REPLICATION AND CONSISTENCY

There are primarily two reasons for replicating data:

i. Improving the reliability of a distributed system and
ii. Improving performance.

Replication introduces a consistency problem: whenever a replica is updated, that replica becomes different from the others.
To keep replicas consistent, we need to propagate updates in such a way that temporary inconsistencies are not noticed. Unfortunately, doing so may severely degrade performance, especially in large-scale distributed systems.

Informally, this means that when one copy is updated we need to ensure that the other copies are updated as well; otherwise the replicas will no longer be the same.

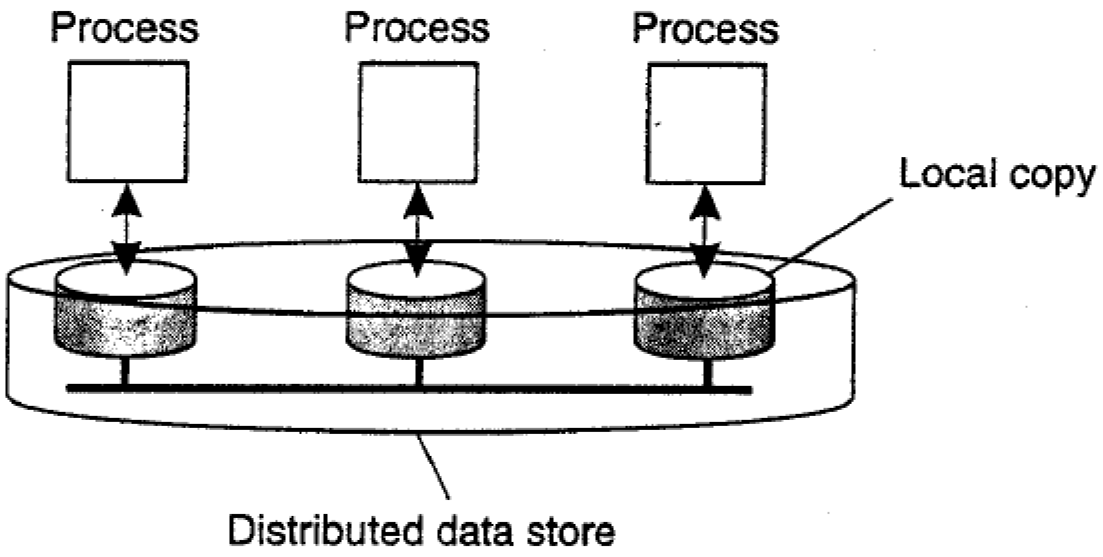
CONSISTENCY MODELS

A **consistency model** 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.

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.

In this section, we use the broader term **data store**. A data store may be physically distributed across multiple machines. In particular, each process that can access data from the store is assumed to have a local (or nearby) copy available of the entire store. Write operations are propagated to the other copies as shown.



A **data operation** is classified as a write operation when it changes the data, and is otherwise classified as a **read operation**.

Normally, a process that performs a read operation on a data item, expects the operation to return a value that shows the results of the last write operation on that data. In the absence of a global clock, it is difficult to define precisely which write operation is the last one.

As an alternative, we need to provide other definitions, leading to a **range of consistency models**. Each model effectively restricts the values that a read operation on a data item can return.

As is to be expected, the ones with major restrictions are easy to use, for example when developing applications, whereas those with minor restrictions are sometimes difficult. The tradeoff is, of course, that the easy-to-use models do not perform nearly as well as the difficult ones. Such is life.

1. Continuous Consistency

From what we have discussed so far, it should be clear that there is no such thing as a best solution to **replicating data**. Replicating data poses **consistency problems** that cannot be solved efficiently in a general way. Only if we loosen consistency can there be hope for attaining efficient solutions. Unfortunately, there are also no general rules for loosening consistency: exactly what can be tolerated is highly dependent on applications.

There are different ways for applications to specify what inconsistencies they can tolerate. Yu and Vahdat (2002) take a general approach by distinguishing three independent axes for defining inconsistencies:

i. Deviation in numerical values between replicas,
ii. Deviation in staleness between replicas, and
iii. Deviation with respect to the ordering of update operations.

They refer to these deviations as forming continuous consistency ranges.
Measuring inconsistency in terms of **numerical deviations** can be used by applications for which the data have numerical semantics. Numerical deviation can be understood in terms of the number of updates that have been applied to a given replica but have not yet been seen by others.
For example, a Web cache may not have seen a batch of operations carried out by a Web server. In this case, the associated deviation in the *value* is also referred to as its ***weight.***

Staleness deviations relate to the last time a replica was updated. For some applications, it can be tolerated that a replica provides old data as long as it is not *too* old. For example, weather reports typically stay reasonably accurate over some time, say a few hours. In such cases, a main server may receive timely updates, but may decide to propagate updates to the replicas only once in a while.

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 grasp than the other two consistency metrics.

2. Consistent Ordering of Operations

Besides continuous consistency, there is a huge body of work on data-centric consistency models from the past decades. An important class of models comes from the field of concurrent programming. Confronted with the fact that in parallel and distributed computing multiple processes will need to share resources and access these resources simultaneously, researchers have sought to express the semantics of concurrent accesses when shared resources are replicated.

This has led to at least one important consistency model that is widely used. In the following,
we concentrate on what is known as **sequential consistency**, and we will also discuss a weaker variant, namely **causal consistency**.

Sequential Consistency

A data store is said to be sequentially consistent when it satisfies the following condition:
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 this sequence in the order specified by its program.

Causal Consistency

For a data store to be considered causally consistent, it is necessary that the store obeys the following condition:
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.

Implementing causal consistency requires keeping track of which processes have seen which writes. It effectively means that a dependency graph of which operation is dependent on which other operations must be constructed and maintained.

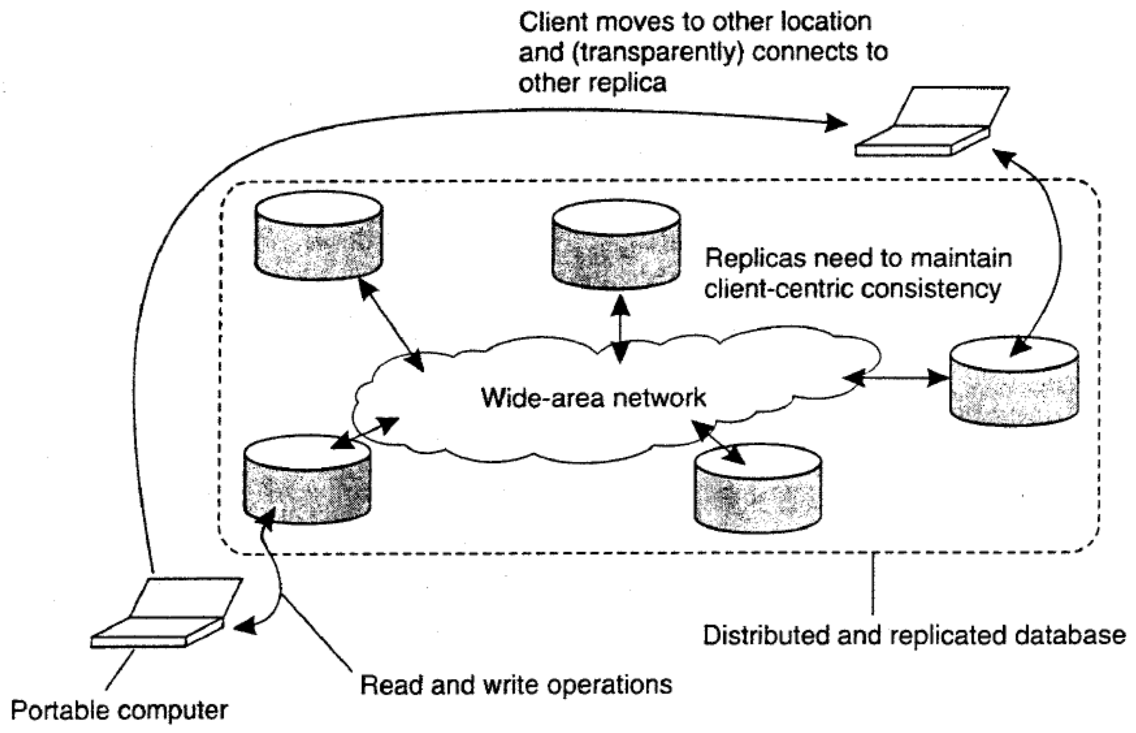
CLIENT-CENTRIC CONSISTENCY MODELS

As opposed to these data-centric models, researchers in the field of distributed databases for mobile users have defined a number of client-centric consistency models.

Such models do not consider the fact that data may be shared by several users, but instead, concentrate on the consistency that an individual client should be offered.

The underlying assumption is that a client connects to different replicas in the course of time, but that such differences should be made transparent.

In essence, client-centric consistency models ensure that whenever a client connects to a new replica, that replica is brought up to date with the data that had been manipulated by that client before, and which may possibly reside at other replica sites.


Client moves to other location and (transparently) connects to other replica
Replicas need to maintain client-centric consistency
Wide-area network
Distributed and replicated database
Read and write operations
Portable computer

Consistency protocols
Consistency protocols describe specific implementations of consistency models. With respect to **sequential consistency** and its variants, a distinction can be made between **primary-based protocols** and **replicated-write protocols**.

In **primary-based protocols**, all update operations are forwarded to a primary copy that subsequently ensures the update is properly ordered and forwarded.

In **replicated-write protocols**, an update is forwarded to several replicas at the same time. In that case, correctly ordering operations often becomes more difficult.

Cache-Coherence Protocols
Caches form a special case of replication, in the sense that they are generally controlled by clients instead of servers. However, cache-coherence protocols, which ensure that a cache is consistent with the server-initiated replicas are, in principle, not very different from the consistency protocols discussed so far.