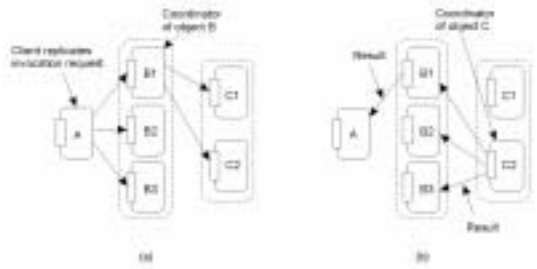
Active Replication (1)



The problem of replicated invocations.

57

Active Replication (2)



a) Forwarding an invocation request from a replicated object.
b) Returning a reply to a replicated object.

58

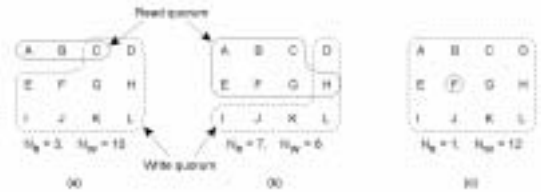
Quorum-Based Protocols (1)

- The basic idea is to require clients to request and acquire the permission of multiple servers before either reading or writing a replicated data item.
- Gifford's scheme:
  - **Nr: read quorum**
  - **Nw: write quorum**

  Two conditions:
  1. Nr+Nw > N
  2. Nw > N/2

59

Quorum-Based Protocols (2)



Three examples of the voting algorithm:
a) A correct choice of read and write set
b) A choice that may lead to write-write conflicts
c) A correct choice, known as ROWA (read one, write all)

60

Cache-Coherence Protocols

- Cache: A special form of replication
- Controlled by clients, not servers
- Three approaches:
- Coherence detection strategy
- Optimistic approach
- Verify whether the cached data were up to date only when the transaction committed.

- Coherence enforcement strategy
- Write-through caches: allow clients to directly modify the cached data and forward the update to the servers.
- Write-back cache: Delay the propagation of updates by allowing multiple writes to take place before informing the servers.

61

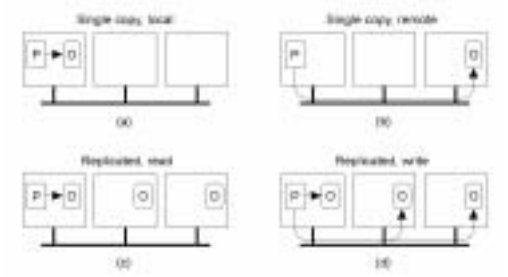
Orca

```
OBJECT IMPLEMENTATION stack;
  top: integer;                              # variable indicating the top
  stack: ARRAY[integer 0..N-1] OF integer    # storage for the stack

  OPERATION push (item: integer)             # function returning nothing
  BEGIN
    GUARD top < N DO
        stack [top] := item;                 # push item onto the stack
        top := top + 1;                      # increment the stack pointer
    OD;
  END;

  OPERATION  pop():integer;                  # function returning an integer
  BEGIN
    GUARD top > 0 DO                         # suspend if the stack is empty
        top := top – 1;                      # decrement the stack pointer
        RETURN stack [top];                  # return the top item
    OD;
  END;
BEGIN
  top := 0;                                  # initialization
END;
```

A simplified stack object in Orca, with internal data and two operations.

62

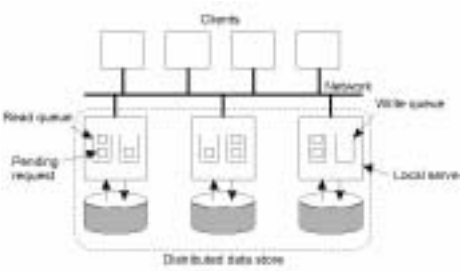
Management of Shared Objects in Orca



Four cases of a process *P* performing an operation on an object *O* in Orca.

63

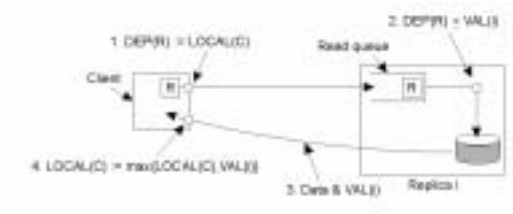
Casually-Consistent Lazy Replication



The general organization of a distributed data store. Clients are assumed to also handle consistency-related communication.

64

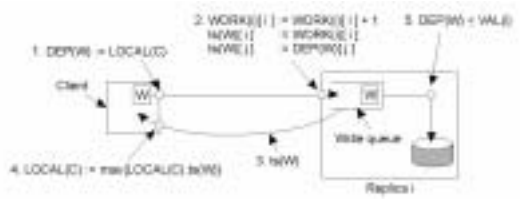
Processing Read Operations



Performing a read operation at a local copy.

65

Processing Write Operations



Performing a write operation at a local copy.

66

Any Questions?

See you next time.