## **Distributed Database Systems**

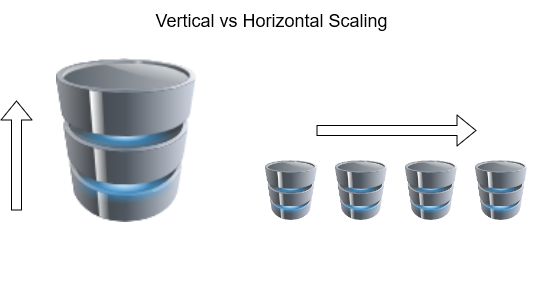
In a NoSQL type distributed database system, multiple computers, or nodes, work together to give an impression of a single working database unit to the user. They store the data in these multiple nodes. And each of these nodes runs an instance of the database server and they communicate with each other in some way.

When a user wants to write to the database, the data is appropriately written to a node in the distributed database. The user may not be aware of where the data is written. Similarly, when a user wants to retrieve the data, it connects to the nearest node in the system which retrieves the data for it, without the user knowing about this.

This way, a user simply interacts with the system as if it is interacting with a single database. Internally the nodes communicate with each other, retrieving data that the user is looking for, from the relevant node, or storing the data provided by the user.

Now, the benefits of a distributed system are quite obvious. With the increase in traffic from the users, we can easily scale our database by adding more nodes to the system. Since these nodes are commodity hardware, they are relatively cheaper than adding more resources to each of the nodes individually. That is, horizontal scaling is cheaper than vertical scaling.

This horizontal scaling makes replication of data cheaper and easier. This means that now the system can easily handle more user traffic by appropriately distributing the traffic amongst the replicated nodes that.



So now the real problem comes while choosing the appropriate distribution system for a task. To answer such a question, we need to understand the CAP theorem.

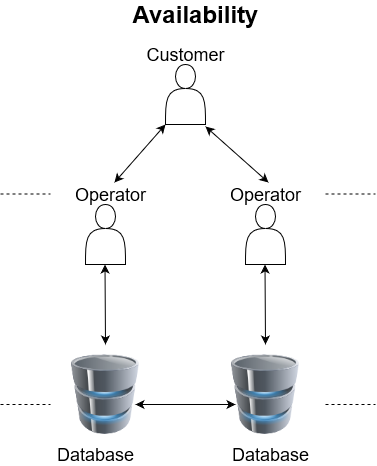
## **Understanding CAP theorem with an Example**

CAP theorem, also known as Brewer’s theorem, stands for Consistency, Availability and Partition Tolerance. But let’s try to understand each, with an example.

#### **Availability**

Imagine there is a very popular mobile operator in your city and you are its customer because of the amazing plans it offers. Besides that, they also provide an amazing customer care service where you can call anytime and get your queries and concerns answered quickly and efficiently. Whenever a customer calls them, the mobile operator is able to connect them to one of their customer care operators.

The customer is able to elicit any information required by her/him about his accounts like balance, usage, or other information. We call this **Availability**because every customer is able to connect to the operator and get the information about the user/customer.

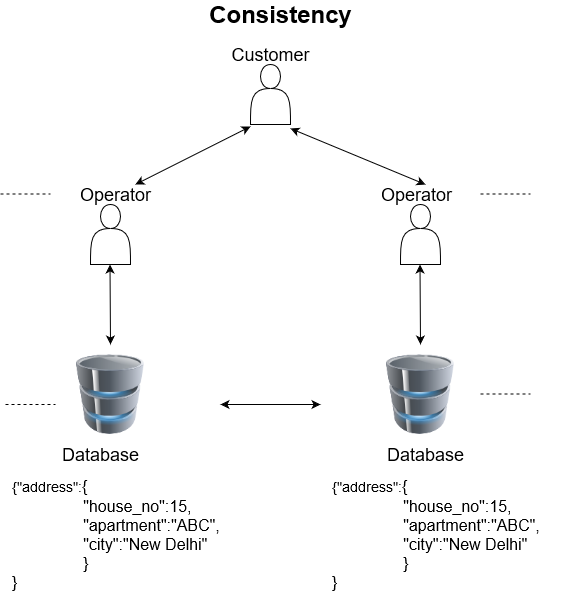


#### **Consistency**

Now, you have recently shifted to a new house in the city and you want to update your address registered with the mobile operator. You decide to call the customer care operator and update it with them. When you call, you connect with an operator. This operator makes the relevant changes in the system. But once you have put down the phone, you realize you told them the correct street name but the old house number (old habits die hard!).

So you frantically call the customer care again. This time when you call, you connect with a different customer care operator but they are able to access your records as well and know that you have recently updated your address. They make the relevant changes in the house number and the rest of the address is the same as the one you told the last operator.

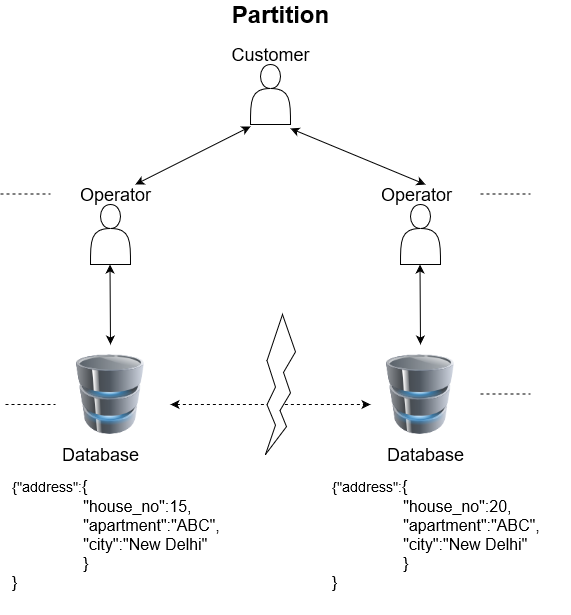
We call this as **Consistency**because even though you connect to a different customer care operator, they were able to retrieve the same information.



#### Partition tolerance

Recently you have noticed that your current mobile plan does not suit you. You do not access that much mobile data any longer because you have good wi-fi facilities at home and at the office, and you hardly step outside anywhere. Therefore, you want to update your mobile plan. So you decide to call the customer care once again.

On connecting with the operator this time, they tell you that they have not been able to update their records due to some issues. So the information lying with the operator might not be up to date, therefore they cannot update the information. We can say here that the service is broken or there is no **Partition tolerance.**



## **Understanding the Terms of the CAP theorem**

Now let’s take up these terms one by one and try to understand them in a more formal manner.

### Consistency

Consistency means that the user should be able to see the same data no matter which node they connect to on the system. This data is the most recent data written to the system. So if a write operation has occurred on a node, it should be replicated to all its replicas. So that whenever a user connects to the system, they can see that same information.

However, having a system that maintains consistency instantaneously and globally is near impossible. Therefore, the goal is to make this transition fast enough so that it is hardly noticeable.

Consistency is of importance when it is required that all the clients or users view the same data. This is important in places that deal with financial or personal information. For example, your bank account should reflect the same balance whether you view it from your PC, tablet, or smartphone!

### Availability

Availability means that every request from the user should elicit a response from the system. Whether the user wants to read or write, the user should get a response even if the operation was unsuccessful. This way, every operation is bound to terminate.

For example, when you visit your bank’s ATM, you are able to access your account and its related information. Now even if you go to some other ATM, you should still be able to access your account. If you are only able to access your account from one ATM and not another, this means that the information is not available with all the ATMs.

Availability is of importance when it is required that the client or user be able to access the data at all times, even if it is not consistent. For example, you should be able to see your friend’s Whatsapp status even if you are viewing an outdated one due to some network failure.

### Partition Tolerance

Partition refers to a communication break between nodes within a distributed system. Meaning, if a node cannot receive any messages from another node in the system, there is a partition between the two nodes. Partition could have been because of network failure, server crash, or any other reason.

So, if Partition means a break in communication then Partition tolerance would mean that the system should still be able to work even if there is a partition in the system. Meaning if a node fails to communicate, then one of the replicas of the node should be able to retrieve the data required by the user.

This is handled by keeping replicas of the records in multiple different nodes. So that even if a partition occurs, we are able to retrieve the data from its replica. As you must have guessed already, partition tolerance is a must for any distributed database system.

## **What is the CAP Theorem?**

In the last section, you understood what each term means in the CAP theorem. Now let us understand the theorem itself.

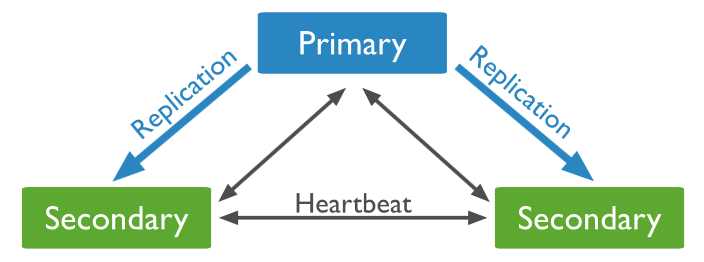
The CAP theorem states that a distributed database system has to make a tradeoff between Consistency and Availability when a Partition occurs.

A distributed database system is bound to have partitions in a real-world system due to network failure or some other reason. Therefore, partition tolerance is a property we cannot avoid while building our system. So a distributed system will either choose to give up on Consistency or Availability but not on Partition tolerance.

For example in a distributed system, if a partition occurs between two nodes, it is impossible to provide consistent data on both the nodes and availability of complete data. Therefore, in such a scenario we either choose to compromise on Consistency or on Availability. Hence, a NoSQL distributed database is either characterized as CP or AP. CA type databases are generally the monolithic databases that work on a single node and provide no distribution. Hence, they require no partition tolerance.

## **Understanding CP with MongoDB**

Let’s try to understand how a distributed system would work when it decides to give up on Availability during a partition with the help of MongoDB.



MongoDB is a NoSQL database that stores data in one or more Primary nodes in the form of JSON files. Each Primary node has multiple replica sets that update themselves asynchronously using the operation log file of their respective primary node. The replica set nodes in the system send a heartbeat (ping) to every other node to keep track if other replicas or primary nodes are alive or dead. If no heartbeat is received within 10 seconds, then that node is marked as inaccessible.

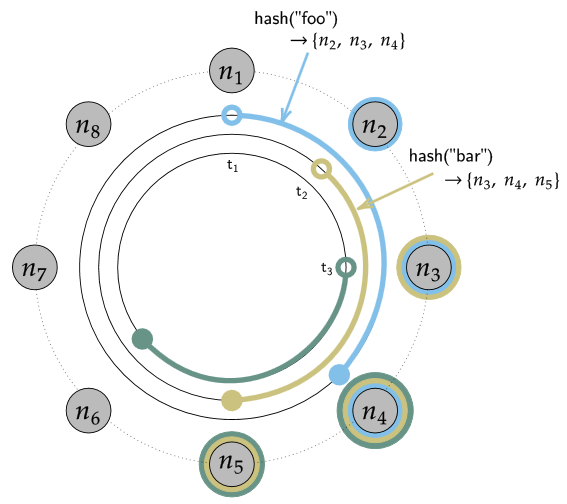
If a Primary node becomes inaccessible, then one of the secondary nodes needs to become the primary node. Till a new primary is elected from amongst the secondary nodes, the system remains unavailable to the user to make any new write query. Therefore, the MongoDB system behaves as a Consistent system and compromises on Availability during a partition.

## **Understanding AP with Cassandra**

Now let’s also look at how a system compromises on Consistency. For this, we will look at the Cassandra database which is called a highly available database.

Cassandra is a peer-to-peer system. It consists of multiple nodes in the system. And each node can accept a read or write request from the user. Cassandra maintains multiple replicas of data in separate nodes. This gives it a masterless node architecture where there are multiple points of failure instead of a single point.

The replication factor determines the number of replicas of data. If the replication factor is 3, then we will replicate the data in three nodes in a clockwise manner.



A situation can occur where a partition occurs and the replica does not get an updated copy of the data. In such a situation the replica nodes will still be available to the user but the data will be inconsistent. However, Cassandra also provides eventual consistency. Meaning, all updates will reach all the replicas eventually. But in the meantime, it allows divergent versions of the same data to exist temporarily. Until we update them to the consistent state.

Therefore, by allowing nodes to be available throughout and allowing temporarily inconsistent data to existing in the system, Cassandra is an AP database that compromises on consistency.

*Note that I have considered the MongoDB and Cassandra databases to be in their default settings.*

*Reference:* [*https://www.analyticsvidhya.com/blog/2020/08/a-beginners-guide-to-cap-theorem-for-data-engineering/*](https://www.analyticsvidhya.com/blog/2020/08/a-beginners-guide-to-cap-theorem-for-data-engineering/)