Consistency and Replication (part 2)

ECE 454 / 751: Distributed Computing

Instructor: Dr. Wojciech Golab

wgolab@uwaterloo.ca

Slides are derived from A. S. Tanenbaum and M. Van Steen,
Distributed Systems: Principles and Paradigms, 2nd Edition, Pearson-Prentice Hall, 2006.

Learning objectives

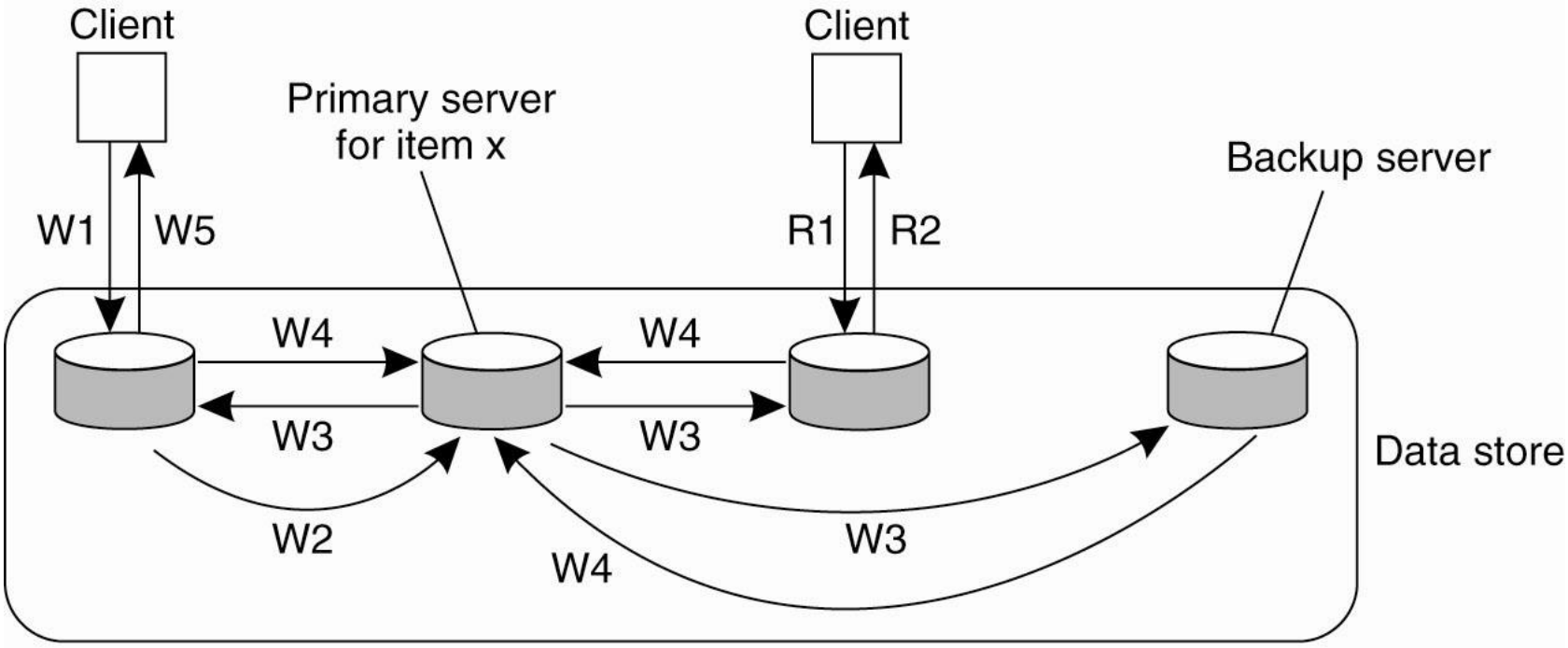
To develop a working knowledge of:

- primary-based replication protocols
- quorum-based replication protocols
- eventually-consistent replication protocols

Primary-based protocols

- Many commercial databases use **primary-based replication** protocols, whereby all updates are executed by a designated primary replica and then pushed to one or more backup replicas.
- Protocols in this family can be classified as **remote-write**, meaning that the primary replica is generally stationary and therefore must be updated remotely by other servers, or **local-write**, meaning that the primary replica migrates from server to server, allowing local updates.
- If the primary replica fails then one of the backup replicas may take over as the new primary. Accurate failure detection is necessary to prevent "split brain" situations.

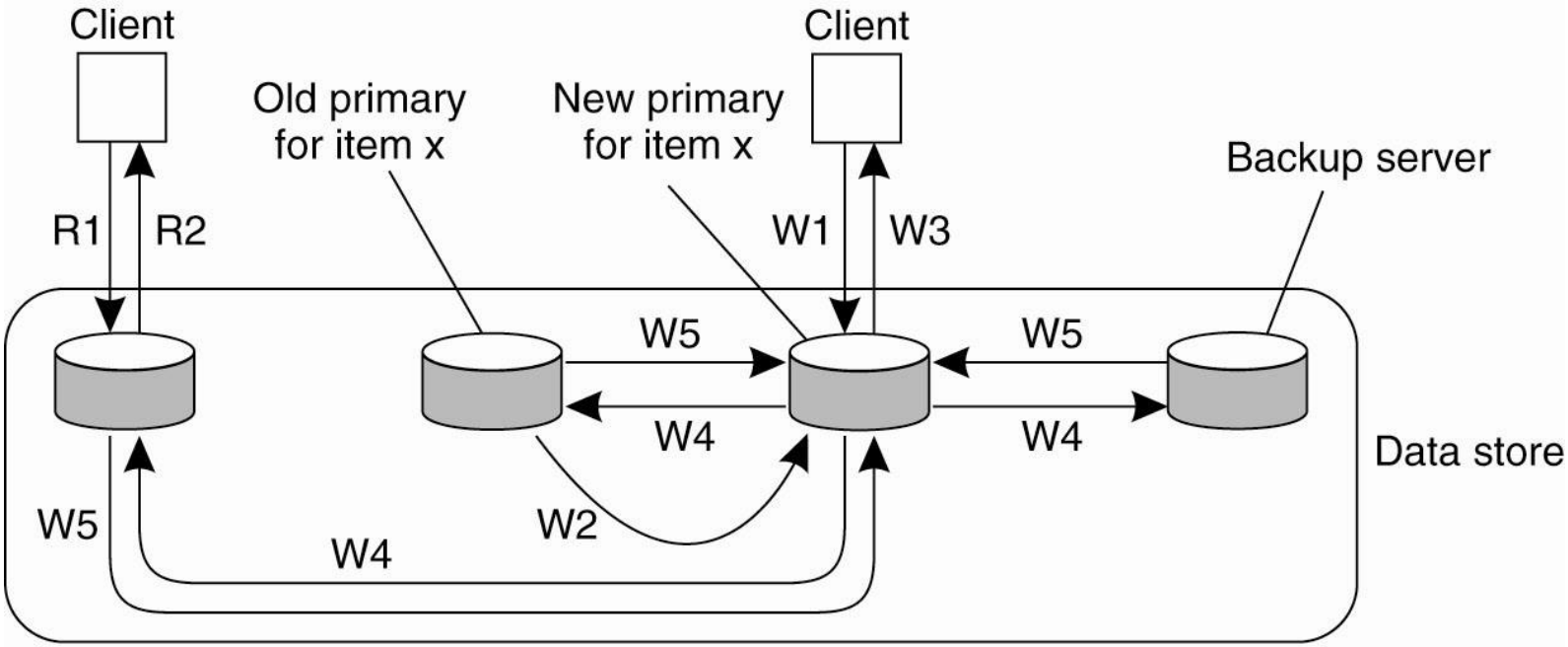
Remote write protocol example



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

Local write protocol example



Quorum-based protocols

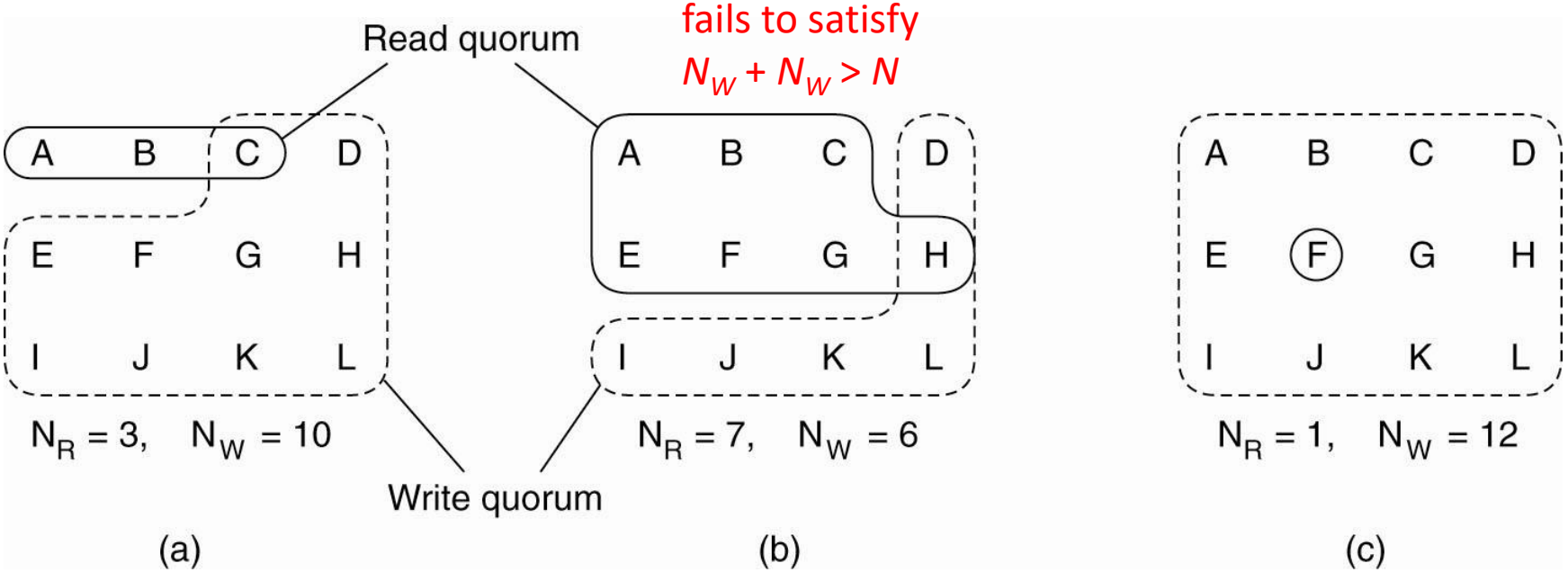
- Forcing all updates through a primary replica makes it possible to implement strong consistency models, such as sequential consistency and linearizability, but can lead to performance bottlenecks and temporary loss of availability when the primary fails.
- Quorum-based protocols instead allow all replicas to receive updates, but each update must be accepted by a sufficiently large subset of replicas called a **write quorum**. Similarly, reading a data object requires accessing a large enough subset of replicas called a **read quorum**.
- Majority quorums are often used in quorum-based protocols, but other configurations are possible as well.

Quorum-based protocols

- Let N, N_R, and N_W denote the total number of replicas for a data object x, the size of the read quorum, and the size of the write quorum.
- In distributed databases, read and write quorums must satisfy two rules of overlap:
  1. $N_R + N_W > N$ (read and write quorums overlap)
  2. $N_W + N_W > N$ (two write quorums overlap)
- Rule 1 enables detection of read-write conflicts, whereas rule 2 enables detection of write-write conflicts.
- In practice N is usually chosen to be an odd integer.
- Example: $N = 5$, $N_R = 2$, $N_W = 4$.

Examples of quorums

The examples below use $N = 12$. Part (c) illustrates a **read-one write-all (ROWA)** scheme.



Partial quorums

- Systems such as Amazon's Dynamo and its various descendants (Cassandra, Voldemort, Riak) can be configured to provide various degrees of consistency by tuning N_R and N_W:

  $N_R + N_W > N$ (strong consistency)

  $N_R + N_W \leq N$ (weak consistency)

- In weak consistency mode, these systems do not avoid read-write or write-write conflicts. Even in strong consistency mode, they do not avoid write-write conflicts.
- Thus, the subsets of replicas accessed by reads and writes do not in general satisfy the rules of overlap for (strict) quorums, and are therefore referred to as **partial quorums**.
- To resolve write-write conflicts, updates are tagged with timestamps, and a resolution policy such as **last write wins** is applied. Use of vector clocks may lead to concurrent timestamps.

Full versus partial replication

- In most of our examples, *N* denotes both the number of replicas for one data object and the total number of servers. This scheme is known as **full replication** (i.e., every server has a copy of the entire data set).
- In practice, we often want the effective storage capacity of the system to grow with the number of servers while keeping the replication factor constant. This implies that each server stores only a subset of data objects when the number of servers exceeds the replication factor. This scheme is known as **partial replication**.

Eventually-consistent replication

Eventual consistency: In the simplest case, reads and updates are resolved using the closest replica. A server that receives an update replies with an acknowledgement to the client first, and then propagates the update lazily to the remaining replicas.

If a replica is unreachable then it can be updated later on using an **anti-entropy** mechanism. For example, replicas may periodically exchange hashes of data to detect discrepancies. Updates can be timestamped to enable determination of the latest version of a data item.

Data staleness

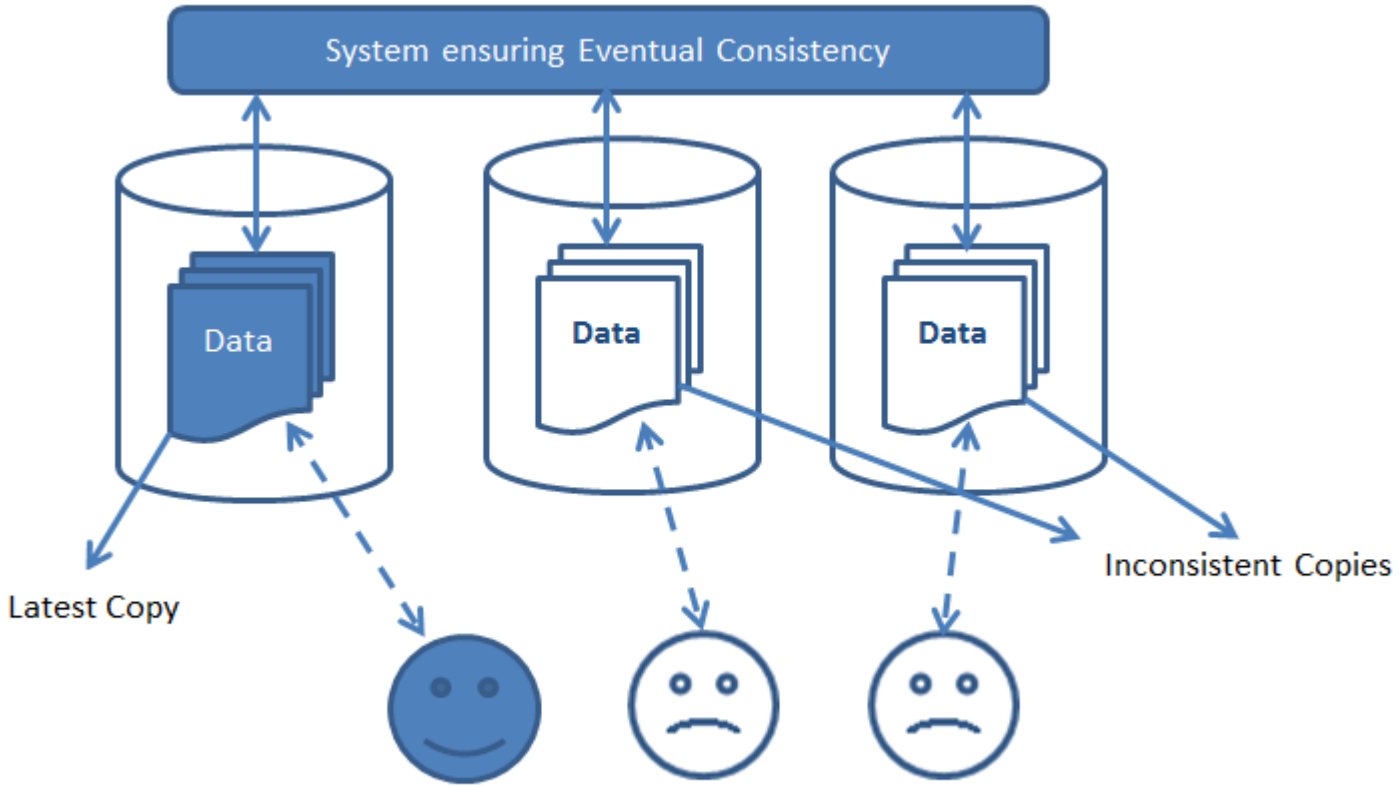


Figure showing a stale state where 2 copies of data are inconsistent with the latest one.

Image source: Shankar Ramakrishnan
http://cloudshankar.blogspot.ca/2013/05/eventual-consistency.html

Anti-entropy

Merkle trees or **hash trees** are exchanged between servers periodically to detect discrepancies between two replicas.

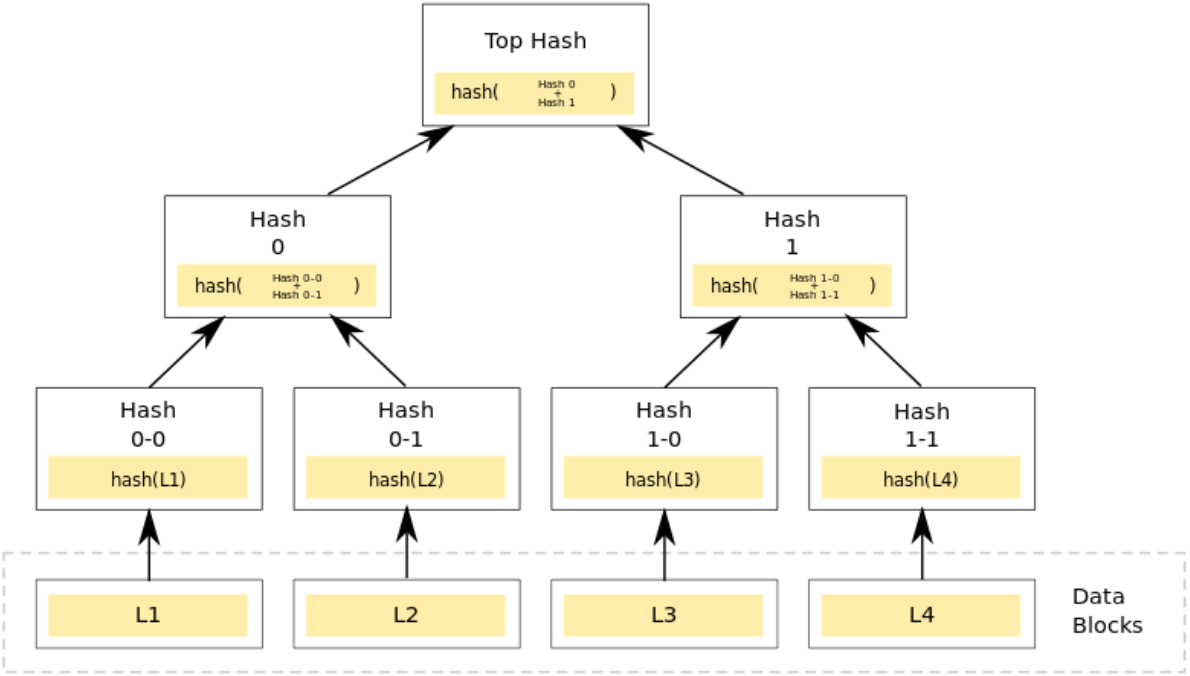


Image source: https://en.wikipedia.org/wiki/Merkle_tree