

Log In

Join

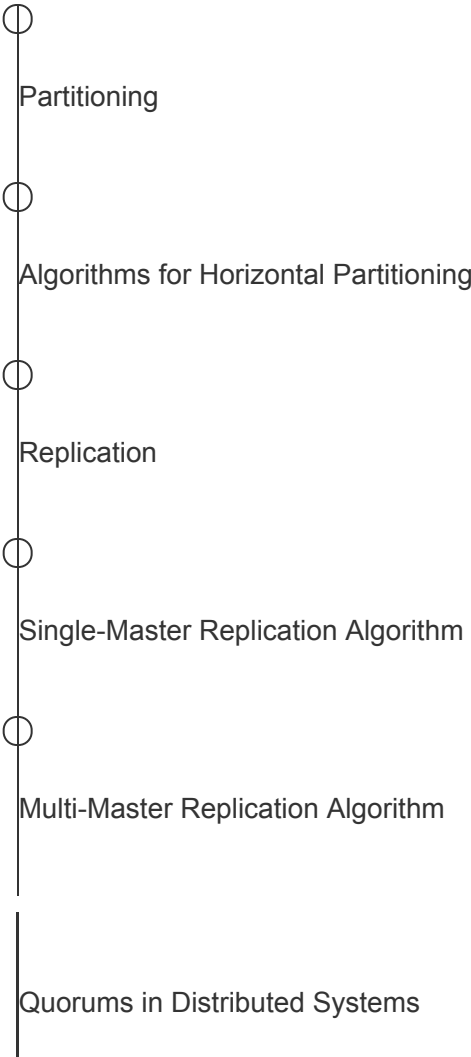
Back To Module Home


Distributed Systems

0% completed

Introduction to Distributed Systems

Basic Concepts and Theorems





	Safety Guarantees in Distributed Systems
	ACID Transactions
	The CAP Theorem
	Consistency Models
	CAP Theorem's Consistency Model
	Isolation Levels and Anomalies
	Prevention of Anomalies in Isolation Levels
	Consistency and Isolation
	Hierarchy of Models
	Why All the Formalities?
	Quiz

Conclusion

Mark Module as Completed

Quorums in Distributed Systems

Look at the concept of quorums and see how they solve low availability problems in synchronous replication.

We'll cover the following

- The problem in synchronous replication
 - Possible solution
- Quorums

The main pattern we've seen so far is this: writes are performed to all the replica nodes, while reads are performed to one of them. When we ensure that writes are performed to all of them *synchronously* before replying to the client, we guarantee that the subsequent reads see all the previous writes—regardless of the node that processes the read operation.

The problem in synchronous replication#

Availability is quite low for write operations, because the failure of a single node makes the system unable to process writes until the node recovers.

Possible solution#

To solve this problem, we can use the reverse strategy. That is, we write data only to the node that is responsible for processing a write operation, but process read operations by

reading from all the nodes and returning the latest value.

This increases the availability of *writes* significantly but decreases the availability of *reads* at the same time. So, we have a trade-off that needs a mechanism to achieve a balance. Let's see that mechanism.

Quorums#

A useful mechanism to achieve a balance in this trade-off is to use **quorums**.

Let's consider an example. In a system of three replicas, we can say that writes need to complete in two nodes (as a quorum of two), while reads need to retrieve data from two nodes. This way, we can be sure that the reads will read the latest value. This is because at least one of the nodes in the *read quorum* will also be included in the latest *write quorum*.

This is based on the fact that in a set of three elements, two subsets of two elements must have at least one common element.

A past paper introduced this technique as a **quorum-based voting protocol** for replica control.

In general, in a system that has a total of V replicas, every read operation should obtain a read quorum of V_r replicas. Meanwhile, a write operation should obtain a write quorum of V_w replicas. The values of these quorums should obey the following properties:

- $V_r + V_w > V$
- $V_w > V/2$

The first rule ensures that a data item is not read and written by two operations concurrently.

The second rule ensures that at least one node receives both of the two write operations

and imposes an order on them. This means that two write operations from two different operations cannot occur concurrently on the same data item.

Both of the rules together guarantee that the associated distributed database behaves as a centralized, one-replica database system.

The concept of a quorum is really useful in distributed systems that have multiple nodes.

The concept of a quorum is used extensively in other areas, like distributed transactions or consensus protocols.

Back

[Multi-Master Replication Algorithm](#)

Next

[Safety Guarantees in Distributed Syst...](#)

[Mark as Completed](#)

[Report an Issue](#)