Replication in Cloud Storage

K V SUBRAMANIAM

CHAPTER 5 : REPLICATION – MARTIN KLEPPMAN – DESIGNING DATA INTENSIVE APPLICATIONS

What and why replication

- Replication copies data across multiple networked machines.
- It improves latency, availability, and read performance by replicating data and metadata.
- Common methods include single-leader, multi-leader, and leaderless approaches.
- WHY - Replication
  - keeps data near users,
  - ensures system reliability during failures,
  - allows scaling to handle more read requests.

Leader based replication

- Leader-based replication, also known as active/passive or master/slave, involves multiple database copies called replicas.

- One replica acts as the leader (master)
  - handling write requests from clients and updating its local storage.
- Other replicas, called followers, receive updates from the leader
  - Via replication log and apply changes in the same order to maintain data consistency.

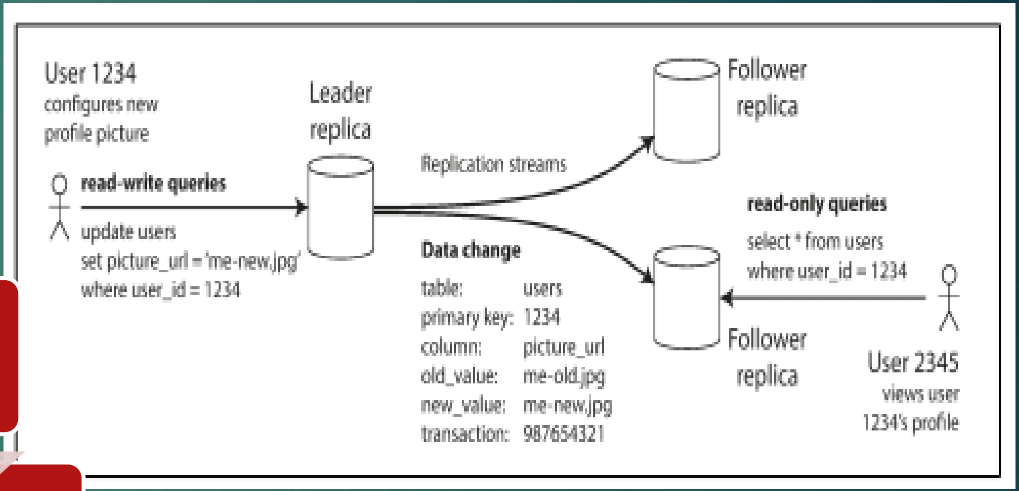
Leader based replication

Clients can read from the leader or followers, but writes go only to the leader.

This replication method is common in relational databases like PostgreSQL, MySQL, Oracle Data Guard, and SQL Server's AlwaysOn.

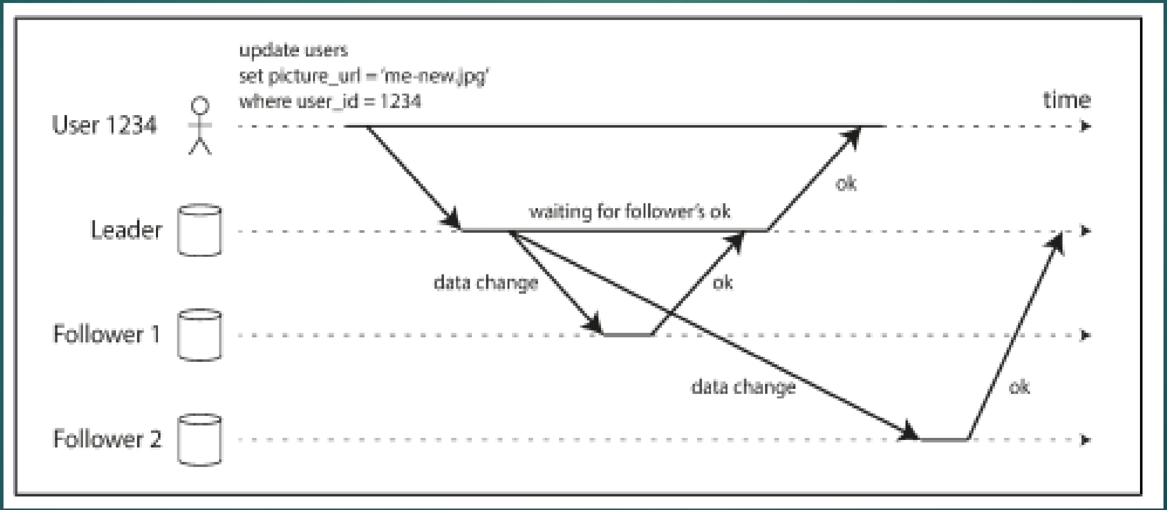
It's also used in non-relational databases such as MongoDB, RethinkDB, and Espresso.

Leader-based replication is applied in distributed message brokers like Kafka and RabbitMQ.



Leader based replication

- **synchronous replication :**
  - the leader waits for followers to confirm the write before confirming success to the user.
- **asynchronous replication:**
  - the leader sends the write without waiting for follower confirmation.



Leader based replication with one synchronous and one asynchronous follower

Follower failure – catch up recover

- Each follower stores a log of data changes from the leader on its local disk.
- **Failure**:
  - When : If a follower crashes or loses connection,
  - Recovery: using this log.
- The follower identifies the last processed transaction before the fault and requests missing updates from the leader.
- After applying these changes, the follower catches up and resumes receiving data normally.

Leader failure

- A follower is promoted to leader.
- Clients redirect writes to the new leader, and followers sync data from it.
- Failover can be manual or automatic.
- Automatic failover steps:
    - detect leader failure, select new leader (already looked at leader election), and reconfigure the system.

Leader based replication - techniques

- **Statement-based replication**
  - logs each write request by the leader and sends it to followers.
- **Write-ahead log (WAL) shipping**
  - records all writes in an append-only log, which followers use to replicate data.
- **Logical log replication**
  - uses a row-level log format to describe changes for replication separately from storage.
- **Trigger-based replication**
  - uses triggers to log data changes, allowing external processes to replicate updates.

Leader based replication – Replication Lag

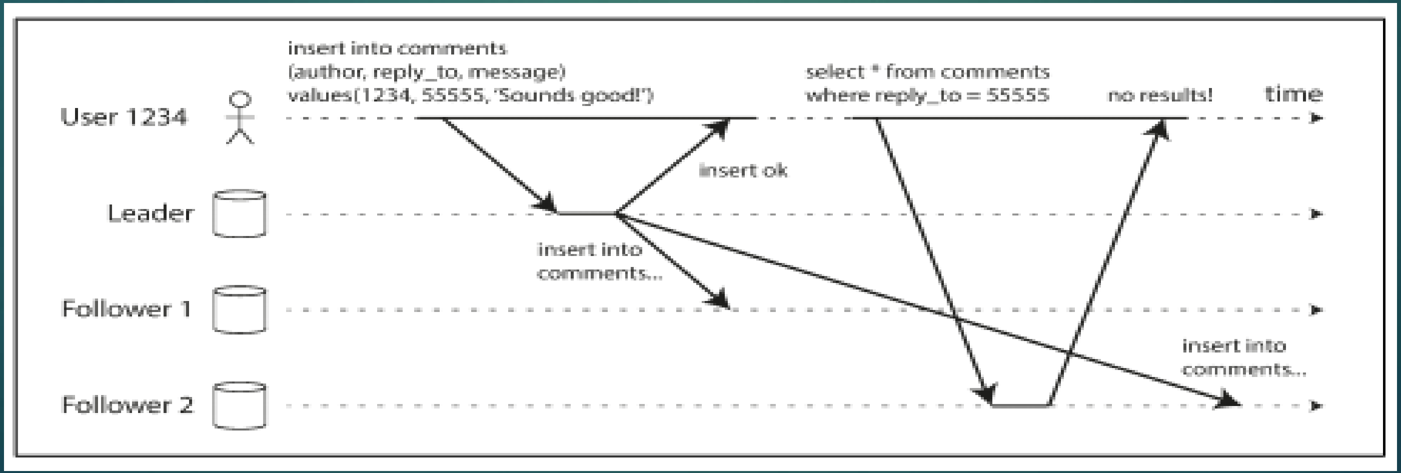
- Reading from an asynchronous follower
  - can show outdated data if the follower lags behind.
- Queries on the leader and follower
  - may return different results temporarily due to replication lag..
- Replication lag is usually brief
  - but can cause real issues if it becomes significant.

Relaxed consistency

- Consistency can be relaxed
  - **Weak consistency**: system does not guarantee to return consistent results
  - **Eventual consistency**: if no further updates, system will become consistent. If updates are infrequent, can wait for some time to get consistent value
  - **Read your writes consistency**: a client performing a read after a write will always see its own updates
  - **Session consistency**: consistency within a session
- Amazon S3
  - US Standard Region: Eventual consistency
  - US West, EU, Asia Pacific Regions: Read your writes consistency for new object creation, eventual consistency for writes and deletes
- Reference: Eventual Consistency by Werner Vogel, Communications of the ACM, January 2009

Reading your own writes consistency

- Read-after-write consistency (read-your-writes consistency)
  - ensures that users always see their own updates when they read data again.
- It does not guarantee
  - updates from other users will be immediately visible.



Leader based replication – Replication Lag - solutions

- **Simple Rule**:
    - Always read critical data from the leader and other data from followers, though this limits read scaling.
- Track replication lag and avoid querying followers with significant delay.
- Clients can store their latest write timestamp to ensure replicas reflect recent updates.
- Use monotonic reads by having users read from the same replica.
- Ensure consistent prefix reads so writes appear in order to all readers.

Other replication architectures

- **Multi Leader Replication**
    - Use cases
        - Multi Data Center operation
        - Clients with offline operation – calendar apps on mobile phone
        - Collaborative Editing – Google Docs

- Leaderless Replication
    - No master node. Any node can potentially be master – Cassandra, Dynamo DB