CLOUD COMPUTING

Leaderless Replication

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 (Recap)

- We looked at **Replication** as a means of 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)
- We discussed that every write needs to be processed by every replica; otherwise, the nodes will not hold the same data.
- We discussed on the two of the 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 (Recap – 2)

- As part of the Leader based replication, we discussed one of the replicas would be designated as a leader and typically all Users/clients wanting to write data to storage, would send their requests to the leader, which first writes the new data to its local storage. The leader then sends the data change to all of the other replicas known as followers using some kind of replication log. These followers are also call read replicas.
- We also discussed that potential issues like the leader failure, follower failure or the **replication lag** which is the lag between the write happening on the leader and getting replicated on the follower or that eventually up-dation would happen leading to eventual concurrency across the nodes. We also discussed on simple techniques which were used to address the same.
- We also saw that a single leader was a bottleneck for the environment and looked at having multiple leaders. We saw the advantages it would provide in scenarios like data centers, we saw the challenges it would bring in like synchronization and looked at a few simple approaches which could be used to address the same.
- We also contrasted the single leader and multi-leader replications in terms of performance, tolerance to data center failures, network failures

CLOUD COMPUTING

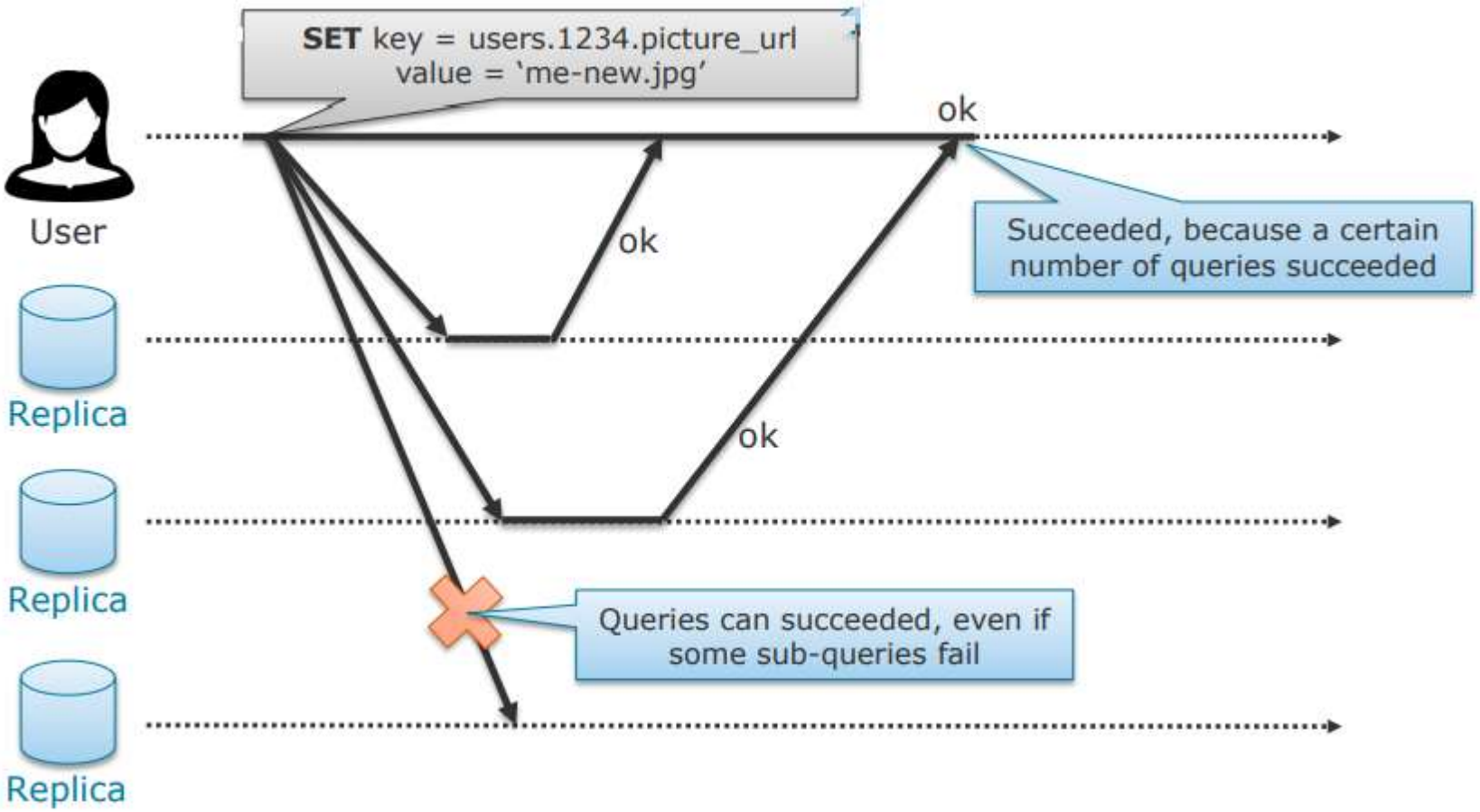
Leaderless Replication

- In contrast to the Single Leader and Multileader replication strategies, Leaderless replication adopts a philosophy that Nodes in the leaderless setting are considered peers
- All the nodes accept writes & reads from the client.
- Without the bottleneck of a leader that handles all write requests, leaderless replication offers better availability.
- A leaderless replication is an architecture where every write must be sent to every replica.
- A write is considered to be successful when the write is acknowledged by (a quorum of) at least k out of n replicas and the read is considered successful in reading a particular value, when (a quorum of) at least k out of n reads must agree on a value
- It has the advantage of parallel writes

Leaderless Replication

Writes with Leaderless Replication

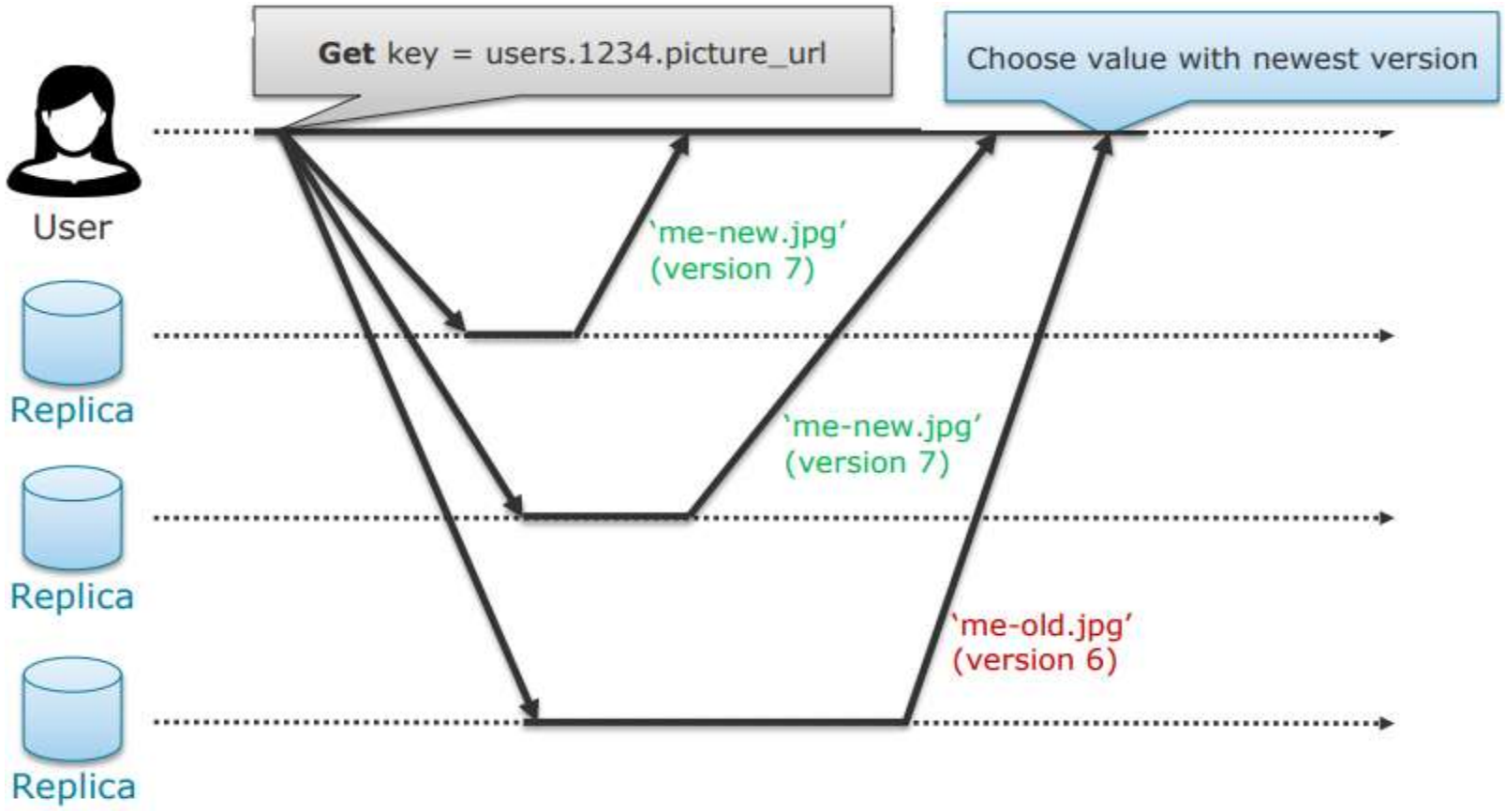
- Upon a **write**, the client broadcasts the request to all replicas instead of a special node (the leader) and waits for a certain number of ACKs.
- The write is completed and considered successful when the write must be acknowledged by at least k out of n replicas. (E.g. in the figure when two of the three replicas send ACK to the client)



Leaderless Replication

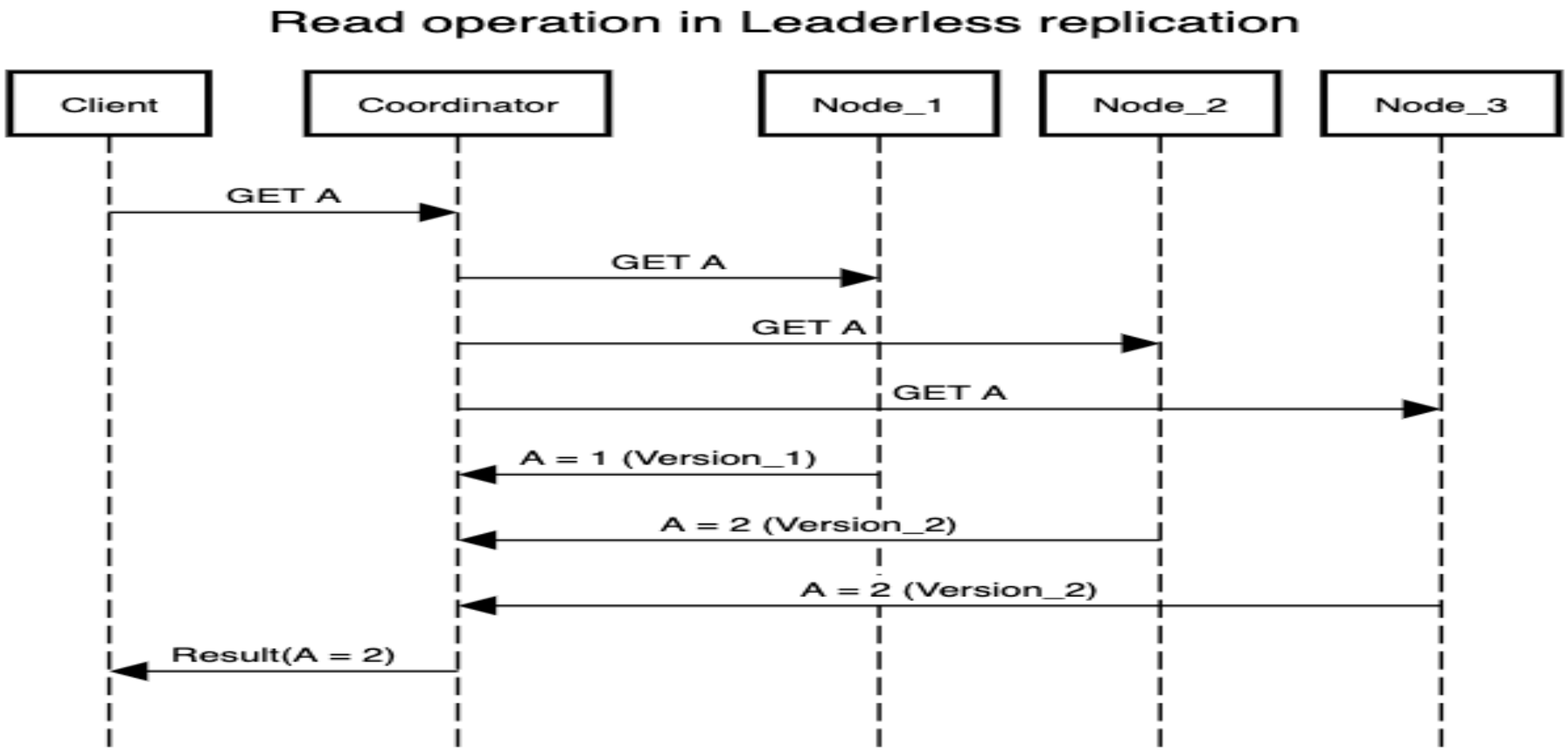
Reads with Leaderless Replication

- Upon read, the client contacts all replicas & waits for some number of responses.
- In order to be considered successful in reading a particular value, at least k out of n reads must agree on a value. E.g read request is completed when two out of the three replicas returns the latest version.
- This approach of the client waiting for many (quorum of) responses, the approach is known as quorum or the value k is known as quorum.
- k value when you are reading is known as read quorum and k value when you are writing is known as write quorum
- This quorum can be configured in a way to provide consistency to the copies.



Leaderless Replication

Reads with Leaderless Replication (another example)



CLOUD COMPUTING

Leaderless Replication : Potential Pitfalls - 1

Writing to the Database When a Node Is Down

- Consider three replicas with one of the replicas being currently unavailable.
- The client sends the write to all three replicas in parallel, and the two available replicas accept the write but the unavailable replica misses it.
- We consider the write to be successful since two out of three replicas have acknowledged the write
- The client simply ignores the fact that one of the replicas missed the write
- Suppose the unavailable node comes back online and clients start reading from it. Any writes that happened while the node was down will be missing from that node. Thus, stale (outdated) values may be read from that node as response.

An approach to address the same

- To solve this problem, read requests are also sent to several nodes in parallel.
- The client may get different responses from different nodes; i.e., the up-to-date value from one node and a stale value from another.
- Version numbers can be used to determine which value is newer

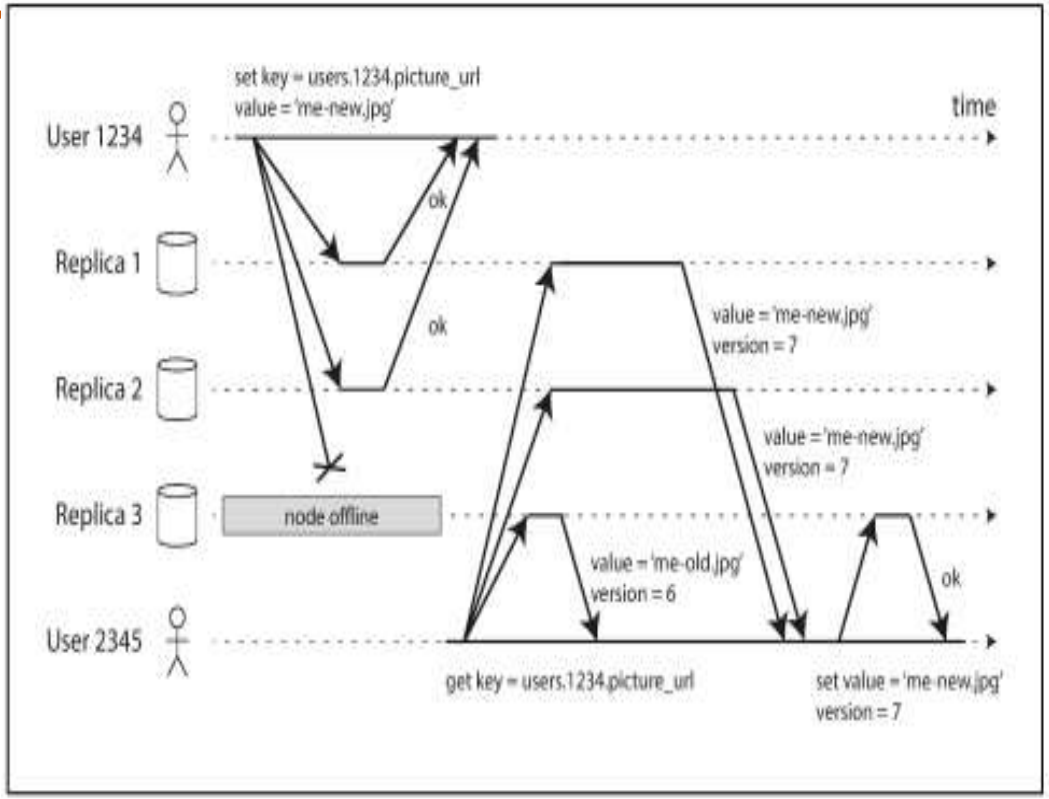
CLOUD COMPUTING

Leaderless Replication – Pitfalls - 2

How does a node catch up on the writes that it missed?

Read repair

- When a client makes a read from several nodes in parallel, it can detect any stale responses. User 2345 gets a version 6 value from replica 3 and a version 7 value from replicas 1 and 2.
- The client sees that replica 3 has a stale value and writes the newer value back to that replica.



Anti-entropy process

- Some datastores have a background process that constantly looks for differences in the data between replicas and copies any missing data from one replica to another
- There are approaches where there are agents run on the replica which periodically match their states and actively propagate the same (Using Gossip kind of approach)

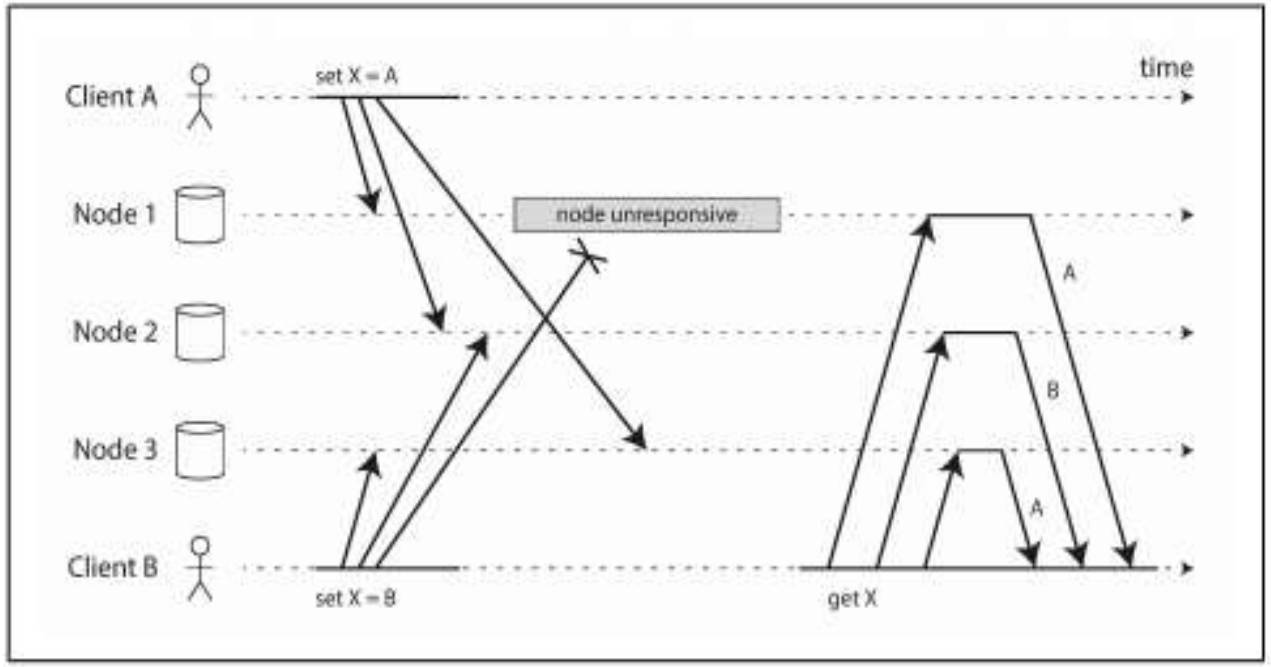
Leaderless Replication : Pitfalls - 3

Detecting Concurrent Writes

- Databases allow several clients to concurrently write to the same key, which means that conflicts will occur even if strict quorums are used
- Events may arrive in a different order at different nodes, due to variable network delays and partial failures
- If each node simply overwrote the value for a key whenever it received a write request from a client, the nodes would become permanently inconsistent
- In order to become eventually consistent, the replicas should converge toward the same value

Approaches to Address this

- Usage of timestamps to resolve these write conflicts can be done. This could lead to additional challenges as clocks may not be synchronized.



Leaderless Replication : Pitfalls - 3

Detecting Concurrent Writes (Cont.)

- **Last write wins** (discarding concurrent writes) - One approach for achieving eventual convergence is to declare that each replica need only store the most “recent” value and allow “older” values to be overwritten and discarded.
- **The “happens-before” relationship and concurrency** - How do we decide whether two operations are concurrent or not?
  - An operation A happens before another operation B if B knows about A, or depends on A, or builds upon A in some way. Whether one operation happens before another operation is the key to defining what concurrency means.
  - Server can determine whether two operations are concurrent by looking at the version numbers
  - In case of multiple replicas, we need to use a version number per replica as well as per key stored in a version vector.

    (cont.)

Leaderless Replication : Pitfalls - 3

Algorithm for determining whether two operations are concurrent by looking at the version numbers

- The server maintains a version number for every key, increments the version number every time that key is written, and stores the new version number along with the value written.
- When a client reads a key, the server returns all values that have not been overwritten, as well as the latest version number. A client must read a key before writing.
- When a client writes a key, it must include the version number from the prior read, and it must merge together all values that it received in the prior read. (The response from a write request can be like a read, returning all current values, which allows to chain several writes.)
- When the server receives a write with a particular version number, it can overwrite all values with that version number or below (since it knows that they have been merged into the new value), but it must keep all values with a higher version number (because those values are concurrent with the incoming write).

Leaderless Replication : Quorum Consistency

Quorum

- Given n nodes, the quorum (w,r) specifies …
  - the number of nodes w that must acknowledge a write and
  - the number of nodes r that must answer a query

Quorum Consistency

- If w + r > n, then each query will contain the newest version of a value
  - Identify the newest value by its version (not by majority!)
- The quorum variables are usually configurable:
  - Smaller r (faster reads) causes larger w (slower writes) and vise versa
- The quorum tolerates:
  - n – w unavailable nodes for writes
  - n – r unavailable nodes for reads

A common choice is to make n an odd number (typically 3 or 5) and to set w = r = (n + 1) / 2

Monitoring staleness

- For leader-based replication, the database typically exposes metrics for the replication lag. By subtracting a follower's current position from the leader's current position, you can measure the amount of replication lag.
- For leaderless replication, there is no fixed order in which writes are applied, which makes monitoring more difficult

Leaderless Replication

Multi-datacenter operation

- Leaderless replication is also suitable for multi-datacenter operation, since it is designed to tolerate conflicting concurrent writes, network interruptions and latency spikes.
- The number of replicas **n** includes nodes in all datacenters, and the number of replicas you want to have in each datacenter can be configured.
- Each write from a client is sent to all replicas, regardless of datacenter, but the client usually only waits for acknowledgment from a quorum of nodes within its local datacenter so that it is unaffected by delays and interruptions on the cross-datacenter link.
- The higher-latency writes to other datacenters are often configured to happen asynchronously

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu