Log In Join

Back To Module Home

Distributed Systems

0% completed

Search Module

Conclusion

Mark Module as Completed

Single-Master Replication Algorithm

Learn about single-master replication, and its practical application, advantages, and disadvantages.

We'll cover the following

Single-master replication#

Single-master replication is a technique where we designate a single node amongst the replicas as the **leader**, or primary, that receives all the updates.

> This technique is also known as **primary-backup replication**.

We commonly refer to the remaining replicas as **followers** or secondaries. These can only handle read requests. Every time the leader receives an update, it executes it locally and also propagates the update to the other nodes. This ensures that all the replicas maintain a consistent view of the data.

Four replica nodes, where node B is designated as the leader node and the remaining three nodes are followers

Techniques for propagating updates#

There are two ways to propagate the updates: synchronously and asynchronously.

Synchronous replication#

In **synchronous replication**, the node replies to the client to indicate the update is complete—only after receiving acknowledgments from the other replicas that they've also performed the update on their local storage. This guarantees that the client is able to view the update in a subsequent read after acknowledging it, no matter which replica the client reads from.

Furthermore, synchronous replication provides increased **durability**. This is because the update is not lost even if the leader crashes right after it acknowledges the update.

However, this technique can make writing requests slower. This is because the leader has to wait until it receives responses from all the replicas.

Created with Fabric.js 3.6.6

A distributed system with a leader-follower architecture, where the Primary node is the leader while Secondary nodes are follower

1 of 7

Asynchronous replication#

In **asynchronous replication**, the node replies to the client as soon as it performs the update in its local storage, without waiting for responses from the other replicas.

This technique increases performance significantly for write requests. This is because the client no longer pays the penalty of the network requests to the other replicas.

However, this comes at the cost of reduced consistency and decreased **durability**. After a client receives a response for an update request, the client might read older (stale) values in a subsequent read. This is only possible if the operation happens in one of the replicas that have not yet performed the update. Moreover, if the leader node crashes right after it acknowledges an update, and the propagation requests to the other replicas are lost, any acknowledged update is eventually lost.

Created with Fabric.js 3.6.6

A distributed system with leader-follower architecture, where the Primary node is the leader while Secondary nodes are follower

1 of 7

> Most widely used databases, such as PostgreSQL or MySQL, use a single-master replication technique that supports both asynchronous and synchronous replication.

Advantages of single-master replication#

- It is simple to understand and implement
- Concurrent operations serialized in the leader node, remove the need for more complicated, distributed concurrency protocols. In general, this property also makes it easier to support transactional operations
- It is scalable for read-heavy workloads, because the capacity for reading requests can be increased by adding more read replicas

Disadvantages of single-master replication#

- It is not very scalable for write-heavy workloads, because a single node (the leader)'s capacity determines the capacity for writes
- It imposes an obvious trade-off between performance, durability, and consistency
- Scaling the read capacity by adding more follower nodes can create a bottleneck in the network bandwidth of the leader node, if there's a large number of followers listening for updates
- The process of failing over to a follower node when the leader node crashes, is not instant. This may create some downtime and also introduce the risk of errors

Failover#

Failover is when the leader node crashes and a follower node takes over.

Created with Fabric.js 3.6.6

Node B is the leader, while Node C, D, and A are followers.

1 of 3

When the *master* node crashes, we need to choose another *master* node. Following are the approaches to perform *failover*.

When the leader node crashes, we need to choose another leader node. We can use the following approaches to perform a failover.

Approaches for performing failover#

In general, there are two approaches to perform a failover: **manual** and **automated**.

Manual approach#

In the manual approach, the operator selects the new leader node and instructs all the nodes accordingly. This is the safest approach, but it incurs significant downtime.

Automated approach#

An alternative is an automated approach, where follower nodes detect that the leader node has crashed (e.g., via periodic heartbeats), and attempt to elect a new leader node. This is faster but is quite risky. This is because there are many different ways in which the nodes can get confused and arrive at an incorrect state.

> The chapter about consensus will cover this topic, called leader election, in more detail.

Replication

Next

Multi-Master Replication Algorithm

Mark as Completed

Report an Issue