arpit.substack.com

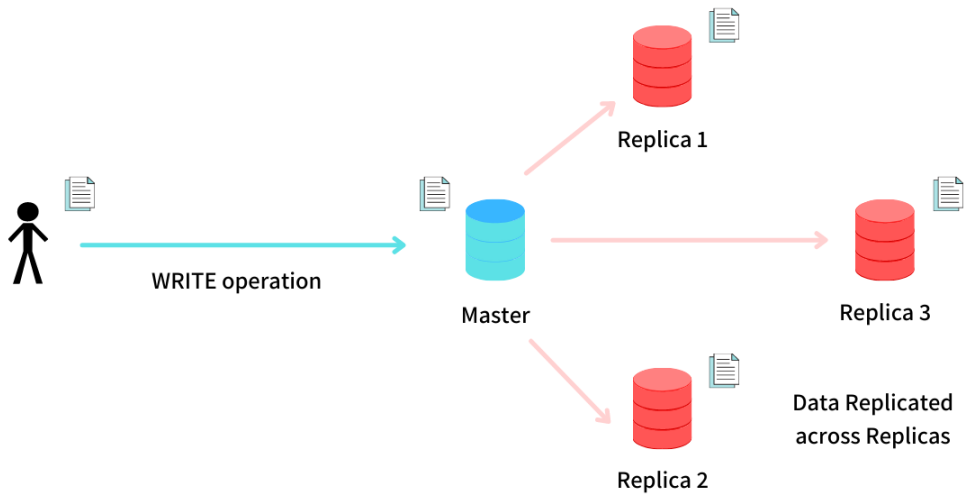
Leaderless Replication

Arpit Bhayani

7-9 minutes

Traditional leader-centric replication strategies revolve around the fact that there will be one Master (leader) node that will acknowledge and accept all the writes operations and then replicate the updates across the replicas (read or master). In this essay, we take a look into a different way of replication, called Leaderless Replication, that comes in handy in a multi-master setup that demands strong consistency.

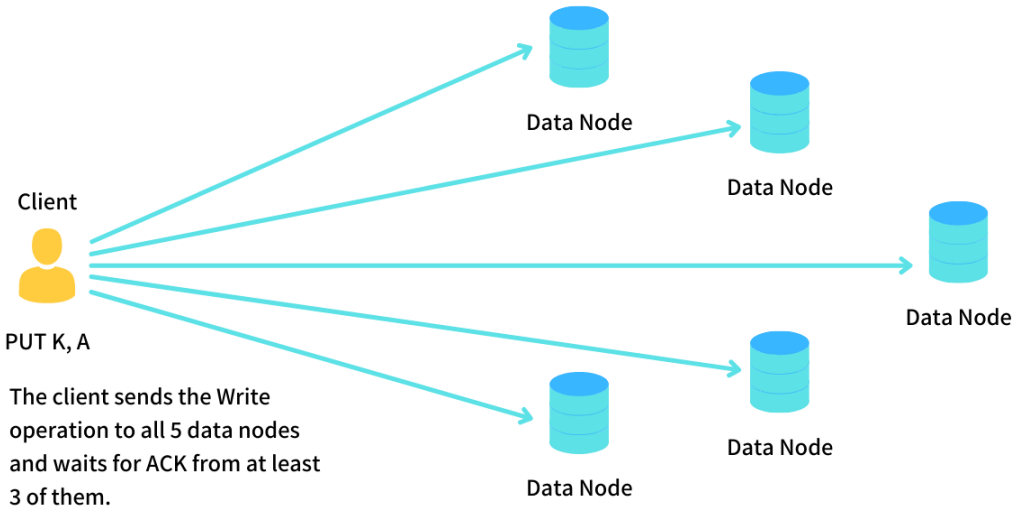
In a leader-centric replication, there is a Master node that accepts the writes. Upon applying the writes on its copy of data, the database engine sends out the updates across read-replicas or master nodes. Given that all the writes flow through the Master node, the order of the writes is deterministic, slimming down the chances of having a write conflict.



Leader-centric replication is not fault-tolerant by design because we lose the write operation when the Master node is down. Leaderless replication addresses this concern and ensures our system can handle Write operations even when a subset of nodes are having an outage.

Leaderless Replication eradicates the need of having a leader accepting the writes; instead, it leverages quorum to ensure strong consistency across multiple nodes and good tolerance to failures. Here's how `WRITE` and `READ` operations happen in such a system.

Say we have a database cluster of 5 nodes (all Masters). In Leaderless Replication, when a client wants to issue a write operation, it fires this operation on all 5 nodes and waits for `ACK` from at least 3 nodes. Once it receives `ACK` from a majority of the nodes, it marks the Write as `OK` and returns; otherwise, it marks the operation as `FAIL`.



Every record in the database has a monotonically increasing version number. Every successful write updates this version number allowing us and the system to identify the latest value of the record upon conflict.

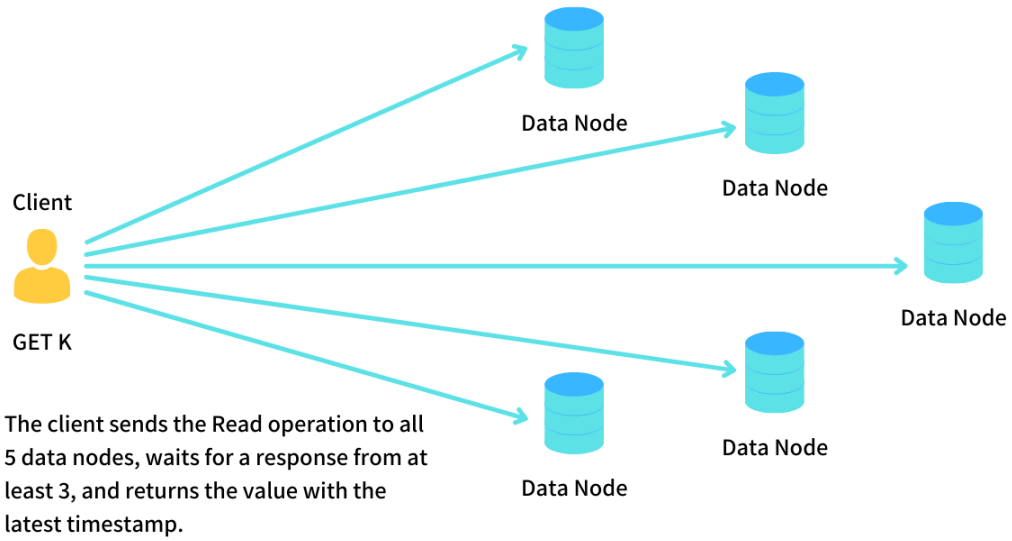
Say, when the write operation was triggered, it reached just 4 nodes because the fifth node was down; so, when this node comes

back up, it gossips with the other 4 nodes and identifies the writes that it missed and then pulls the necessary updates.

Once the write is ACK and confirmed, the nodes gossip internally to propagate the writes to other nodes. There could be a significant delay for the writes to propagate and sync across all nodes; hence we need to ensure that our reading strategy is robust enough to handle this delay while guaranteeing strong consistency.

Given that there could be a significant delay in the updates to propagate across all N nodes, the Read strategy in Leaderless Replication needs to be robust enough.

Like how the client fanned out the write operation to all the nodes, it also fans out the Read operation to all N nodes. The client waits to get responses from at least `N/2 + 1` nodes. Upon receiving the responses from a majority of the nodes, it returns the value having the largest version number.



Given that we mark a write as 0K only when at least `N/2 + 1` of them ACK it and we return our read-only when we get responses from at least `N/2 + 1` nodes, we ensure that there is at least one node that is sending the latest value of the record.

If we send our read operation to just one node chosen at random,

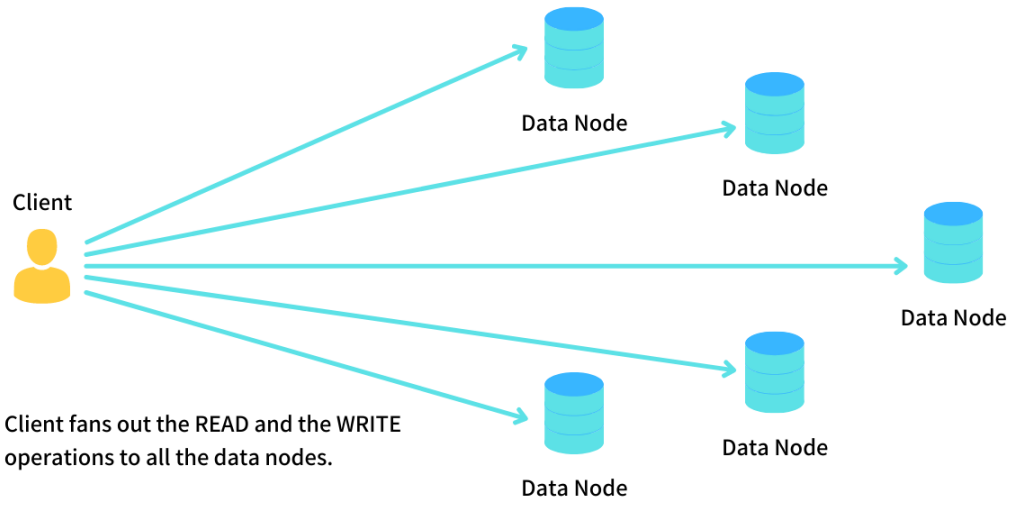
there is a high chance that the value it returns in the response is a stale one, defeating our guarantee of having a strong consistency.

The Leaderless Replication system we just discussed is specific because it restricts clients to send write to all `N` nodes and wait for ACK from at least `N/2 + 1` nodes. This constraint is generalized in real-world with

- `w`: number of nodes that confirm the writes with an ACK
- `r`: number of nodes we query for the read
- `n`: total number of nodes

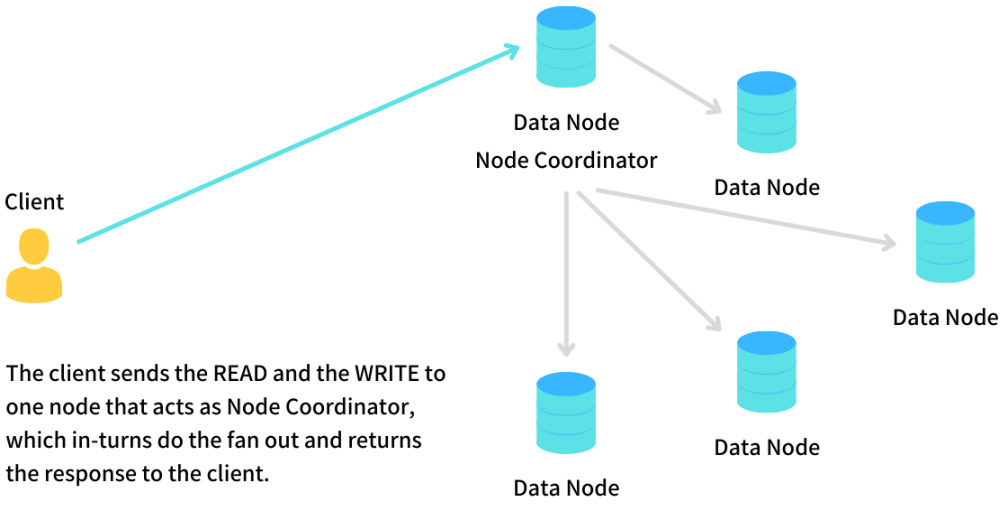
to have a strong consistency i.e. we can expect to get the latest value of any record so long as `w + r > n`, because with this there will be at least one node that has the latest value that will return it. Such reads and writes are called Quorum Reads and Writes.

Now that we understand Leaderless Replication, we look at approaches to implementing it. In the flow discussed above, the approach we saw was the client was sending reads and writes to all the replicated data nodes and, depending on the quorum configuration, decides the correctness. This approach is called *client-driven fan-out* and is very popular.



Another popular approach to implement Leaderless Replication is

to have a *Node Coordinator*. The client will make the request to any one node, and it then starts to act as the coordinator for that transaction. This node coordinator will then take care of the fan-out to other nodes and complete the transaction. Upon completion, it returns the response to the client.



Node coordinator-based replication makes life simpler for clients by offloading all the complications and coordination to the node coordinator. Apache Cassandra uses this approach for implementing Leaderless Replication.

Now that we understand the micro-nuances of Leaderless Replication, let's address the elephant in the room - Why should we even use Leaderless Replication when it is so complex to set up? There are a couple of strong reasons.

Leaderless Replication is strongly consistent by default. In an over-simplified database cluster of N nodes, we are fan-out writes to all N nodes and wait for the ACK from at least `N/2 + 1` nodes; and while reading, we fan-out reads to all N nodes and wait for a response from at least `N/2 + 1` nodes, and then return the value that is most recent among all.

Given that we are playing with the majority here, the set of nodes that handle reads and that handles writes cannot be mutually

exclusive and will always have at least 1 overlapping node having the latest value. The assurance of fetching the latest value when Read, no matter how immediate, is how Leaderless Replication ensures a Strong Consistency by design.

> Note: As discussed in previous sections, it is not mandatory to fan-out reads and writes to majority nodes, instead we need a subset of nodes for reads `r`, and another subset for writes `w`, and ensure `r + w > N` for strong consistency.

Leaderless Replication is fault-tolerant by design with no Single Point of Failure in the setup. Given there is no single node acting as a leader, the system allows us to fan-out writes to multiple nodes and wait for an ACK; and once we get it from a majority of nodes we are assured that our Write is registered and will never be lost.

Similarly, the reads do not go to just one node; instead, the reads are also fanned-out to all the nodes, and upon receiving the response from a majority of the nodes, we are bound to get the most recent value in one of those responses given there will be an overlap of at least 1 node where the latest write went and the value was read from.