CAP Theorem in DBMS: System Design Concept

The CAP theorem is an important concept in distributed database systems that helps architects and designers understand the fundamental trade-offs while designing a system.

What is CAP Theorem?

Almost a couple of decades ago, Eric Brewer introduced the idea that there exists a fundamental trade-off among consistency, availability, and partition tolerance. This trade-off is famously known as the CAP Theorem and has been extensively discussed ever since.

The CAP theorem states that distributed databases can have at most two of the three properties: Consistency, Availability, and Partition Tolerance. One can make trade-offs between the three properties based on their system requirements.

Consistency

In the CAP Theorem, Consistency means that all nodes in a distributed database system should have the same data at any given time. This ensures that all clients accessing the system will see the same data, regardless of the node they are connected to. When a client performs a read operation, they should receive the most recent write value.

To maintain consistency, the node where the write operation is performed has the responsibility of instantly replicating the data to all other nodes. Failing to maintain consistency can result in conflicting or outdated data, which can lead to errors or inconsistencies in the application. So, consistency is crucial for preserving the integrity and accuracy of the data in a distributed database system.

Availability

In the CAP Theorem, Availability means: A distributed database system should always be able to respond to client requests. So, when a client requests data, it should receive a response, regardless of the state of any individual node i.e. system must be operational at all times, even if one or more nodes are down.

Ensuring availability is important for maintaining functionality and usability. To achieve availability, a distributed system must have redundant nodes or mechanisms in place to ensure that it can continue operating even if some nodes fail. This can include techniques like load balancing, replication, and so on.

Partition Tolerance

Sometimes due to some disruption in the network (network failures, hardware issues, or other factors), nodes get separated into groups such that they can not communicate with each other. This is called network partition. So in the CAP Theorem, partition tolerance means: A distributed database system should continue operating even if there is a network partition.

This is one of the essential requirements because it allows the system to handle network partitions without compromising the overall availability or consistency! To achieve partition tolerance, we can use approaches like replication, redundancy, quorum-based systems, consensus algorithms, and load balancing.

Why CAP theorem important?

In a distributed database system, network failures can disrupt the system's ability to access and update data. So partition tolerance is a must-have requirement. But achieving partition tolerance often requires making trade-offs with consistency and availability. Why? Think about it.

Maintaining consistency can come at the cost of increased latency and reduced availability, as the system may need to wait for all nodes to agree

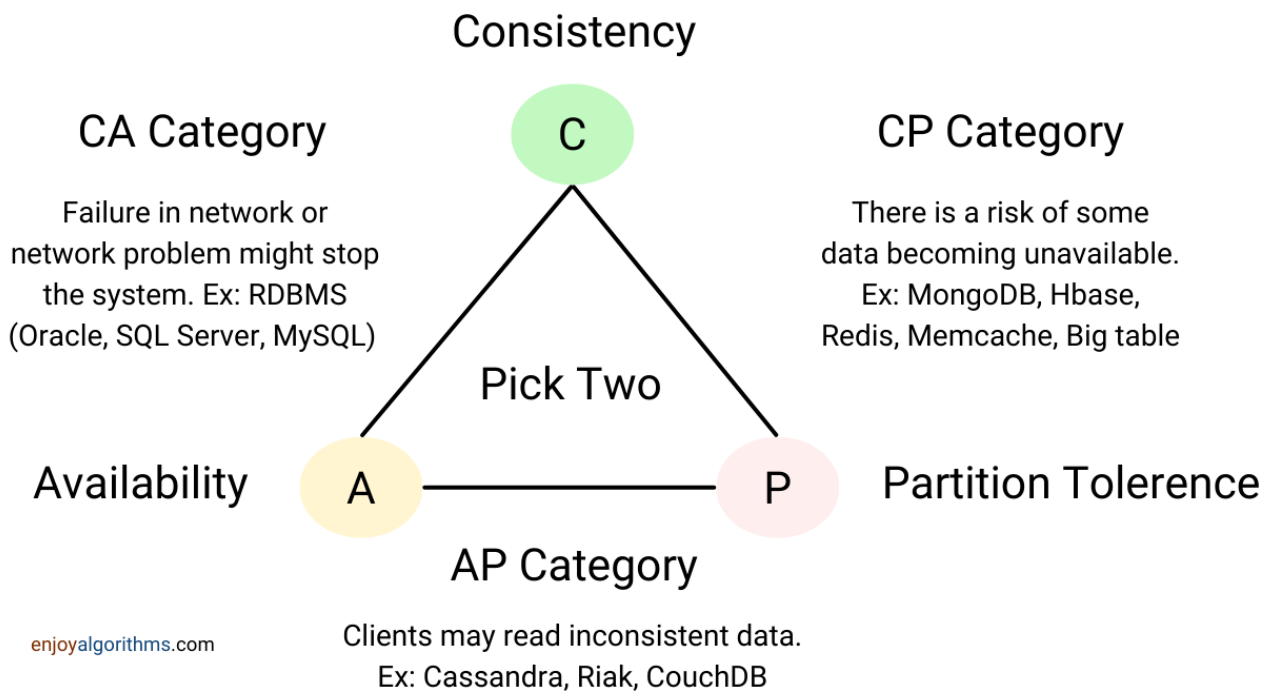
on the current state of the data before responding to client requests. On the other side, maintaining availability can lead to data inconsistencies or conflicts, as nodes may return different or stale versions of the data.

CAP Theorem Database Architecture

NoSQL databases are highly distributed and offer horizontal scalability. They can rapidly scale across a growing network of multiple interconnected nodes. Due to this, the idea of the CAP theorem is highly relevant in the case of NoSQL databases!

But as discussed above, one can only have two of the three available functionalities. Here are three different combinations of the systems based on the CAP theorem:

- **CP System:** This system focuses more on consistency and partition tolerance. So these systems are not available most of the time. When any issue occurs in the system, it has to shut down the non-consistent node until the partition is resolved, and during that time, it is not available.
- **AP System:** This type of database focuses more on availability and partition tolerance rather than consistency. When any issue occurs in the system, then it will no longer remain in a consistent state. However, all the nodes remain available, and affected nodes might return a previous version of data, and the system will take some time to become consistent.
- **CA System:** This type of database focuses more on consistency and availability across all nodes rather than partition tolerance. Fault-Tolerance is the basic necessity of any distributed system, and hence it is almost rare to use a CA type of architecture for any practical purpose.



Use Cases of CAP Theorem

MongoDB is a popular NoSQL database that focuses on a "CP" database style. By default, it maintains consistency while compromising on availability: If you read, then you will always get the most recent value of write. There is a reason: By default, MongoDB is a single-leader system, where all reads will go to the leader. Note: If required, MongoDB provides the option to enable read operations from the follower nodes. In that case, MongoDB becomes eventually consistent and it will work as an AP system!

Cassandra is another popular NoSQL database that focuses on an "AP" database style, which concentrates entirely on availability and partition tolerance rather than consistency. In other words, Cassandra will prioritize the ability to always respond to client requests and maintain partition tolerance. However, Cassandra does provide eventual consistency by figuring out all the inconsistencies in a certain period of time.

Microservices-based applications often rely on the CAP theorem to design the most efficient databases for the application. For example, if horizontal scalability is essential to the application with eventual consistency, an "AP" database like Cassandra can help meet deployment

requirements and simplify deployment. On the other hand, if the application depends heavily on data consistency, such as in a payment service, it may be better to opt for a relational database like PostgreSQL, which focuses on a "CP" database style.

Note: Sometimes, modern database systems rarely fall completely into any one of these categories because they provide configurations to select replication methods, consistency level, etc. This can have an effect on all three parameters. Even the truth is: Most of the systems have the goal to achieve all combinations of these three states. So rather than classify a database system as CP or AP, we need to think in terms of the options they provide for tuning these properties, based on our use case.

Conclusion

The CAP theorem provides a foundational understanding of the challenges and trade-offs involved in designing distributed database systems. It guides decision-making, database selection, and system design to achieve the desired balance between consistency, availability, and partition tolerance based on specific requirements.

Additional reading

CAP Twelve Year Later: How rules have changed?

Thanks to Suyash for his contribution in creating the first version of this content. If you have any queries or feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design!