Strong Consistency

In this lesson, we will discuss strong consistency.

We'll cover the following

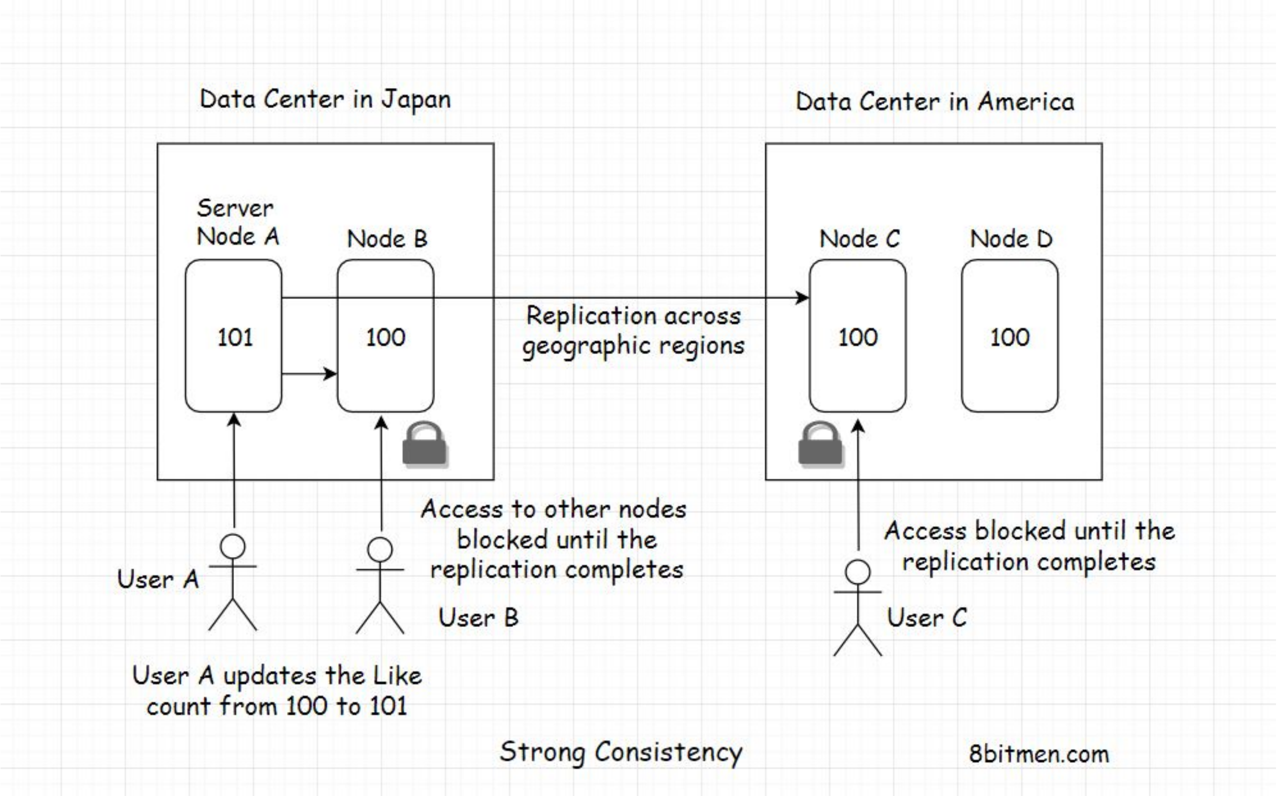
What is strong consistency?#

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. The only way to implement this behavior is by locking down the nodes as they are 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 that, 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 desirable for 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. However, we will discuss a theorem called the *CAP theorem* which is key to implementing these consistency models.

So, by now, I am sure you have realized that picking the *strong*

So, by now, I am sure you have realized that picking the *strong consistency* model hits the capability of the system to be *highly available*.

While being updated by one user, the system 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, and 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 consistent entity nodes reside in the same geographic zone to make the *ACID transactions* possible.

Well, this is pretty much it for *strong consistency*. Now, let’s take a look into the *CAP theorem*

← Back
Eventual Consistency

Next →
CAP Theorem

☑ Mark as Completed

Report an Issue