

Consistency and Replication

Chapter 6

Reasons for Replication

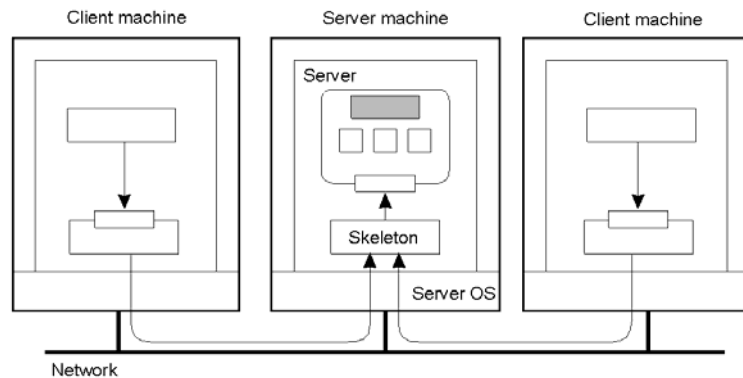
- Reliability
 - Replication helps improve reliability in case of system failures
 - Example: A replicated file system will allow clients to continue to access files in case one of the replica crashes or network connectivity to that replica is lost
- Performance
 - Replication based on load and proximity to clients will improve performance
 - Example: Access to a web site during peak traffic
- There is a price for replication, keeping multiple copies of data up-to-date leads to consistency problems

7/19/2005

2

Object Replication (1)

If objects (or data) are shared, we need to do something about concurrent accesses to guarantee state consistency.

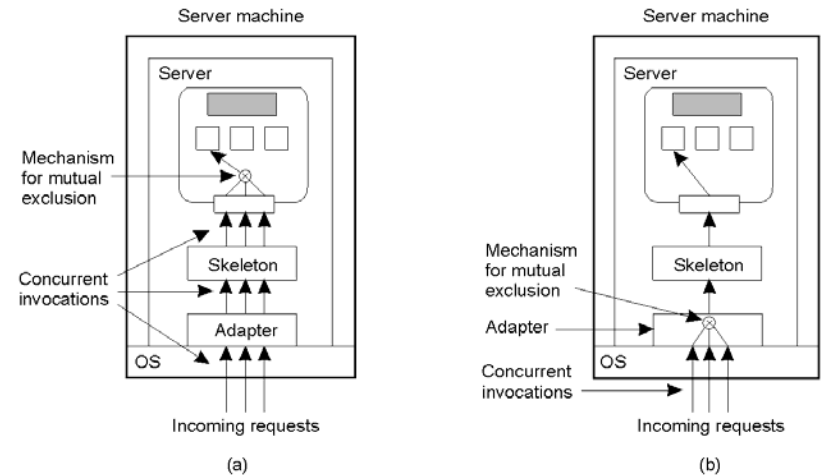


Organization of a distributed remote object shared by two different clients.

7/19/2005

3

Object Replication (2)

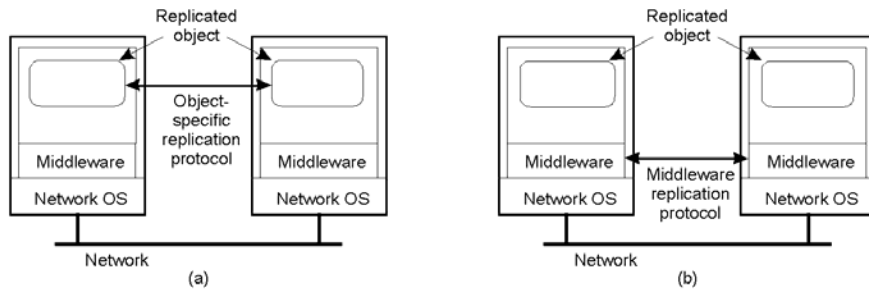


- a) A remote object capable of handling concurrent invocations on its own (e.g., method declared with *synchronized* keyword in Java)
- b) A remote object for which an object adapter is required to handle concurrent invocations (server responsible for concurrency control)

7/19/2005

4

Object Replication (3)



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

Performance and Scalability

- Main issue: To keep replicas consistent, we generally need to ensure that all conflicting operations are done in the same order everywhere
- Conflicting operations: From the world of transactions:
 - Read–Write conflict: a read operation and a write operation act concurrently
 - Write–Write conflicts: two concurrent write operations
- Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability
- Solution: Weaken consistency requirements so that hopefully global synchronization can be avoided

Overview

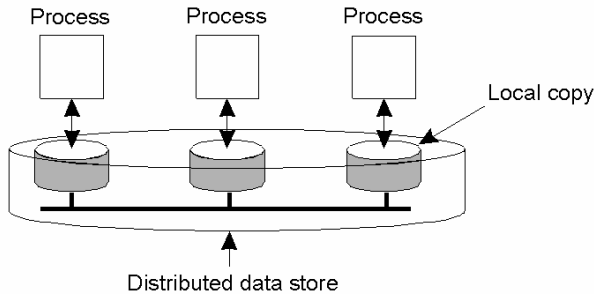
- Consistency Models
 - Data-centric
 - Client-centric
- Distribution protocols
- Consistency protocols

Data-Centric Consistency Models

Data-Centric Consistency Models

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

Essence: A data store is a distributed collection of storages accessible to clients:



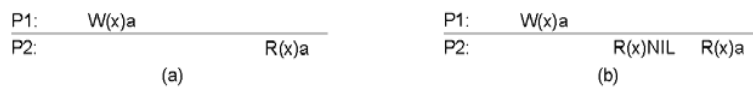
The general organization of a logical data store, physically distributed and replicated across multiple processes.

Data-Centric Consistency Models

- Strong consistency models: Operations on shared data are synchronized:
 - Strict consistency (related to time)
 - Sequential consistency (what we are used to)
 - Causal consistency (maintains only causal relations)
 - FIFO consistency (maintains only individual ordering)
- Weak consistency models: Synchronization occurs only when shared data is locked and unlocked:
 - General weak consistency
 - Release consistency
 - Entry consistency
- Observation: The weaker the consistency model, the easier it is to build a scalable solution.

Strict Consistency

- Any read to a shared data item X returns the value stored by the most recent write operation on X .
- It doesn't make sense to talk about "the most recent" in a distributed environment.
- Assume 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 what you get in the normal sequential case, where your program does not interfere with any other program.
- All writes are instantaneously visible to all processes and an absolute global time order is maintained.



Behavior of two processes, operating on the same data item.

(a) A strictly consistent store.

(b) A store that is not strictly consistent.

Linearizability and Sequential Consistency (1)

- The result of any execution is the same as if the operations of all processes were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program.
- Executions are interleaved, hence there is some total ordering for all operations taken together and time does not play a role
- **Linearizable:** Sequential plus operations are ordered according to a global time.



a) A sequentially consistent data store.

b) A data store that is not sequentially consistent.

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.

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.

Causal Consistency (1)

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

Operations that are not causally related are said to be concurrent.

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

This sequence is allowed with a causally-consistent store, but not with sequentially or strictly consistent store. W(x)a and W(x)c are concurrent operations.

Causal Consistency (2)

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)

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

(b)

A dependency graph of which operation is dependent on which other operation must be constructed and maintained to implement causal consistency.

- a) A violation of a causally-consistent store. If b is computed based on R₂(x)a then W₂(x)b depends on W₂(x)a, so the two writes are causally related and all processes must see them in the same order.
- b) A correct sequence of events in a causally-consistent store. W₁(x)a and W₂(x)b are concurrent writes.

FIFO Consistency (1)

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.

Also called as PRAM consistency (PRAM = Pipelined RAM).

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

A valid sequence of events of FIFO consistency, only the order of $W_2(x)b$ and $W_2(x)c$ are important.

FIFO Consistency (2)

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

FIFO Consistency (3)

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

Two concurrent processes.

- FIFO consistency can lead to non-intuitive results
- In the above example, three outcomes are possible: P1 is killed, P2 is killed, or neither process is killed
- However, with FIFO consistency both processes may be killed if P1 reads R(y)0 before it sees P2's W(y)1 and P2 reads R(x)0 before it sees P1's W(x)1
- With sequentially consistent store this will not happen

Weak Consistency (1)

- Basic idea: You don't care that reads and writes of a series of operations are immediately known to other processes. You just want the effect of the series itself to be known.
- 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.
- Weak consistency implies that we need to lock and unlock data (implicitly or not).

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. Many optimizing compilers postpone the processing of writing back to memory and store the results in registers until there is an explicit reference to that memory location.

7/19/2005

21

Weak Consistency (3)

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

(a)

P1:	W(x)a	W(x)b	S		
P2:				S	R(x)a

(b)

- A valid sequence of events for weak consistency. Since P2 and P3 did not synchronize before read there is no guarantee no the values they see.
- An invalid sequence for weak consistency. $R_2(x)a$ is not correct since after synchronization x should be set to b.

7/19/2005

22

Release Consistency

- 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).

P1:	Acq(L)	W(x)a	W(x)b	Rel(L)	
P2:				Acq(L)	R(x)b
P3:					R(x)a

A valid event sequence for release consistency. Since P3 did not perform an acquire and release, $R_3(x)$ in P3 returns a.

7/19/2005

23

Entry Consistency (1)

- Where release consistency affects all shared data, entry consistency affects only those shared data associated with a synchronization variable.
- 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.

7/19/2005

24

Entry Consistency (1)

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

A valid event sequence for entry consistency.

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.

Client-Centric Consistency Models

Client-Centric Consistency Models

- **Goal:** Show how we can perhaps avoid systemwide consistency, by concentrating on what specific clients want, instead of what should be maintained by servers.
- **Background:** Most large-scale distributed systems (i.e., databases) apply replication for scalability, but can support only weak consistency:
- **DNS:** Updates are propagated slowly, and inserts may not be immediately visible.
- **NEWS:** Articles and reactions are pushed and pulled throughout the Internet, such that reactions can be seen before postings.
- **Lotus Notes:** Geographically dispersed servers replicate documents, but make no attempt to keep (concurrent) updates mutually consistent.
- **WWW:** Caches all over the place, but there need be no guarantee that you are reading the most recent version of a page.

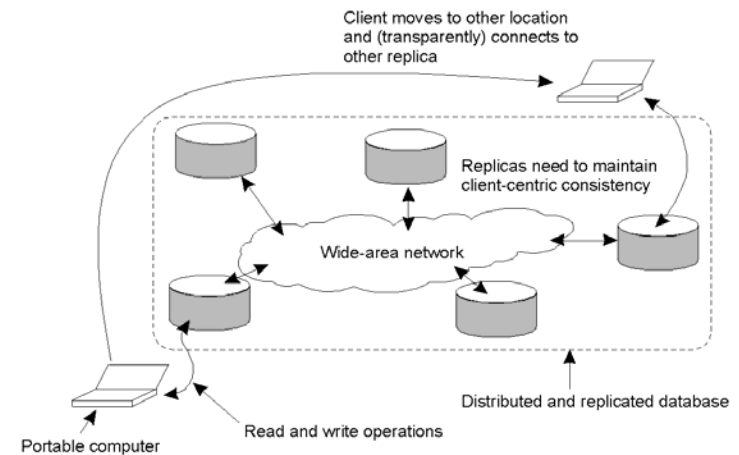
Consistency for Mobile Users

- Example: Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database.
 - At location A you access the database doing reads and updates.
 - At location B you continue your work, but unless you access the same server as the one at location A, you may detect inconsistencies:
 - your updates at A may not have yet been propagated to B
 - you may be reading newer entries than the ones available at A
 - your updates at B may eventually conflict with those at A
- Note: The only thing you really want is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, the database will appear to be consistent to you.

7/19/2005

29

Eventual Consistency



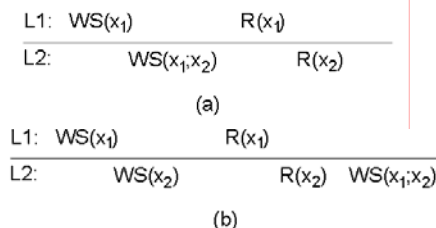
The principle of a mobile user accessing different replicas of a distributed database.

7/19/2005

30

Monotonic Reads

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 or a more recent value.



WS(x_i[t]) is the set of write operations at L_i that lead to version x_i of x (at time t); WS(x_i[t₁];x_j[t₂]) indicated that it is known that WS(x_i[t₁]) is part of WS(x_j[t₂]).

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 since there are no guarantees that this set contains all operations contained in WS(x₁).

Examples: Automatically reading your personal calendar updates from different servers. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

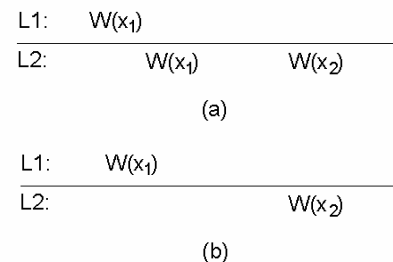
Reading (not modifying) incoming mail while you are on the move. Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.

7/19/2005

31

Monotonic Writes

A write operation by a process on a data item x is completed before any successive write operation on x by the same process (similar to data-centric FIFO consistency).



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.

Examples: Updating a program at server S_2 , and ensuring that all components on which compilation and linking depends, are also placed at S_2 .

Maintaining versions of replicated files in the correct order everywhere (propagate the previous version to the server where the newest version is installed).

7/19/2005

32

Read Your Writes

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).

L1:	$W(x_1)$	
<hr/>		
L2:	$WS(x_1, x_2)$	$R(x_2)$

(a)

L1:	$W(x_1)$	
<hr/>		
L2:	$WS(x_2)$	$R(x_2)$

(b)

- a) A data store that provides read-your-writes consistency.
- b) A data store that does not since the effects of the previous write operation are not propagated to L_2 .

Example: Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

Writes Follow Reads

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.

L1:	$WS(x_1)$	$R(x_1)$
<hr/>		
L2:	$WS(x_1, x_2)$	$W(x_2)$

(a)

L1:	$WS(x_1)$	$R(x_1)$
<hr/>		
L2:	$WS(x_2)$	$W(x_2)$

(b)

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

Example: See reactions to posted articles only if you have the original posting (a read “pulls in” the corresponding write operation).