Strong Consistency

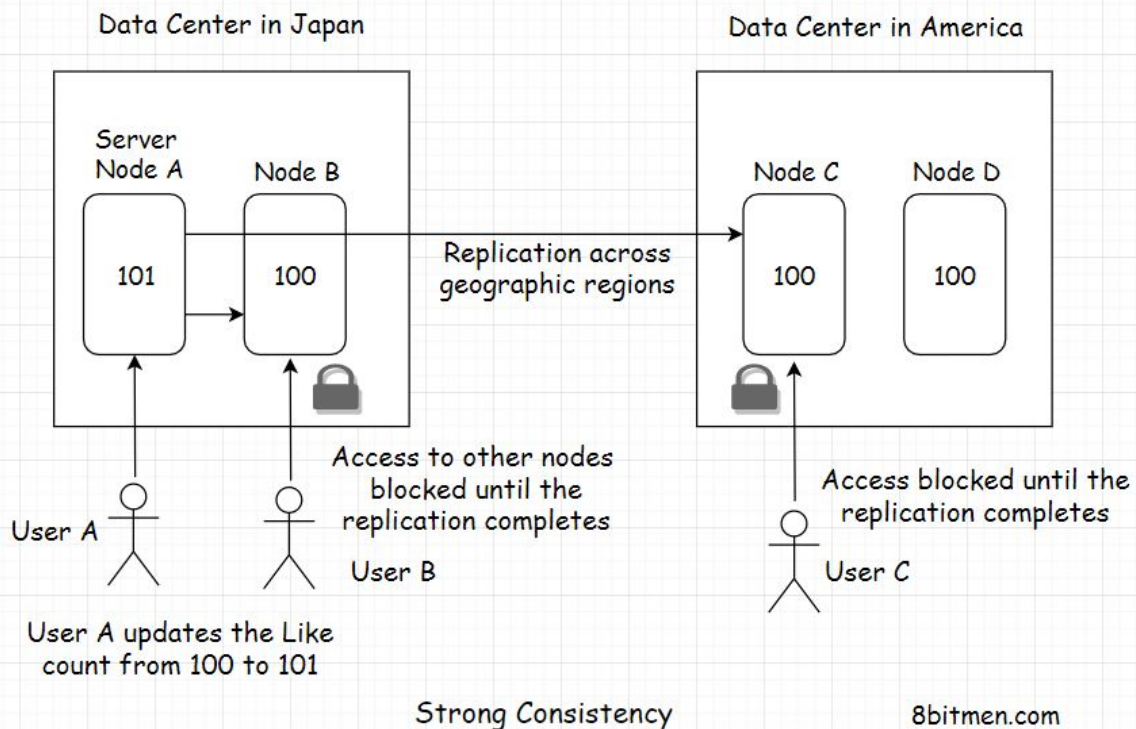In this lesson, we will discuss Strong Consistency.

What Is Strong Consistency?

Strong Consistency simply means the data has to be strongly consistent at all times. All the server nodes across the world should contain the same value of an entity at any point in time. And the only way to implement this behaviour is by locking down the nodes when being updated.

Real World Use Case

Let's continue the same Eventual Consistency example from the previous lesson. To ensure *Strong Consistency* in the system, when the user in Japan likes the post, all the nodes across different geographical zones have to be locked down to prevent any concurrent updates.

This means at one point in time, only one user can update the post *"Like"* counter value.

Data Center in Japan · Data Center in America

Server Node A · Node B · Node C · Node D

101 · 100 · 100 · 100

Replication across geographic regions

Access to other nodes blocked until the replication completes

Access blocked until the replication completes

User A · User B · User C

User A updates the Like count from 100 to 101

Strong Consistency · 8bitmen.com

So, once the user in Japan updates the "*Like*" counter from 100 to 101. The value gets replicated globally across all nodes. Once all the nodes reach a consensus, the locks get lifted.

Now, other users can *Like* the post. If the nodes take a while to reach a consensus, they have to wait until then.

Well, this is surely not desired in case of social applications. But think of a stock market application where the users are seeing different prices of the same stock at one point in time and updating it concurrently. This would create chaos.

Therefore, to avoid this confusion we need our systems to be *Strongly Consistent*. The nodes have to be locked down for updates.

Queuing all the requests is one good way of making a system *Strongly Consistent*. Well, the implementation is beyond the scope of this course. Though we will discuss a theorem called the *CAP theorem* which is key to implementing these consistency models.

So, by now I am sure you would have realized that picking the *Strong Consistency* model hits the capability of the system to be *Highly Available*.

The system while being updated by one user does not allow other users to perform concurrent updates. This is how strongly consistent ACID transactions are implemented.

ACID Transaction Support

Distributed systems like *NoSQL* databases which scale horizontally on the fly don't support *ACID transactions* globally & this is due to their design. The whole reason for the development of *NoSQL* tech is the ability to be *Highly Available* and *Scalable*. If we have to lock down the nodes every time, it becomes just like *SQL*.

So, NoSQL databases don't support *ACID transactions* and those that claim to, have terms and conditions applied to them.

Generally, the transaction support is limited to a geographic zone or an entity hierarchy. Developers of the tech make sure that all the Strongly consistency entity nodes reside in the same geographic zone to make the *ACID transactions* possible.

Well, this is pretty much about *Strong Consistency*. Now let's take a look into the *CAP Theorem*