DBA

MongoDB and CAP Theorem

Apr, 2024

The CAP Theorem suggests that any distributed database can satisfy only two of the three properties, i.e. **C**onsistency, **A**vailability and **P**artition tolerance. So, a database can be either CA, CP or AP, but not CAP compliant. In this article, let's explore more about CAP and how MongoDB balances the three properties in its applications.

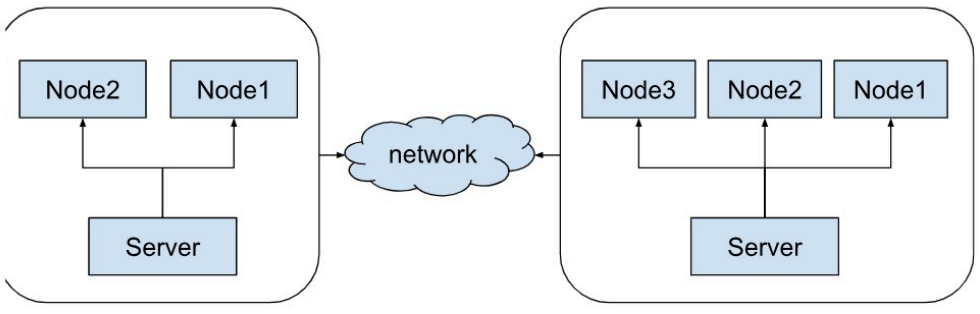
Introduction to CAP Theorem

Before getting into the CAP Theorem, let’s understand what a distributed system/database is.

Distributed database

A distributed system provides scalability and flexibility in storing and retrieving data. Traditional databases were only vertically scalable, which meant after a certain limit, the capacity (RAM/storage) could not be enhanced. With today’s enterprises more focused on data, traditional storage systems are not enough as they can’t scale beyond a certain level, and are not necessarily highly available. If there is only one server that handles multiple requests, and if that fails - then the entire system collapses.
Distributed systems provide a set of servers (think of them as storage systems) spread across different places, and are connected by a network. They follow a master-slave architecture, N-tier architecture or peer-to-peer architecture, so that multiple requests from clients can be handled by different servers. So, if server1 is busy, the next request can be diverted to server2 and so on. This way, a lot of requests can be parallelly processed and the system is always available.

Distributed systems connected over network



So, generally we would think that a distributed system would always be available and since it is distributed, it would have more fault tolerance, i.e. if one system goes down due to network issues, the others would work without hindrance, and data would be eventually consistent, as the systems are connected to each other to send/receive updates.

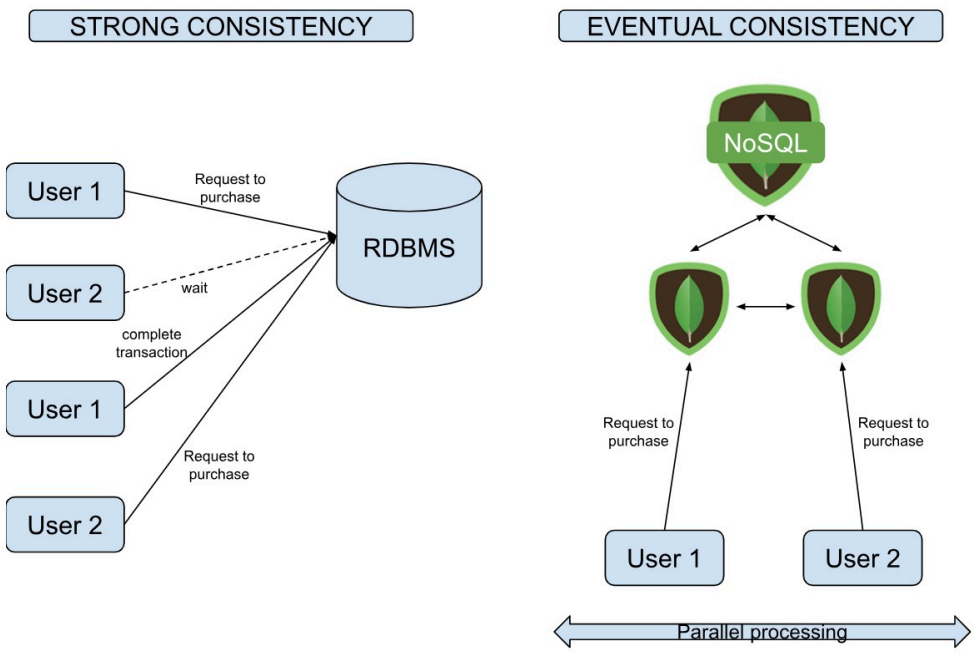
But!

Distributed systems are more complex than that. As per Brewer’s CAP Theorem, a distributed database can only achieve 2 out of the 3 properties. Let us learn what each of the properties mean.

Consistency

DBA

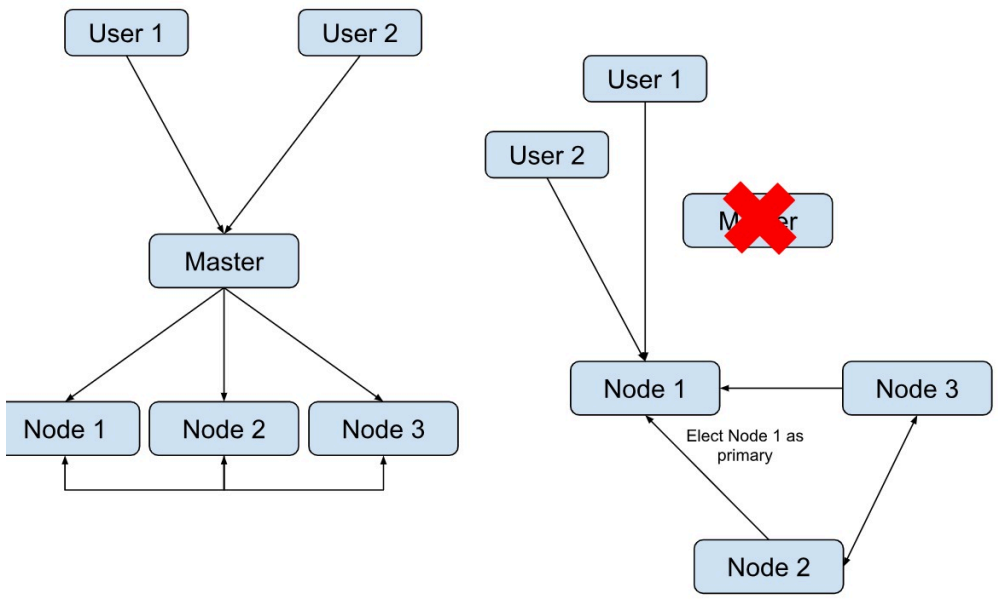
database, all the systems might not be immediately up to date. Consider an example where a user purchases a laptop. The inventory has 2 of the same products, and around the same time another user wants to purchase the same. Now, since both operations are happening in parallel, there is going to be a while before the two nodes that are processing the requests are in sync. Hence, both the users see 2 of the products, before the 2nd user gets the updated number as 1.



Availability

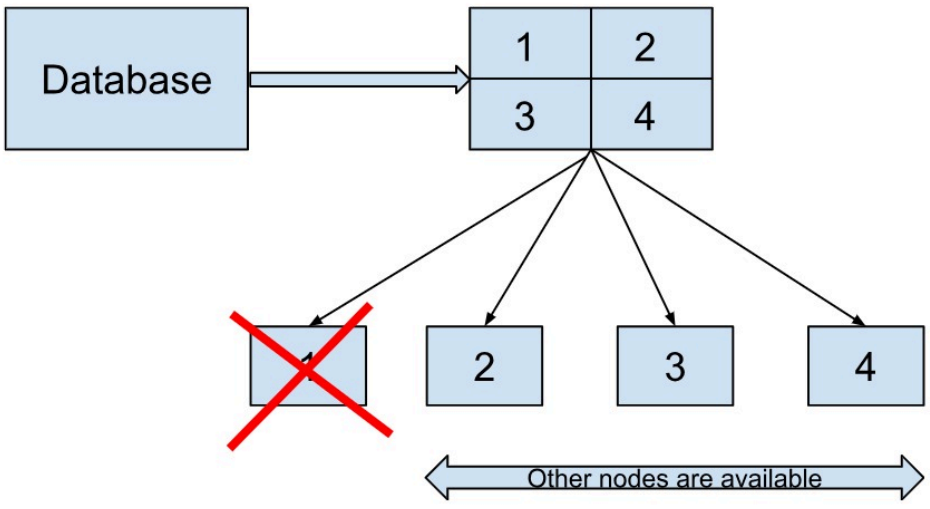
An available system always gives a response to the user for their queries, even if one or few of its nodes are down.

However, there might be times when the system is unavailable due to some sync or maintenance activities. At this time, the system may not be available to provide an immediate response to the user query. For example, if a database is designed as a master-slave architecture, and the master goes down, the system will take a while to elect the new primary. At this time, the system will be unavailable to give response to user queries.



DBA

arbitrarily partitions the affected nodes, and the others will perform as usual. At this time, nodes are disconnected from the affected node partition until it recovers.



NoSQL databases can guarantee 2 out of the 3 properties while leaving the third one out. For example, Cassandra is a wide-column database with a masterless architecture, thus having multiple nodes that always work. So, while Cassandra provides AP (Availability and Partition tolerance), it cannot guarantee Consistency as all the nodes work in parallel.
MongoDB, on the other hand, prefers Consistency and Partition tolerance over availability in general, and is said to be a CP database. We can tune MongoDB to trade off between availability and partition tolerance depending on the use cases, which we will discuss in a later section.

Evolution of the CAP Theorem

The concept of CAP came around 1998, when distributed systems first came into limelight. Earlier than that, databases were only ACID compliant. Distributed systems were BASE compliant, i.e., unlike relational database systems, which were strongly consistent, distributed systems were only eventually consistent.
Eric Brewer invented the concept and it was published in 1999 in an IEEE paper as the CAP principle. Later in 2002, Seth Gilbert and Naney Lynch of MIT proved it as a theorem.

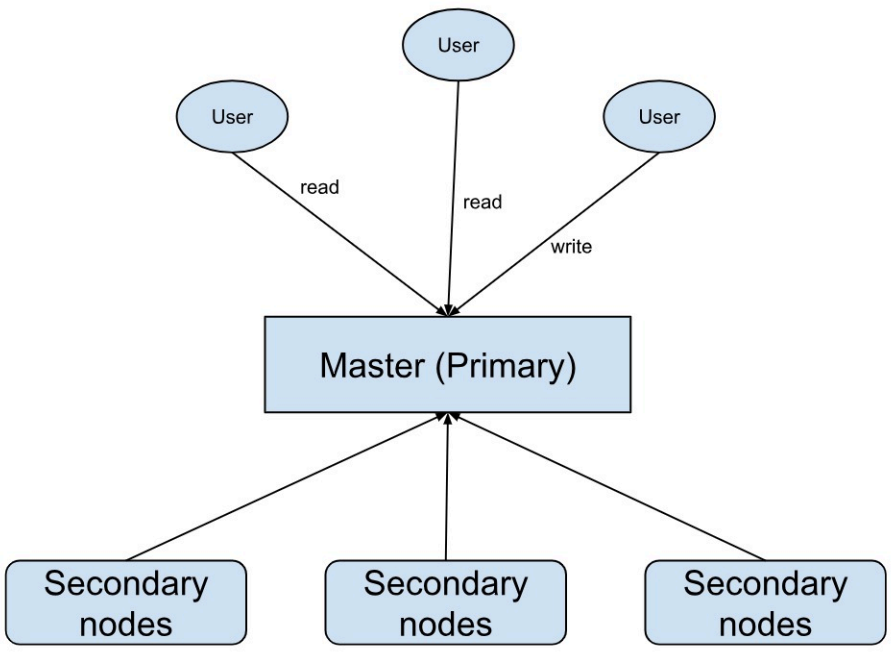
Why do distributed systems promise only 2 properties?

Consider a scenario where due to a network failure, a system is partitioned to achieve scalability and availability. Now, a system that needs to be available cannot be consistent in this case, because the node that is down is not in sync. But, if consistency is a priority, the other nodes stop certain operations until the partition is resolved, thus compromising availability of the system.
In another scenario, when there are no partitions, the system can provide both consistency and availability. Every request that comes in always gets a response (availability) and gets the most recent data (consistency).

How CAP is applied in MongoDB

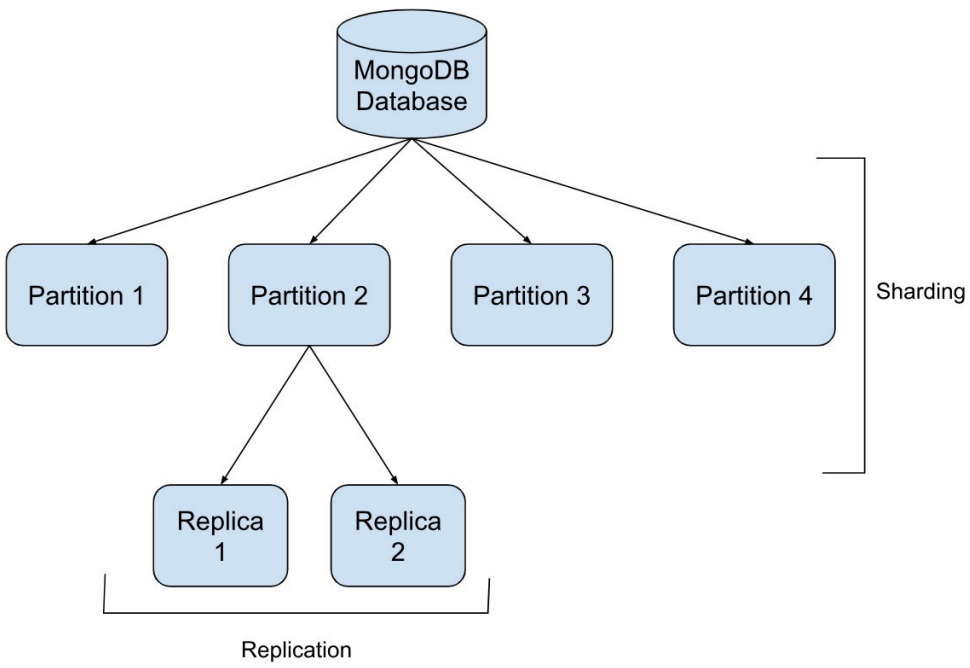
As we mentioned earlier, MongoDB has a master-slave architecture. In MongoDB, there is one primary node, which is the master, and all others are the secondary or slave nodes. All reads and writes go to the primary node by default. If the primary node fails or is

DBA



However, MongoDB, being a data platform, provides multiple ways to customize the read preferences and write concern to be able to optimize between availability and partition tolerance.

As databases today store huge volumes of data, NoSQL databases like MongoDB offer logical data partitions for faster and more efficient data retrieval. Data can be logically partitioned based on some criteria, for example, data of users between 11-18 years of age, 19-35 years of age and so on. This way, queries can be redirected based on the criteria resulting in less scanning time through the data. Further, MongoDB applies sharding and replication as effective mechanisms to cater to availability and partition tolerance, while maintaining consistency, even if it is eventual.
Due to horizontal partitioning (sharding), MongoDB provides high fault tolerance, as data can be replicated across multiple partitions.



Learn more about sharding and replication.

Conclusion

As per the CAP theorem, in a distributed environment, either of the three properties, i.e. consistency, availability and network partitioning are compromised while completely fulfilling the other 2 properties. In this article, we discussed what each of them means, how they are important to bring