Eventual Consistency

In this lesson, we will discuss eventual consistency.

We'll cover the following

- What is eventual consistency?
- Real-world use case

What is eventual consistency?#

Eventual consistency is a consistency model that enables the data store to be *highly available*. It is also known as *optimistic replication* and is key to distributed systems.

So, how exactly does it work?

We will break this down with the help of a use case.

Real-world use case#

Think of a popular microblogging site deployed across the world in different geographical regions like Asia, America, Europe, and so on. Moreover, each geographical region has multiple data center zones: North, East, West, and South. Furthermore, each of the zones has multiple clusters which have multiple server nodes running

clusters which have multiple server nodes running.

So, we have many datastore nodes spread across the world which the micro-blogging site uses for persisting data.

Since there are so many nodes running, there is no *single point of failure*. The data store service is *highly available*. Even if a few nodes go down the persistence service as a whole is still up.

Alright, now let's say a celebrity makes a post on the website, and everybody around the world starts liking it.

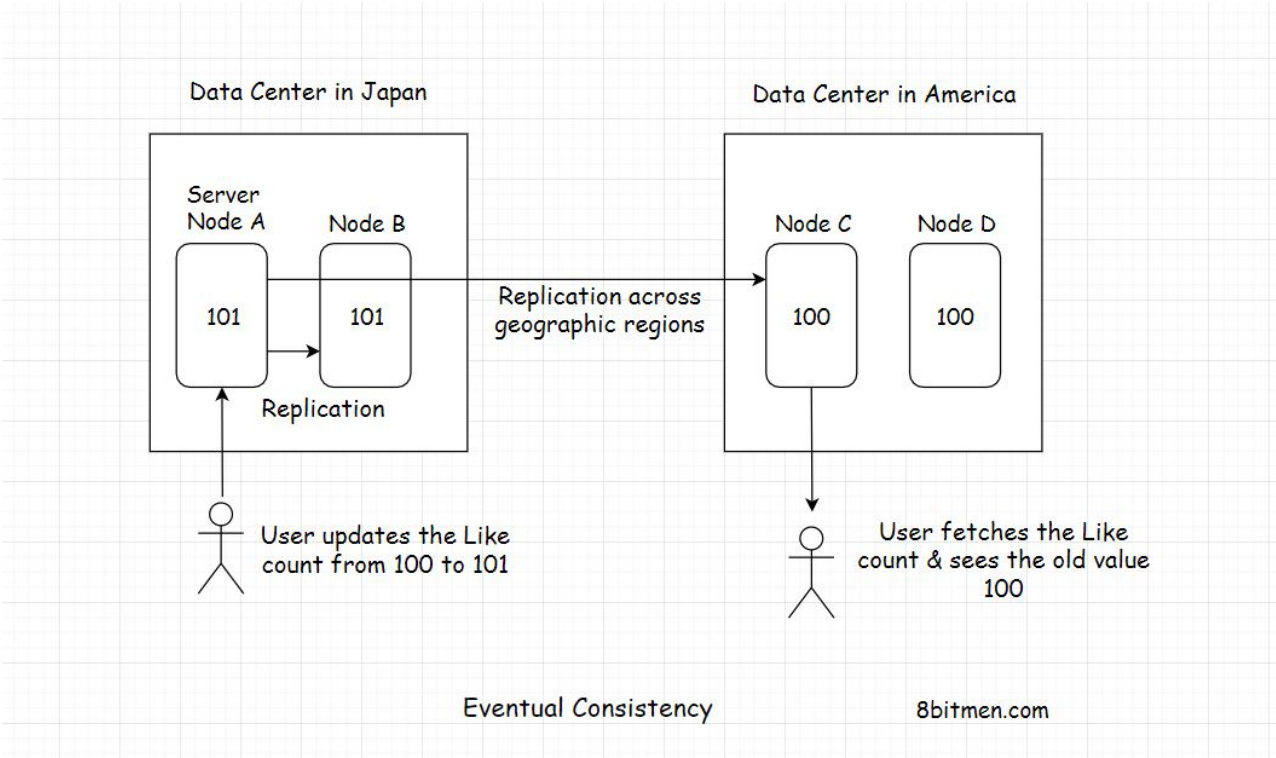
At a point in time, a user in Japan likes the post which increases the "Like" count of the post from say 100 to 101. At the same time, a user in America, a different geographical zone clicks on the post and sees the "Like" count as 100, not 101.

Why did this happen?

Simply, because the new updated value of the post's "Like" counter needs some time to move from Japan to America and update the server nodes running there.

Although the value of the counter at that point in time was 101, the user in America sees the old inconsistent value.

However, when they refresh their web page after a few seconds, the "Like" counter value shows as 101. So, the data was initially inconsistent but eventually became consistent across the server nodes deployed around the world. This is what *eventual consistency* is.



Let's take it one step further. What if, at the same point in time, both the users in Japan and America *Like* the post, and a user in another geographic zone, say Europe, accesses the post.

All the nodes in different geographic zones have different post values, and they will take some time to reach a consensus.

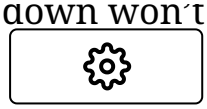
The upside of eventual consistency is that the system can add new nodes on the fly without the need to block any of them, the nodes are available to the end-users to make an update at all times.

Millions of users across the world can update the values at the same time without having to wait for the system to reach a common final value across all nodes before they make an update. This feature enables the system to be *highly available*.

Eventual consistency is suitable for use cases where the accuracy of values doesn't matter much, like in the use case above.

Other eventual consistency use cases can be when keeping the count of users watching a Live video stream online. When dealing with massive amounts of analytics data, a couple of counts won't matter

amounts of analytics data, a couple of counts up and down won't matter much.



However, there are use cases where the data has to be laser accurate, like in banking and stock markets. We just cannot have our systems to be *eventually consistent*, and we need *strong consistency*.

Let's discuss this in the next lesson.

← Back

Multi-Model Databases

Next →

Strong Consistency

Mark as Completed

Report an Issue