CLOUD COMPUTING

Replication
Leader Based Replication and
Replication Lag

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

Replication

- We looked at partitioning as way of distributed data across different nodes in a cluster, such that query or IO operations can be done across these multiple nodes supporting the performance and throughput expectations of applications.
- **Replication** is a means keeping a copy of the same data on multiple machines that are connected via a network.
- The data and the metadata are replicated for reasons like:
  1. To keep data geographically close to the users (and thus reduce latency)
  2. To allow the system to continue working even if some of its parts have failed (and thus increase availability)
  3. To scale out the number of machines that can serve read queries (and thus increase read throughput)
- This is very beneficial from a performance perspective for read-only data.

CLOUD COMPUTING

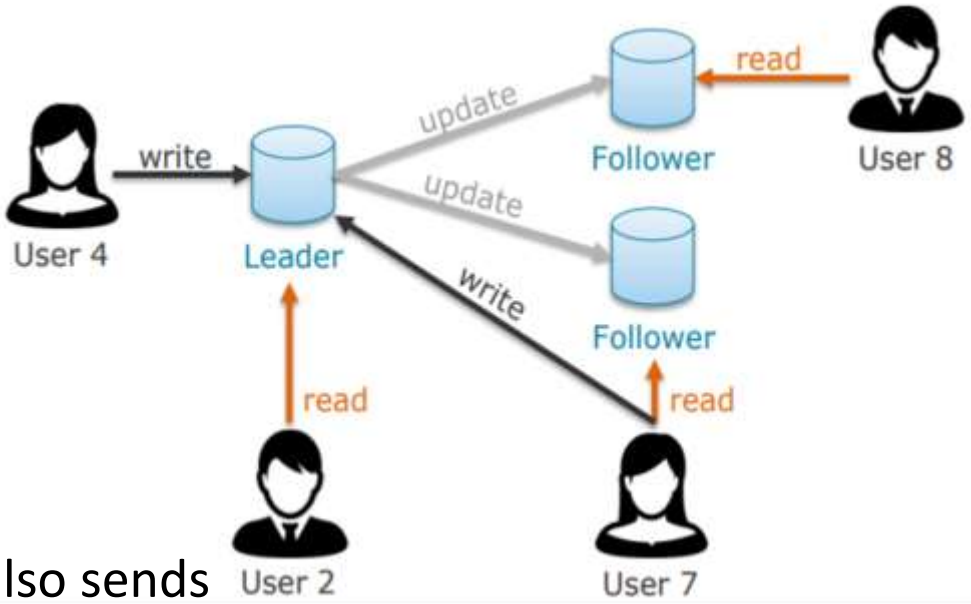
Replication - 2

- Each node that stores a copy of the dataset is called a ***replica***.
- Every write needs to be processed by every replica; otherwise, the nodes will not hold the same data.
- The challenge is how to handle data that changes in a replicated system:
  - Should there be a leader replica and if yes, how many?
  - Should one use a synchronous or asynchronous propagation of the updates among the replicas?
  - How to handle a failed replica if it is the follower? What if the leader failed? How does a resurrection work
- Three popular algorithms for replicating changes between nodes:
  - **Leader based or single-leader based replication**
    - **Synchronous Replication**
    - **Asynchronous Replication**
  - **Multi-leader**
  - **Leaderless** replication

CLOUD COMPUTING

Replication : Leader Based Replication

- Leader based replication is also known as Leader-Follower or master-slave replication
- How do we ensure that all the data is consistent across multiple replicas?
  - One of the replicas is designated the **Leader**.
  - When Users/clients want to write data, they must send their requests to the **leader**, which first writes the new data to its local storage.
  - The other replicas are known as **followers** (read replicas)
  - Whenever the leader writes new data to its local storage, it also sends the data change to all of its followers as part of a replication log
  - Each follower takes the log from the leader and updates its local copy of the data- base accordingly, by applying all writes in the same order as they were processed on the leader.
  - The client can read from anywhere (leader or the followers) but writes are accepted only by the leader
  - Leader-based replication is used in some DBs & distributed message brokers like Kafka and RabbitMQ.



Replication : Single Leader Replication (Rephrased)

Leader

- Dedicated compute node (usually also a replica) responsible for propagating changes
- Also known as master or primary
- Accepts read and write queries
- Sends changes as replication logs to followers

Follower

- General replica
- Also known as slave, secondary, or hot standby
- Accepts only read queries and responds with data from local storage/copy
- Receives changes from leader(s) and updates local copy accordingly:
- Apply all writes in the same order as applied on the leader

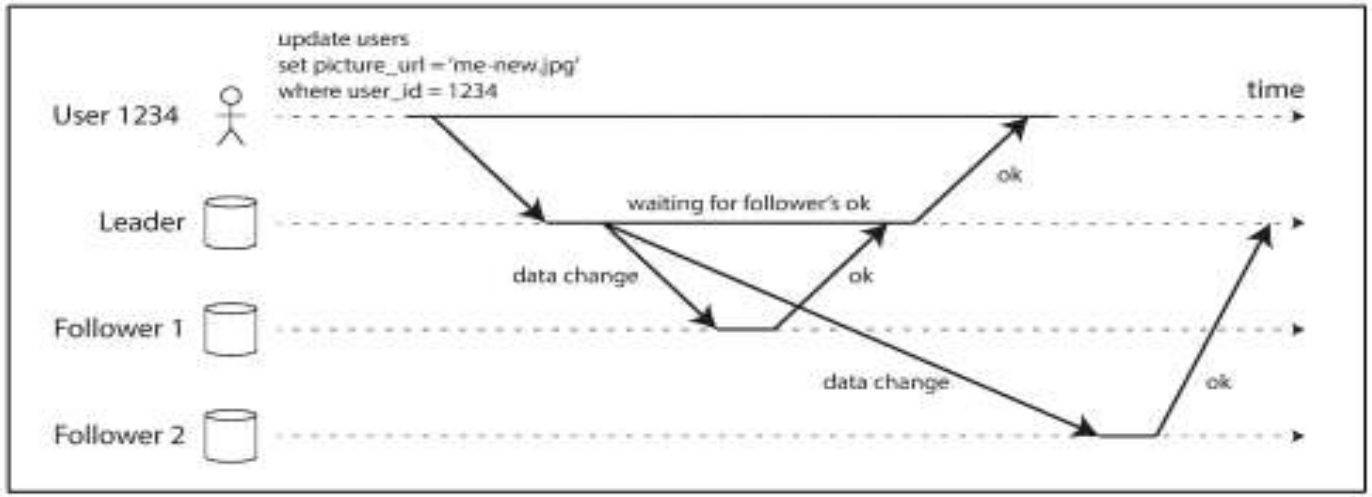
Replication : Leader Based Replication : Synchronous Versus Asynchronous Replication

- In **synchronous** replication, the leader waits until followers have confirmed that it received the write before reporting success to the user and before making the write visible to other clients.

  E.g. the replication to follower 1 is synchronous

- In **asynchronous** replication, the leader sends the message to its follower (s) but doesn't wait for a response from the followers before answering success to the User

  E.g. The replication to follower 2 is asynchronous



Leader based replication with one synchronous & one asynchronous follower

CLOUD COMPUTING

Replication : Leader Based Replication : Implementation of Replication Logs

1. **Statement based replication -** The leader logs every write request that it executes and sends that statement log to its followers.
2. **Write-ahead log (WAL) shipping -** The log is an append-only sequence of bytes containing all writes to the database. The leader writes the log to disk and sends it across the network to its followers. Then the leader updates the data (say the DB) and the followers processes also processes this log and make the changes to the database and thus they builds copy of the exact same data structures as found on the leader.
3. **Change data capture (CDC) based replication - Logical log replication –** Sequence of records that describe the write to database tables at the granularity of rows. Its based on the identification, capture and delivery of the changes made. Replicas can run on different versions or storage engines but use different log formats for different storage engines. Its also easier to parse for external applications. These logs are also called a **logical log**. A logical log for a relational database is usually a sequence of records describing writes to database tables at the granularity of a row.
4. **Trigger based replication (application layer) -** A trigger lets users register custom application code that is automatically executed when a data change (write transaction) occurs in a database system. The trigger has the opportunity to log this change which can be read by an external process. The external process can then apply any necessary application logic and replicate the data change to another system.

Follower Failure: Catch-up recovery

- On its local disk, each follower keeps a log of the data changes it has received from the leader.
- If a follower crashes and is restarted, or if the network between the leader and the follower is temporarily interrupted, the follower can recover quite easily from its log
- Follower knows the last transaction that was processed before the fault occurred. Thus, the follower can connect to the leader and request all the data changes that occurred during the time when the follower was disconnected.
- When the follower has applied these changes, it has caught up to the leader and can continue receiving a stream of data changes as before

Leader Failure - Failover

- One of the followers needs to be promoted to be the new leader
- Clients need to be reconfigured to send their writes to the new leader and the other followers need to start consuming data changes from the new leader.
- Failover can happen manually (an administrator is notified that the leader has failed and takes the necessary steps to make a new leader) or automatically
- Steps followed in an automatic failover process:
  1. Determining that the leader has failed.
  2. Choosing a new leader
  3. Reconfiguring the system to use the new leader

Replication Lag

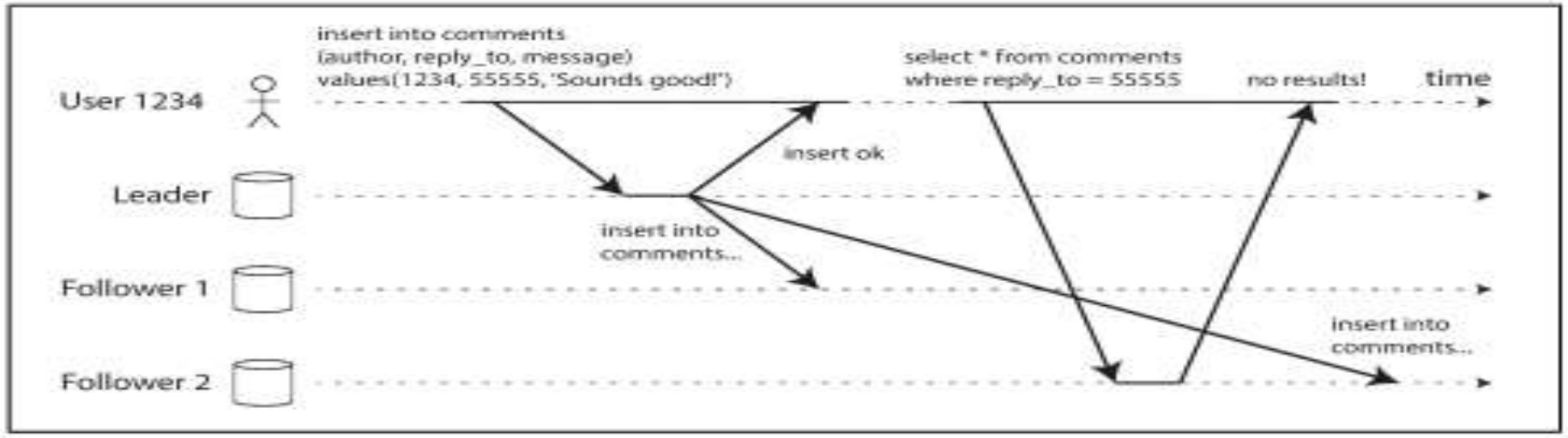
- In Leader-based replication all writes go to the leader, but read-only queries can go to any replica.
- This makes it attractive also for scalability and latency, in addition to fault-tolerance.
- For read-mainly workloads: have many followers and distribute the reads across those followers.
  - Removes load from the leader, allows read requests to be served by nearby replicas.
  - But, only realistic for asynchronous replication otherwise the system will not be available
- If an application reads from an asynchronous follower, it may see outdated information if the follower has fallen behind.
- This leads to apparent inconsistencies in the database: if you run the same query on the leader and a follower at the same time, you may get different results, because not all writes would have been reflected in the follower.

Replication Lag (Continued)

- This inconsistency is just temporary—if there are no writes to the database in a while, the followers will eventually catch up and become consistent with the leader. This effect is known as **eventual consistency**
- In normal operation, the delay between a write happening on the leader and the same being reflected on a follower is known as the ***replication lag*****.** This may be only a fraction of a second and not noticeable in practice
- When the lag is large, the inconsistencies it introduces are not just a theoretical issue but a real problem for applications.

Identification of Inconsistencies due to Replication Lag

Reading Your Own Writes



- Reading your Own writes will help identify that if the user re-reads the data, they will always see any updates they submitted themselves.

There are different models for Consistency like the one above, Read-after-write consistency, different eventual consistency models which will bring in consistency.

Some possible solutions for Replication Lag:

- A simple rule: always read critical data from the leader and rest from a follower (negates the benefit of read scaling)
- Monitor the replication lag on followers and prevent queries on any follower with significant lag behind the leader.
- The client can remember the timestamp of its most recent write—then the system can ensure that the replica serving any reads for that user reflects updates at least until that timestamp
- Monotonic reads - make sure that each user always makes their reads from the same replica
- Consistent prefix reads - if a sequence of writes happen in a certain order, then anyone reading those writes should see them appear in the same order

Multi-Leader Replication

Recap: Replication *is a means keeping a copy of the same data on multiple machines that are connected via a network. We discussed that there were three popular algorithms for replicating changes between nodes: single-leader, multi-leader, and leaderless replication.*

- *We discussed* ***Leader Based Replication*** *earlier.*
- Leader-based replication has a single bottleneck in the leader
- All writes must go through it. If there is a network interruption between the user and the leader, then no writes are allowed.
- An alternate approach to consider is , what if we have more than one leader through whom you can do the writes.

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu