



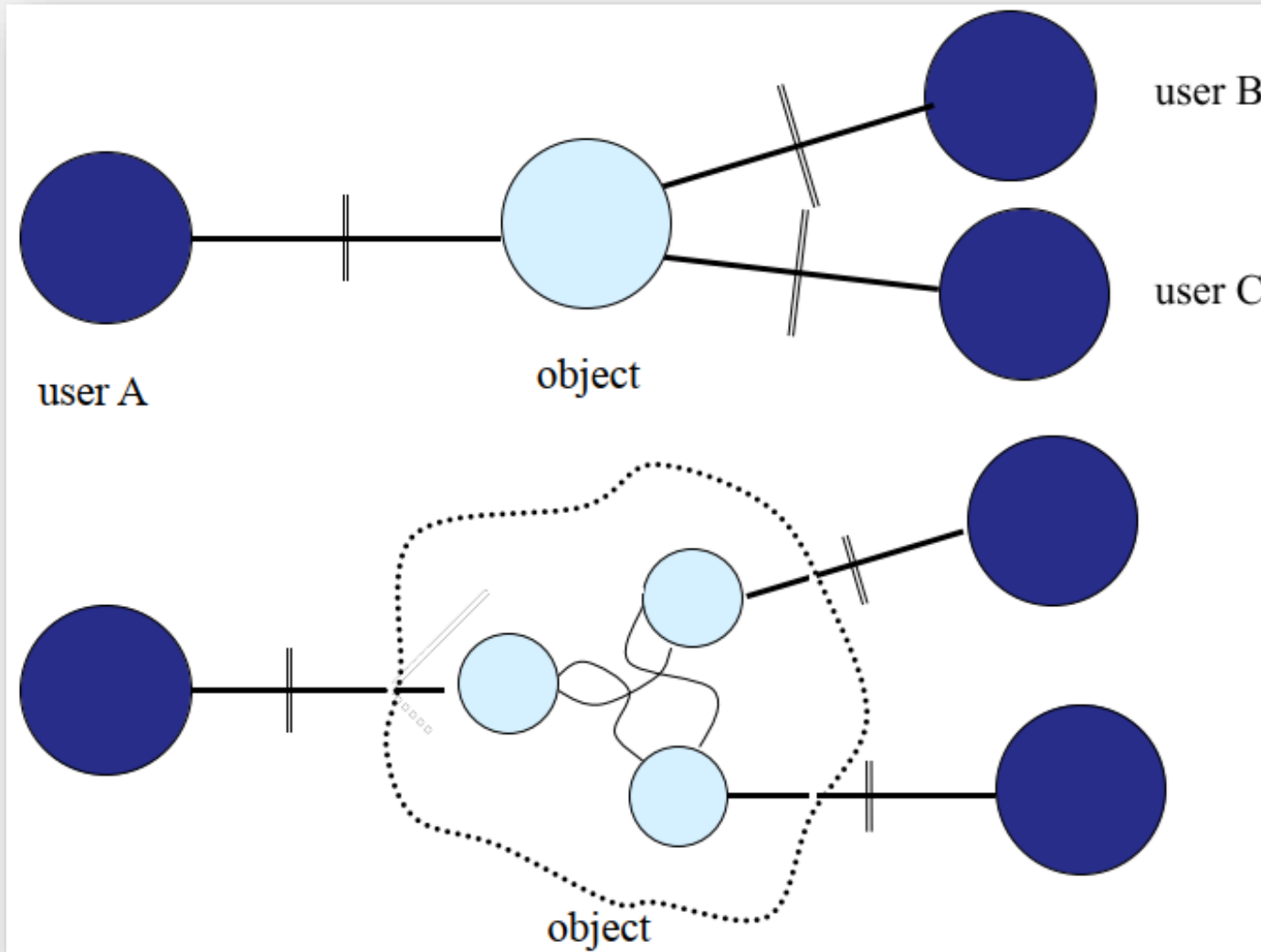
Parallel and Distributed Computing

Dr. Ali Sayyed
Department of Computer Science
National University of Computer & Emerging Sciences



Consistency And Replication

Data Replication



Reasons for Replication

There are two primary reasons for replicating data.

- **First, data are replicated to increase the reliability of a system.**
 - The system works if one system crashes
 - Also, by maintaining multiple copies, it becomes possible to provide better protection against corrupted data.
- **The other reason for replicating data is performance.**
 - when a distributed system needs to scale in terms of size or in terms of the geographical area it covers.
 - performance can be improved by replicating the server and subsequently dividing the workload.


Challenges in Replication

- Unfortunately, there is a **price to be paid** when data are replicated.
- Multiple copies may lead to **consistency problems**
- Consider improving access times to Web pages.
 - To improve performance, Web browsers often **locally store (cache)** a copy of a previously fetched Web page.
 - If a user requires that page again, the browser automatically returns the local copy.
 - The problem is that if the page has been modified in the meantime, **modifications will not have been propagated** to cached copies.



Challenges in Replication



- **Keeping track of all replicas and updating them is not simple**
 - Difficulties come from the fact that we need **to synchronize all replicas**.
 - Replicas may need to decide on **a global ordering** of operation and global synchronization simply takes a lot of communication time, especially when replicas are spread across a wide-area network.
 - We are now faced with a dilemma.
 - **On the one hand**, scalability problems can be reduced by applying replication.
 - **On the other hand**, to keep all copies consistent requires global synchronization, which is costly in terms of performance.
- 




A way out



- In many cases, the only real solution is **to relax the consistency constraints**.
- In other words, we relax **the requirement of (instantaneous) global synchronizations**.
- **The price paid is that copies may not always be the same everywhere.**
- To what extent consistency can be relaxed depends highly on the access and update patterns of the replicated data, as well as on the purpose for which those data are used.
- There are a range of consistency models and many different ways to implement models through what are called **distribution and consistency protocols**.

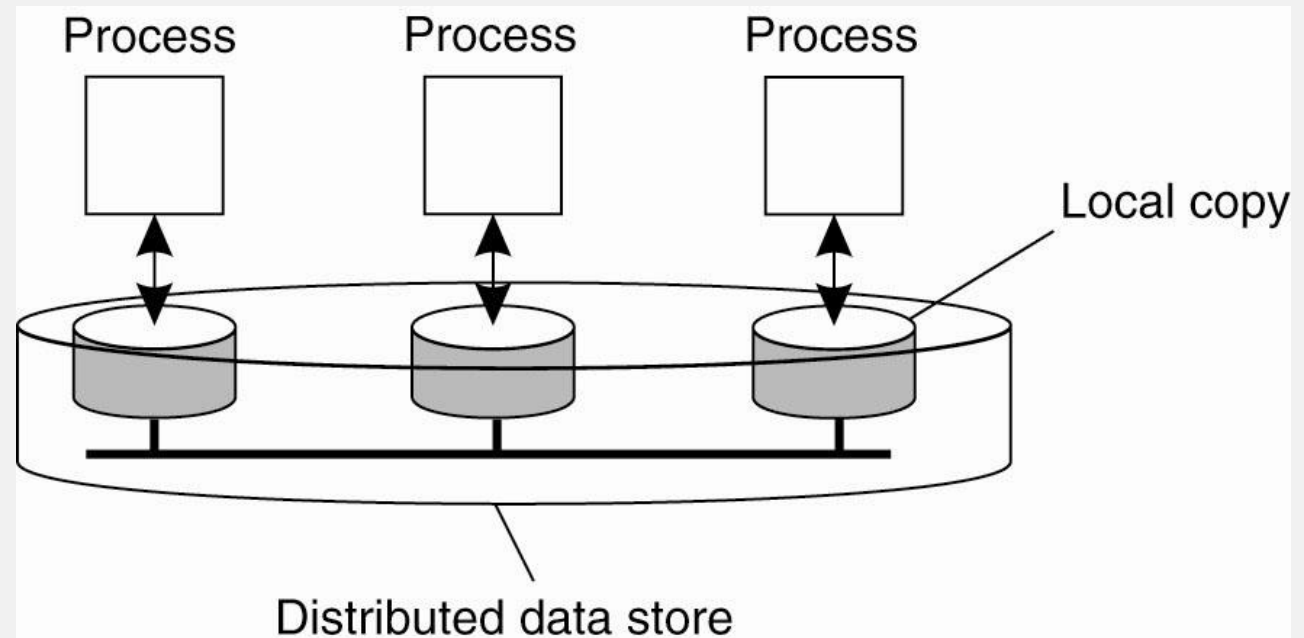


Consistency Model

- A **consistency model** is a **set of rules** or guidelines that determine how a distributed system should behave in terms of the ordering and visibility of updates made to shared data.
 - It is essentially a **contract between processes and the data store**.
 - It says that if processes agree to obey certain rules, the store promises to work correctly.
 - In a distributed system, where multiple nodes can access and modify the same data, it's important to ensure that all nodes have a consistent view of the data to avoid conflicts and inconsistencies. So, a consistency model defines the level of consistency required for the system and specifies the mechanisms used to achieve it.
- 

Data-centric Consistency Models

- Traditionally, consistency has been discussed in the context of **read and write operations** on shared data, available by means of (distributed) shared memory, a (distributed) shared database, or a (distributed) file system.
- Here, we use the broader term **data store**. A data store may be physically distributed across multiple machines.



Consistency Notations

- We will use a notation in which we draw **the operations of a process along a time axis**.
- The time axis is always drawn horizontally, with time increasing from left to right.
- **Wi(x)a** denote that process P_i writes value a to data item x .
- **Ri(x)b** denotes that process P_i reads x and is returned the value b .

| | |
|-------|------------------|
| P1: | W(x)a |
| <hr/> | |
| P2: | R(x)NIL R(x)a |

Strong Consistency Model

- Strict consistency is the **strongest consistency model**. Under this model, a write operation by any process needs to be seen instantaneously by all processes.
- One popular example is password update of someone's bank account.
- It's the model a bank machine uses when dispensing cash from your account - you get your cash, your account gets debited.
- Behavior of two processes, operating on the same data item.
 - a) A strictly consistent store.
 - b) A store that is not strictly consistent.

| | | |
|-----|-------|-------|
| P1: | W(x)a | |
| P2: | | R(x)a |

(a)

| | | |
|-----|-------|---------|
| P1: | W(x)a | |
| P2: | | R(x)NIL |

(b)

implementation
requires absolute
global time



Continuous Consistency



- There are different ways to specify what inconsistencies a data store can tolerate.
- We can define **three independent axes for inconsistencies**:
 - deviation in numerical values between replicas
 - deviation in staleness between replicas
 - deviation with respect to the ordering of update operations.
- If data have **numerical semantics**. One example is the replication of records containing stock market prices. In this case, an application may specify that two copies should not deviate more than \$0.5
- **Staleness deviations** relate to the freshness or when last time a replica was updated. For example, weather reports typically stay reasonably accurate over some time, say a few hours.



Continuous Consistency



- Finally, there are classes of applications in which the **ordering of updates are allowed to be different at the various replicas, as long as the differences remain bounded.**
- One way of looking at these updates is that they are **applied tentatively to a local copy, awaiting global agreement from all replicas.** As a consequence, some updates may need to be **rolled back** and applied in a different order before becoming permanent.
- Intuitively, ordering deviations are much harder to implement than the other two consistency metrics.



Continuous Consistency

Order Deviation

- A has three tentative operations pending to be committed
- B has one tentative operations pending to be committed

Numerical Deviation

- The numerical deviation at a replica R consists of two components
- The number of operations at all other replicas that have not yet been seen by R, along with the sum of corresponding missed values

Replica A

Conit

d = 558 // distance
g = 95 // gas
p = 78 // price

| Operation | Result |
|---------------------|-------------|
| < 5, B> g ← g + 45 | [g = 45] |
| < 8, A> g ← g + 50 | [g = 95] |
| < 9, A> p ← p + 78 | [p = 78] |
| <10, A> d ← d + 558 | [d = 558] |

Vector clock A = (11, 5)
Order deviation = 3
Numerical deviation = (2, 482)

Replica B

Conit

d = 412 // distance
g = 45 // gas
p = 70 // price

| Operation | Result |
|---------------------|-------------|
| < 5, B> g ← g + 45 | [g = 45] |
| < 6, B> p ← p + 70 | [p = 70] |
| < 7, B> d ← d + 412 | [d = 412] |

Vector clock B = (0, 8)
Order deviation = 1
Numerical deviation = (3, 686)



Continuous Consistency



- We may restrict order deviation by specifying an acceptable **maximal value**.
- Likewise, we may want two replicas to never **numerically deviate** by more than **1000 units**.
- Having such consistency schemes does require that a **replica knows how much it is deviating from other replicas**, implying that we need **separate communication** to keep replicas informed.
- The underlying assumption is that such communication is much less expensive than communication to keep replicas synchronized.

FIFO Consistency

- 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.

| | | | | |
|-----|-------|-------|-------|-------------|
| 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
- Guarantee:**
 - writes from a single source must arrive in order
 - no other guarantees.



Sequential Consistency



- **A data store is sequentially consistent when:**
- The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in the same sequence
- Sequential consistency implies that operations appear to take place in some total order, and that order is consistent on each individual process.

Sequential Consistency

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

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

(a)

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

(b)

When you view the posts of a specific friend on Facebook, you expect to see their posts in the order they were posted. This means that if your friend posted "A" before "B", you should see "A" before "B" when you view their timeline. However, the overall timeline on Facebook, which combines posts from all your friends and possibly other sources, might not adhere to a sequential consistency. This is because posts from different friends, pages, or groups can be mixed together, and they may not always appear in strict chronological order across the entire timeline.



Causal Consistency



- Causal Consistency ensures that operations that are causally related appear in the same order on all nodes, while operations that are independent can appear in different orders across different nodes.
- **This model** makes a distinction between events that are potentially causally related and those that are not.
 - If event b is caused by an earlier event a, causality requires that **everyone else first see a, then see b.**
- Suppose that process **P1** writes a data item **x**. Then **P2** reads **x** and writes **y**. Here the reading of x and the writing of y are potentially causally related because the computation of y may have depended on the value of x.



Causal Consistency



- Distributed Social Media (Facebook, Twitter, Instagram)
 - Post-Reaction Relationship:
 - If A asks, “shall we have lunch?”, and B & C respond with “yes”, and “no”, respectively.
 - Causal consistency allows A to see “lunch?”, “yes”, “no”; and B to observe “lunch?”, “no”, “yes”.
 - However, no participant ever observes “yes” or “no” prior to the question “lunch?”.
- In a multiplayer online game, If Player A shoots at Player B and Player B takes damage, all players must see the shot before the damage event occurs.

Causal Consistency

- This sequence is allowed with a causally-consistent store.

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

- The thing to note is that the writes $W_2(x)b$ and $W_1(x)c$ are concurrent, so it is not required that all processes see them in the same order.

Causal Consistency

- (a) A violation of a causally-consistent store.

P1 : $W(x) a$

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

P3 : $R(x) a$ $R(x) b$ ✓

P4 : $R(x) b$ $R(x) a$ ✗

- $W_2(x) b$ potentially depending on $W_1(x) a$ because writing the value b into x may be a result of a computation involving the previously read value by $R_2(x) a$.

A Causal Consistent System

P1 : $W(x)a$

P2 : $W(x)b$

P3 : $R(x)a$ $R(x)b$ ✓

P4 : $R(x)b$ $R(x)a$ ✓

- $W_1(x)a$ and $W_2(x)b$ are concurrent