Distributed Systems

Chapter 7

Consistency and Replication

Consistency and replication

- An important issue in distributed systems is the replication of data.
- Data are generally replicated to enhance reliability or improve performance.
- 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.
- In this chapter, we take a detailed look at what consistency of replicated data actually means and the various ways that consistency can be achieved.

Reasons for Replication

1. Performance Enhancement

The copy of data is placed in multiple locations, so client can get the data from nearby location. This decreases the time take to access data .It enhances performance of the distributed system. Multiple servers located at different locations provided the same service to the client. It allows parallel processing of the client's request to the resource.

2. Increase availability

Replication is a technique for automatically maintaining the availability of data despite server failure.

3. Fault tolerance

If a server fails , the data can be accessed from other servers.

Challenges in Replication

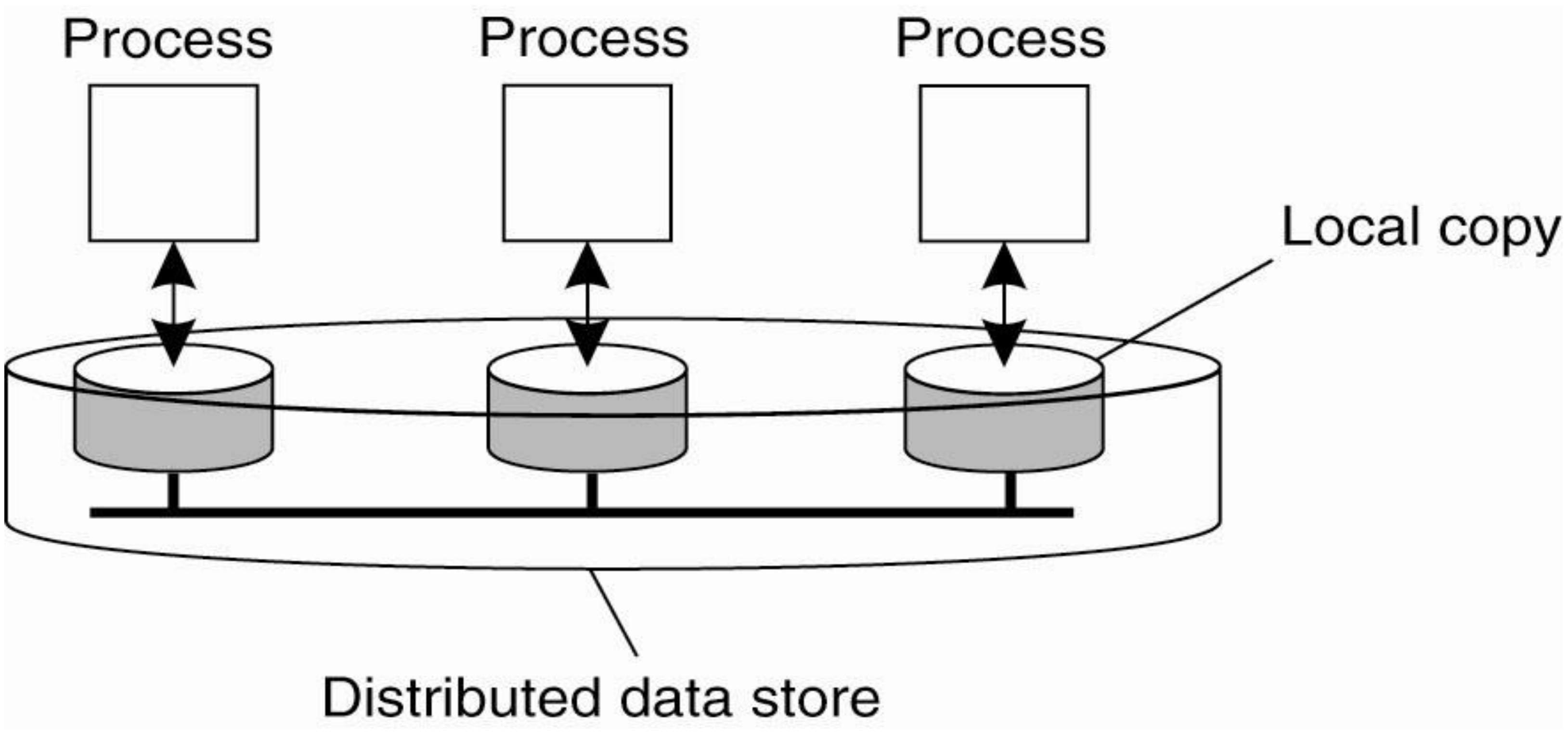
1. Placement (where to place replication)
   - permanent replication: permanent replicas consist of cluster of servers that may be geographically dispersed.
   - Server initiated replication :server initiated caches including placing replicas in the hosting servers and server caches.
   - Client initiated replicas: client initiated replicas include web browsers cache.

2 . Propagation of updates among replicas

3.Lack of consistency

It makes some time for all the copies to be consistent.

Data-centric Consistency Models



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

What is a Consistency Model?

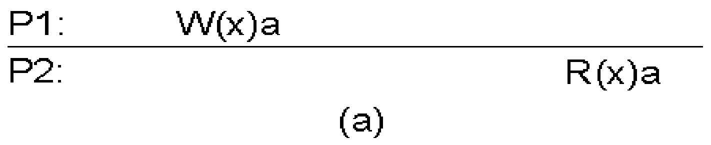
- A “consistency model” is a CONTRACT between a DS data-store and its processes.
- If the processes agree to the rules, the data-store will perform properly and as advertised.
- We start with *Strict Consistency*, which is defined as:
  - *Any read on a data item ‘x’ returns a value corresponding to the result of the most recent write on ‘x’ (regardless of where the write occurred).*

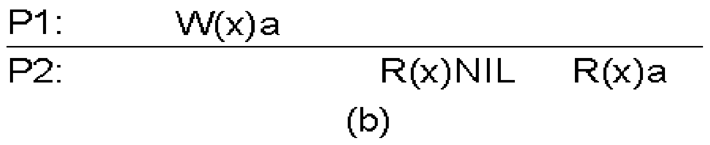
Consistency Model Diagram Notation

- W_i(x)a – a write by process ‘i’ to item ‘x’ with a value of ‘a’. That is, ‘x’ is set to ‘a’.

  (Note: The process is often shown as ‘P_i’).
- R_i(x)b – a read by process ‘i’ from item ‘x’ producing the value ‘b’. That is, reading ‘x’ returns ‘b’.
- Time moves from left to right in all diagrams.

Strict Consistency



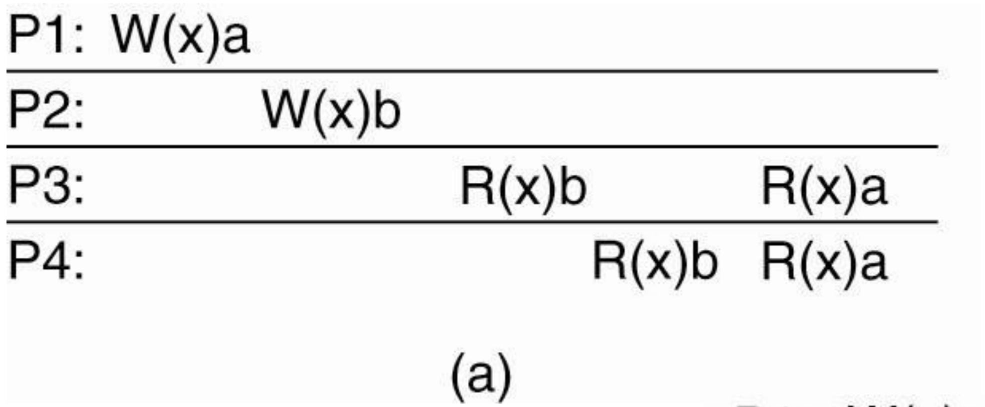


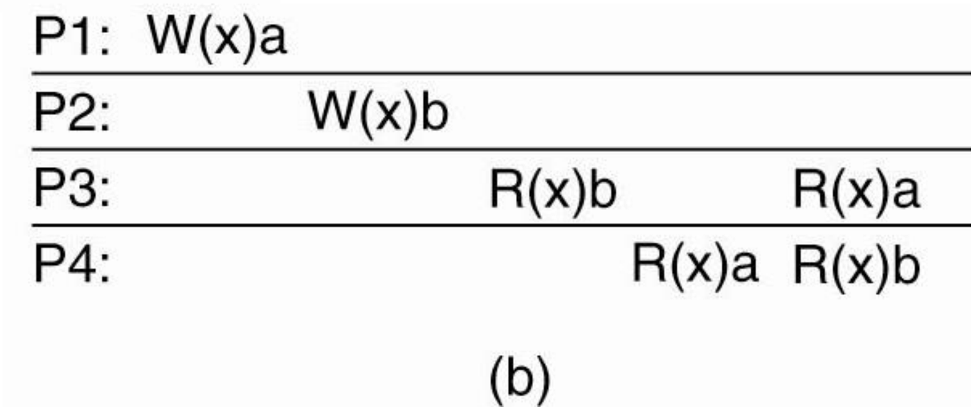
- Behavior of two processes, operating on same data item:
  - a) A strictly consistent data-store.
  - b) A data-store that is not strictly consistent.

Sequential Consistency (1)

- A weaker consistency model, which represents a relaxation of the rules.
- It is also much easier (possible) to implement.
- Definition of "Sequential Consistency":
  - *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 the same sequential order and the operations of each individual process appear in this sequence in the order specified by its program.*

Sequential Consistency (2)

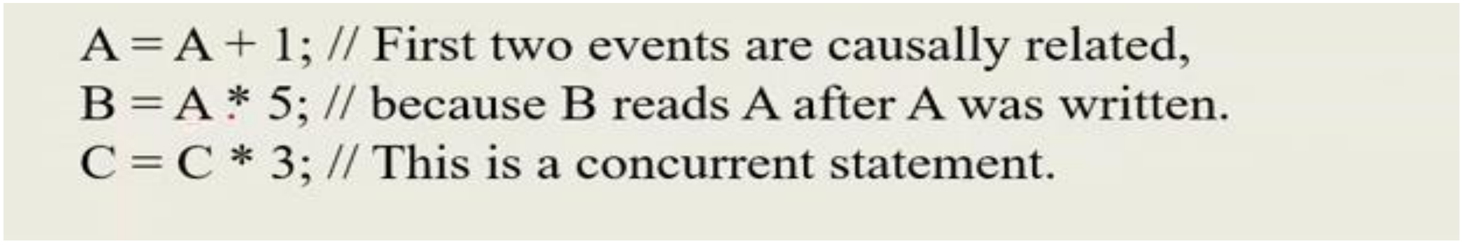




(a) A sequentially consistent data store.
(b) A data store that is not sequentially consistent.

Causal Consistency

- This model distinguishes between events that are “causally related” and those that are not.
- *If event B is caused or influenced by an earlier event A, then causal consistency requires that every other process see event A, then event B.*
- *Weakening of sequential consistency*
- *Causally not related is called concurrently*

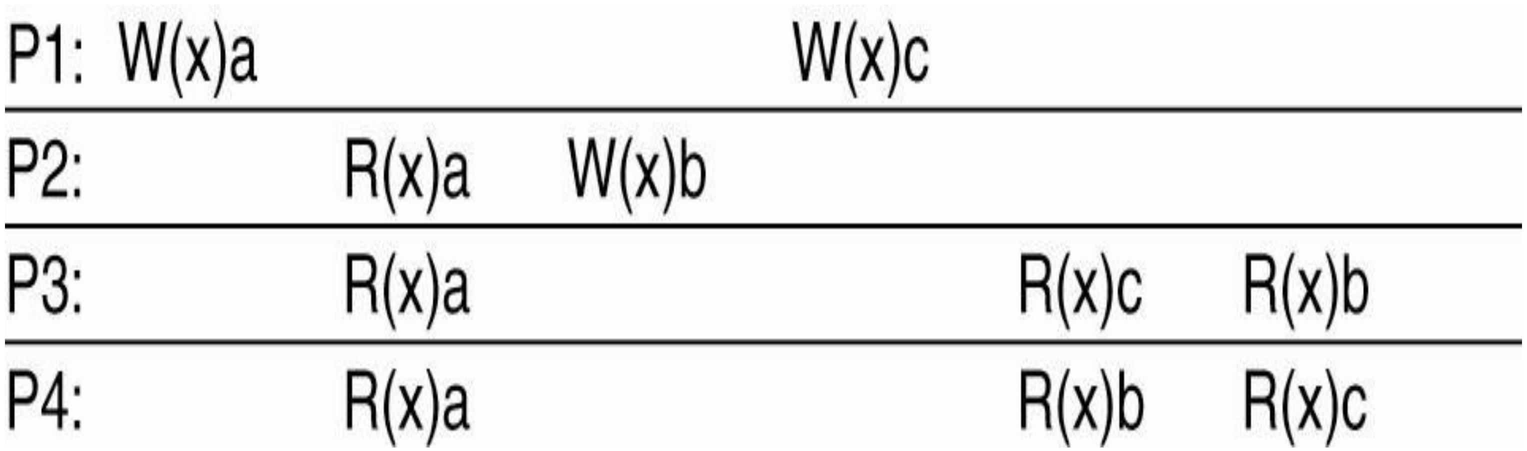


```
A = A + 1; // First two events are causally related,
B = A * 5; // because B reads A after A was written.
C = C * 3; // This is a concurrent statement.
```

Causal Consistency (1)

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

Causal Consistency (4)



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

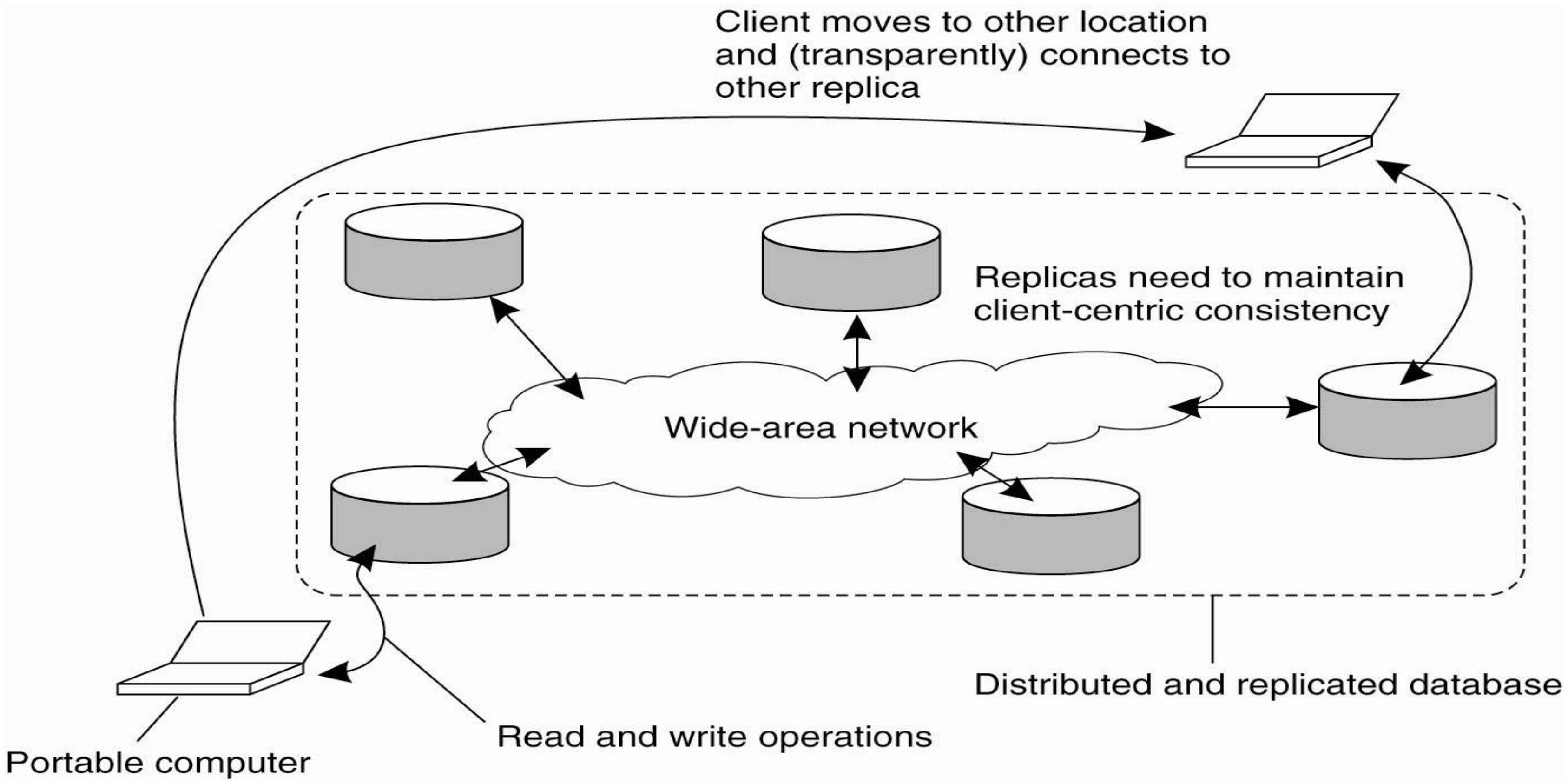
Client-Centric Consistency Models

- The previously studied consistency models concern themselves with maintaining a consistent (globally accessible) data-store in the presence of concurrent read/write operations
- Another class of distributed data-store is that which is characterized by *the lack of simultaneous updates*. Here, the emphasis is more on maintaining a consistent view of things *for the individual client process* that is currently operating on the data-store.

Toward Eventual Consistency

- The only requirement is that all replicas will *eventually* be the same.
- All updates must be guaranteed to propagate to all replicas … *eventually*!
- This works well if every client always updates the same replica.
- Things are a little difficult if the clients are *mobile*.

Eventual Consistency


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

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

Monotonic Reads (1)

- A data store is said to provide monotonic-read consistency if the following condition holds:
  - 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 value or a more recent value.

Monotonic Writes (1)

- In a monotonic-write consistent store, the following condition holds:
  - A write operation by a process on a data item x
    - is completed before any successive write operation on x by the same process.

Read Your Writes (1)

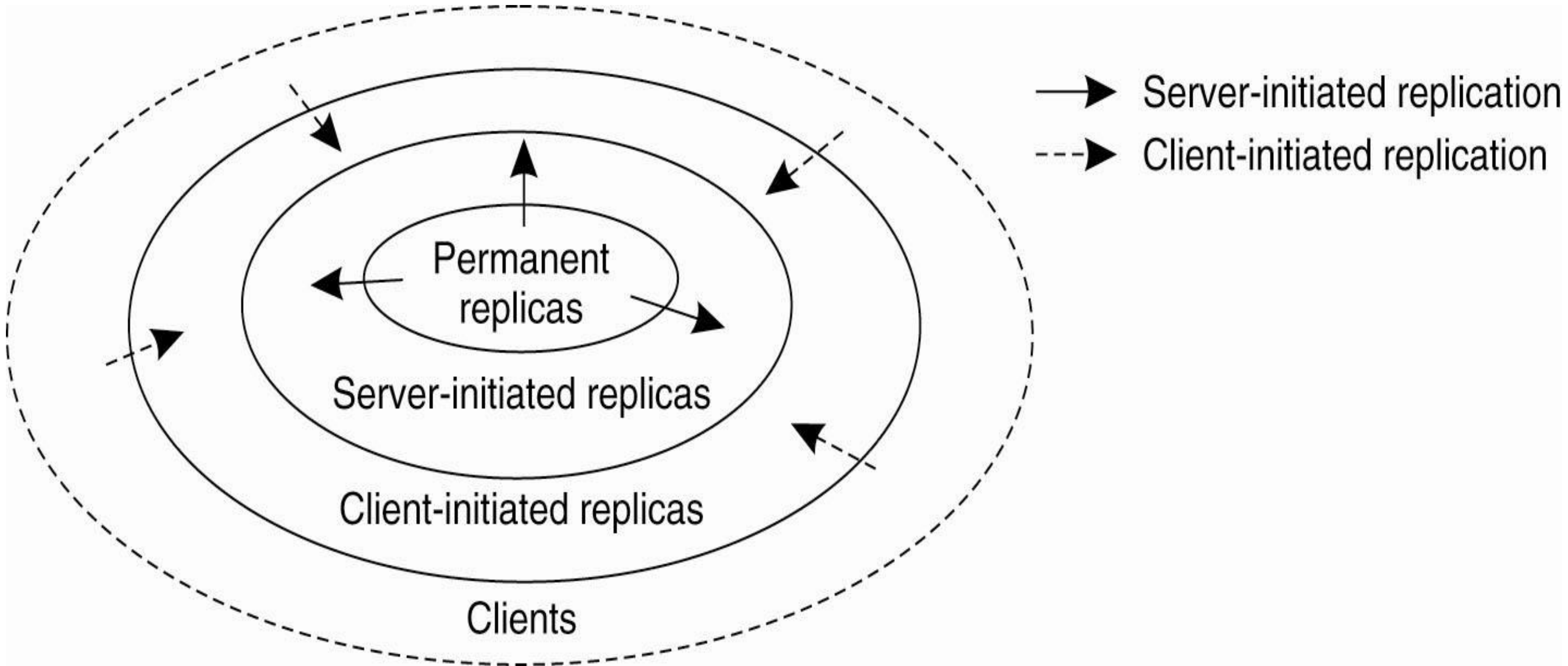
- A data store is said to provide read-your-writes consistency, if the following condition holds:
  - The effect of a write operation by a process on data item x
    - will always be seen by a successive read operation on x by the same process.
    - (After write you read – eg change password, update webpage)

Writes Follow Reads (1)

- A data store is said to provide writes-follow-reads consistency, if the following holds:
  - 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.
    - ➢ Example read blog and then review,reaction to posted article

Content Replication and Placement(Replica Management)

- *Regardless of which consistency model is chosen, we need to decide **where**, **when*** and ***by whom*** copies of the data-store are to be placed.



Server-initiated replication
Client-initiated replication
Permanent replicas
Server-initiated replicas
Client-initiated replicas
Clients

Replica Placement Types

- There are three types of replica:

1. ***Permanent replicas***: tend to be small in number, organized as Clusters of Workstations or mirrored systems.
2. ***Server-initiated replicas***: used to enhance performance at the initiation of the owner of the data-store. Typically used by web hosting companies to geographically locate replicas close to where they are needed most. (Often referred to as “push caches”).
3. ***Client-initiated replicas***: created as a result of client requests – think of browser caches. Works well assuming, of course, that the cached data does not go *stale* too soon.

Consistency Protocols

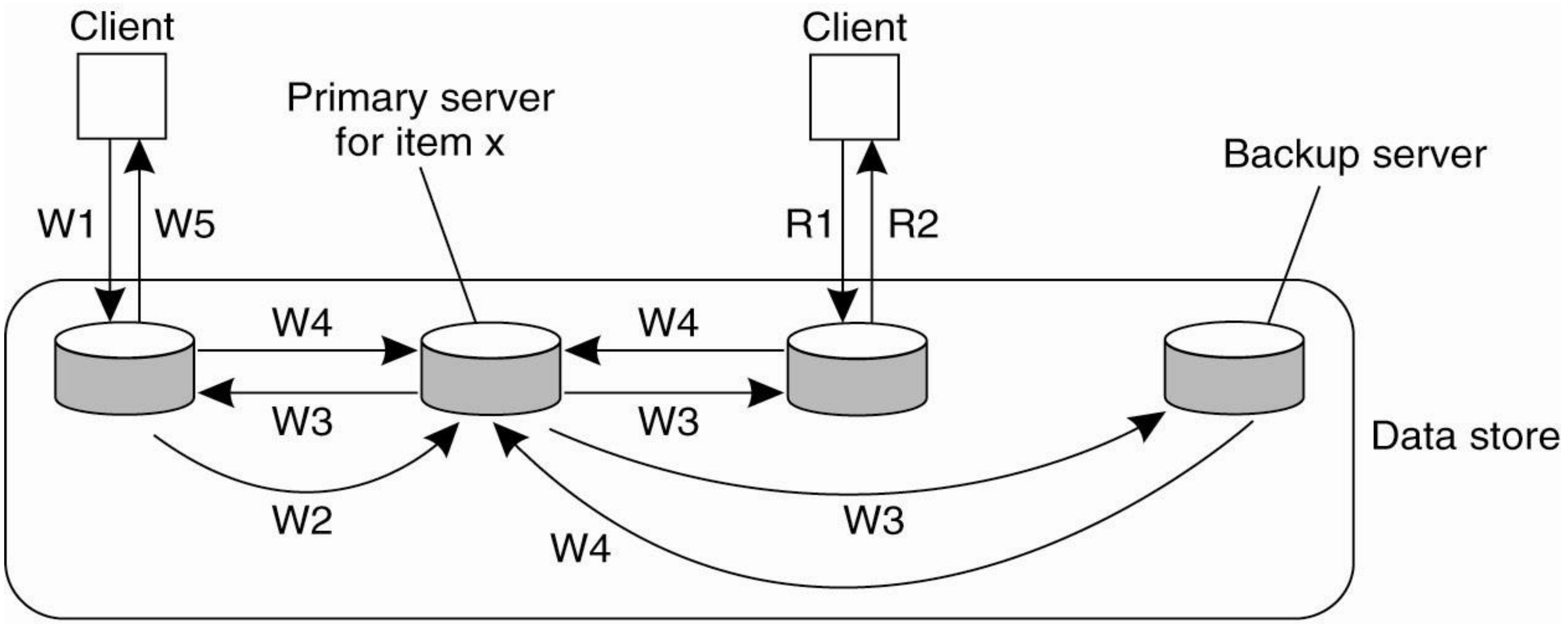
- This is a specific implementation of a consistency model.
- The main approaches are as follows:
  - Primary based protocols
  - Replicated write protocol
  - Cache coherence protocol

Primary-Based Protocols

- Each data item is associated with a “primary” replica.
- The primary is responsible for coordinating writes to the data item.
- There are two types of Primary-Based Protocol:

1. Remote-Write
2. Local-Write

Remote-Write Protocols



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

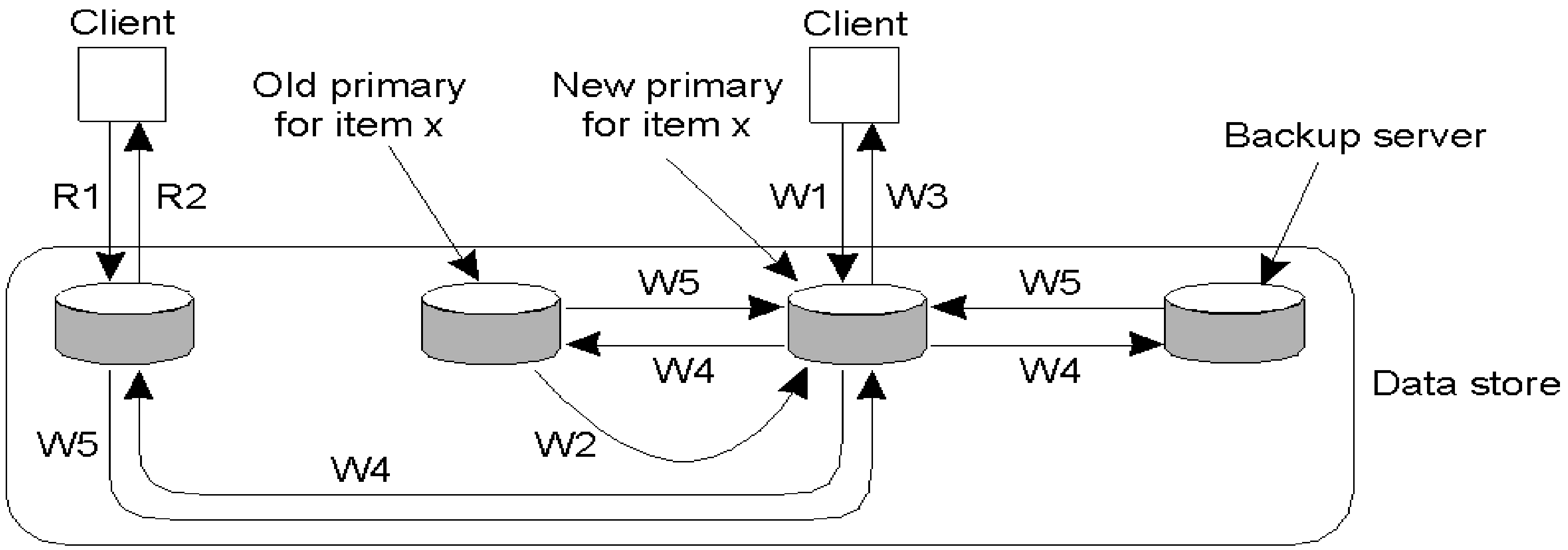
The principle of a primary-backup protocol

- Good: The benefit of this scheme is, as the *primary* is in control, all writes can be sent to each backup replica *IN THE SAME ORDER*, making it easy to implement *sequential consistency*.

Local-Write Protocols

- In this protocol, a single copy of the data item is still maintained.
- Upon a write, the data item gets transferred to the replica that is writing.
- That is, the status of *primary* for a data item is *transferable*.
- This is also called a “fully migrating approach”.

Local-Write Protocols



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-*backup* protocol in which the primary *migrates* to the process wanting to perform an update, *then updates the backups*. Consequently, reads are much more efficient.

Replicated-Write Protocols

- With these protocols, writes can be carried out at *any* replica.
- Another name might be: “Distributed-Write Protocols”
- There are two types:

1. Active Replication- update are forwarded to multiple replicas.
2. Quorum based protocol – 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.

Cache-coherence protocols

- Caches form a special case of replication, in the sense that they are generally controlled by clients instead of servers.
- First, caching solutions may differ in their **coherence detection strategy**, that is, *when* inconsistencies are actually detected. In static solutions, a compiler is assumed to perform the necessary analysis prior to execution, and to determine which data may actually lead to inconsistencies because they may be cached.
- Another design issue for cache-coherence protocols is the **coherence enforcement strategy**, which determines *how* caches are kept consistent with the copies stored at servers. The simplest solution is to disallow shared data to be cached at all. Instead, shared data are kept only at the servers, which maintain consistency using one of the primary-based or replication-write protocols. Clients are allowed to cache only private data. Obviously, this solution can offer only limited performance improvements.

Caching and replication in the Web

- The Web is arguably the largest distributed system ever built. Originating from a relatively simple client-server architecture, it is now a sophisticated system consisting of many techniques to ensure performance and availability requirements. These requirements have led to numerous proposals for caching and replicating Web content.
- Client side caching in the web generally occurs at two place. In the first place most browsers are equipped with a relatively simple caching facility.
- In the 2nd place ,A Web proxy accepts requests from local clients and passes these to Web servers. When a response comes in, the result is passed to the client. The advantage of this approach is that the proxy can cache the result and return that result to another client, if necessary.
- In addition to caching at browsers and proxies, ISPs generally also place caches in their networks. Such schemes are mainly used to reduce network traffic (which is good for the ISP) and to improve performance (which is good for end users)