System Design: Database Replication Strategies

Database Replication

The different data replication algorithm, is as follows,

1. Single Leader based replication

*In Single Leader based database replication, also known as **Master-Slave replication**, a single node acts as a leader and rest of the nodes acts as the followers of the leader node. All the writes, done by the client, first happens on the leader node and the followers subsequently gets the data from the leader.*

The followers can get the data from the leader
either **synchronously** or **asynchronously**.

In **Synchronous mode**, the data written on the leader node is synched to the followers and the transaction is not deemed successful, until unless all the followers give a success response of the replication to the leader,

Pros:

1. The leader and the followers will always have consistent state.
2. For a read heavy system, the data can be read from any follower, thus it will improve latency.

Cons:

1. If any one follower fails to acknowledge the replication of data to the leader, the transaction will fail.
2. Since, the write only happens to the leader, in case if the leader fails, the whole system will come to a halt.

whereas,

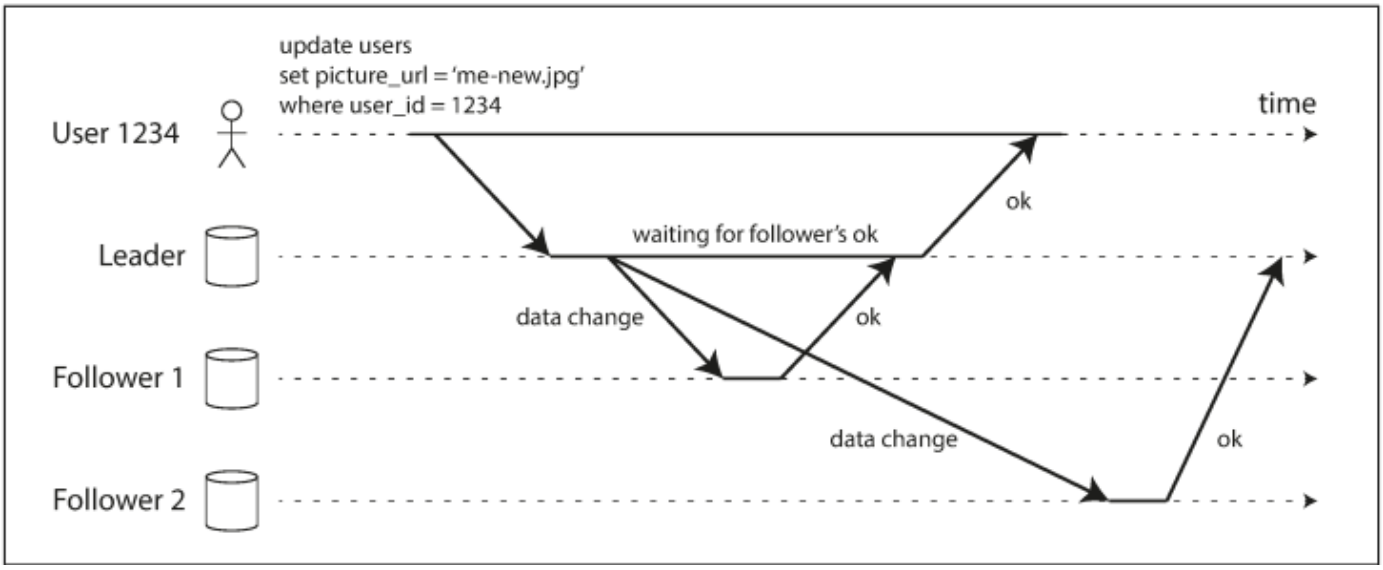
In **Asynchronous mode**, the data is replicated to the followers in asynchronous mode and the transaction is deemed successful, when the leader sends the acknowledgement of the transaction back to the client without taking in account the status of the replication at the followers end.

Pros:

1. The leader continues to accept writes even if all the followers have fallen behind.

Cons:

1. Even though the writes has been confirmed to the client, it is not guaranteed that the write was durable since there is a possibility that many followers might miss out on the write.



Leader based replication with one synchronous and one asynchronous follower

In the above picture, the follower 1 is a synchronous follower and the leader waits for the acknowledgement of the followers before sending OK response back the the user, whereas the follower 2 is an asynchronous follower where the leader doesn't wait for a response and sends an OK response.

After the above discussions, you might be curious to know, if a follower missed out on these writes, how it gets again in synch with the leader.

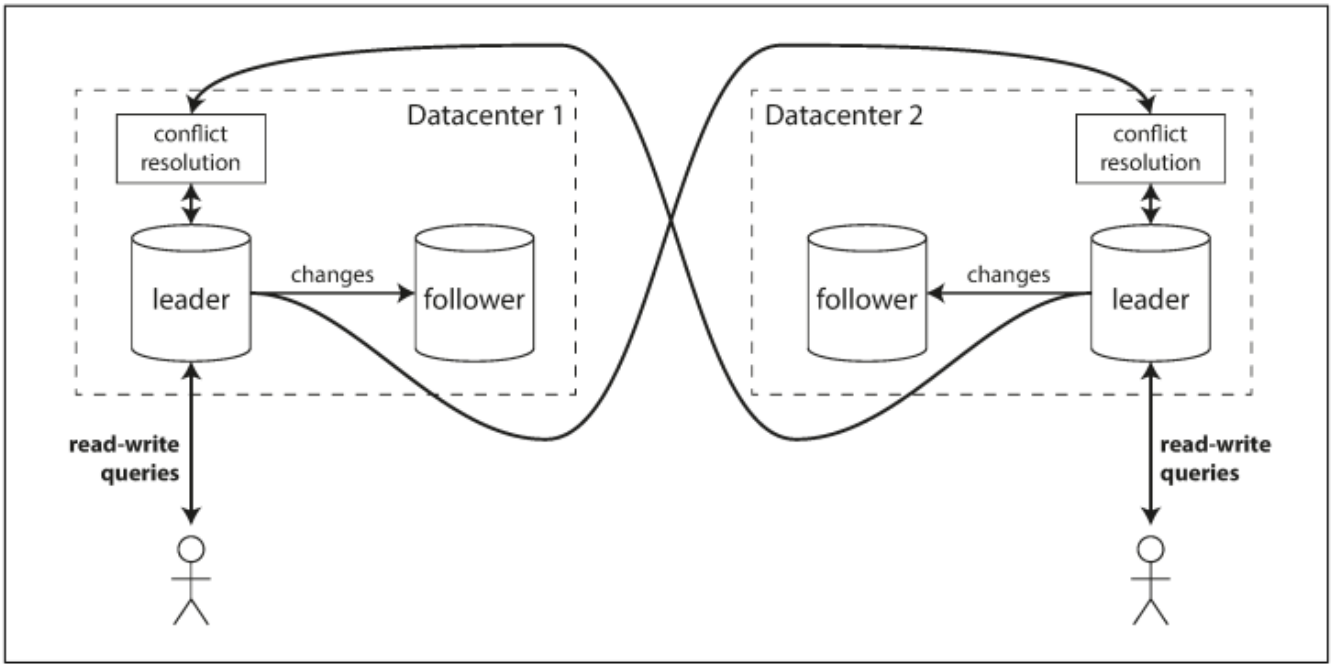
On its local disk, each follower keeps a log of the data changes it has received from the leader. If a follower crashes and is restarted, or if the network between the leader and the follower is temporarily interrupted, the follower can recover quite easily: from its log, it knows the last transaction 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 it has applied these changes, it has caught up to the leader and can continue receiving a stream of data changes as before.

---

2. Multi Leader based replication or master–master or active/active replication

Leader based replication has one major downside, there is only one leader and all writes must go through it. If there is a network interruption between you and the leader, then you can't write to the database.

In multi leader based replication, more than one node is allowed to accept writes and data is then replicated to the follower nodes and each leader simultaneously acts as a follower to the other leaders.



Multi-leader replication across multiple datacenters.

Use cases of multi-leader replication:

Multi-datacenter operation

In multi-datacenter application, we have leader in each datacenter which enables each datacenter to work independently of other datacenter and any outage in one datacenter will not affect the other datacenter and application will not come to halt.

Clients with offline operation

Suppose you have an application that needs to continue to work while it is disconnected from the internet.

Consider there is calendar app on your phone which should be able to accept a new meeting (write request) or see your meeting (read request) regardless of whether you are connected to the internet or not.

Every device has a local database that acts as a leader (it accepts write requests), and there is an asynchronous multi-leader replication process (sync) between the replicas of your calendar on all of your devices. The replication lag may be hours or even days, depending on when you have internet access available.

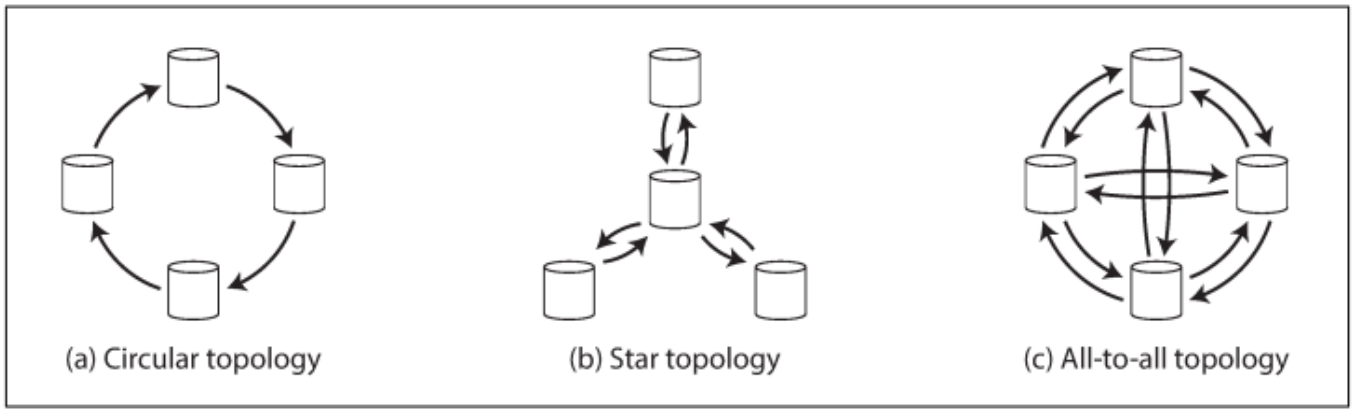
Tolerance of data center outages

When one datacenter fails if doesn't halt the whole application, since each datacenter has its own leader and thus each datacenter works independently of other datacenter and replication catches up when the failed datacenter comes back online.

Collaborative editing

When one user edits a document, the changes are instantly applied to their local replica (the state of the document in their web browser or client application) and asynchronously replicated to the server and any other users who are editing the same document.

Multi-Leader Replication Topologies



Database replication topologies

A replication topology describes the communication paths along which data is transferred from one node to another.

Circular Topology

In circular topology, each node receives writes from one node and forwards those writes (plus any writes of its own) to one other node.

Star Topology

Another popular topology has the shape of a star where, one designated root node forwards writes to all of the other nodes. The star topology can be generalized to a tree.

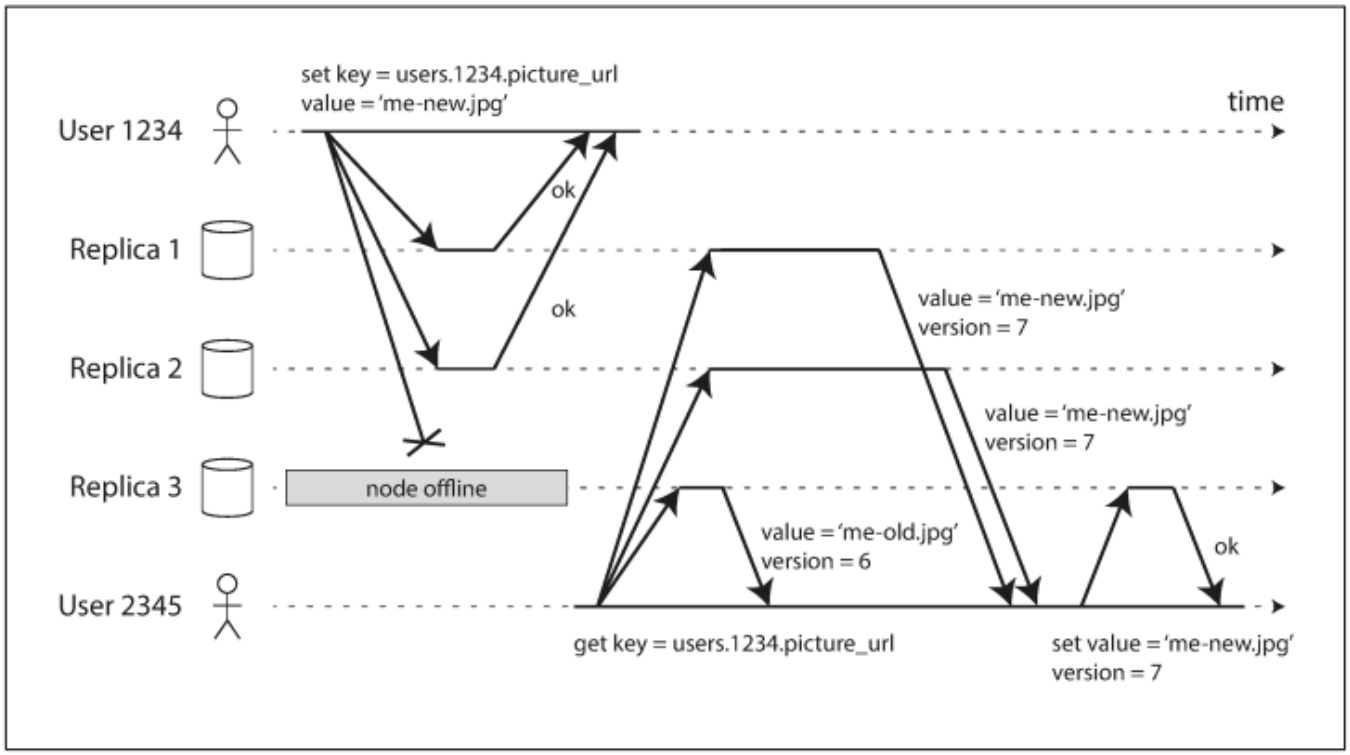
All-to-all Topology

All-to-all topology is generally used and in this topology, every leader sends its writes to every other leader.

3. Leaderless replication

In leaderless replication, unlike single and multileader replication, there is no leader. The client directly sends its writes to several replicas and it majority of the replicates gives an OK response, the write is considered to be successful.

In leaderless configuration, failover does not exist. The below figure shows what happens: the client (user 1234) sends the write to all three replicas in parallel, and the two available replicas accept the write but the unavailable replica misses it. Let's say that it's sufficient for two out of three replicas to acknowledge the write: after user 1234 has received two ok responses, we consider the write to be successful. The client simply ignores the fact that one of the replicas missed the write.



Leaderless replication illustration

Now, what if the node, which missed the write, comes back online. When the reader reads the data from this node, it will always get the stale data.

How to ensure that user gets the updated/recent data and also synch the node with the updated data?

To solve the first problem, when a client reads from the database, it doesn't just send its request to one replica: 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 are used to determine which value is newer.

When a client makes a read from several nodes in parallel, it can detect any stale responses. For example, in the above figure, 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. This approach works well for values that are frequently read.
