CAP Theorem

Source: IBM Services (2024)

What is the CAP theorem?

The CAP theorem says that a distributed system can deliver only two of three desired characteristics: ***consistency***, ***availability*** and ***partition tolerance*** (the **'C,' 'A'** and **'P'** in CAP).

Have you ever seen an advertisement for a landscaper, house painter, or some other tradesperson that starts with the headline, "*Cheap, Fast, and Good: Pick Two*"? The CAP theorem applies a similar type of logic to distributed systems.

A distributed system is a network that stores data on more than one node (physical or virtual machines) at the same time. Because all cloud applications are distributed systems, it's essential to understand the CAP theorem when designing a cloud app so that you can choose a data management system that delivers the characteristics your application needs most.

The CAP theorem is also called Brewer's Theorem, because it was first advanced by Professor Eric A. Brewer during a talk he gave on distributed computing in 2000. Two years later, MIT professors Seth Gilbert and Nancy Lynch published a proof of "Brewer's Conjecture."

More on the ‘CAP’ in the CAP theorem

Let’s take a detailed look at the three distributed system characteristics to which the CAP theorem refers.

Consistency

Consistency means that all clients see the same data at the same time, no matter which node they connect to. For this to happen, whenever data is written to one node, it must be instantly forwarded or replicated to all the other nodes in the system before the write is deemed ‘successful.’

Availability

Availability means that any client making a request for data gets a response, even if one or more nodes are down. Another way to state this—all working nodes in the distributed system return a valid response for any request, without exception.

Partition tolerance

A partition is a communications break within a distributed system—a lost or temporarily delayed connection between two nodes. Partition tolerance means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.

CAP theorem NoSQL database types

NoSQL databases are ideal for distributed network applications. Unlike their vertically scalable SQL (relational) counterparts, NoSQL databases are horizontally scalable and distributed by design—they can rapidly scale across a growing network consisting of multiple interconnected nodes. (See "SQL vs. NoSQL Databases: What's the Difference?" for more information.)

Today, NoSQL databases are classified based on the two CAP characteristics they support:

- **CP database:** A CP database delivers consistency and partition tolerance at the expense of availability. When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved.

- **AP database:** An AP database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.)

- **CA database:** A CA database delivers consistency and availability across all nodes. It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.

We listed the CA database type last for a reason—in a distributed system, partitions can't be avoided. So, while we can discuss a CA distributed database in theory, for all practical purposes a CA distributed database can't exist. This doesn't mean you can't have a CA database for your distributed application if you need one. Many relational databases, such as PostgreSQL, deliver consistency and availability and can be deployed to multiple nodes using replication.

MongoDB and the CAP theorem

MongoDB is a popular NoSQL database management system that stores data as BSON (binary JSON) documents. It's frequently used for big data and real-time applications running at multiple different locations. Relative to the CAP theorem, MongoDB is a CP data store—it resolves network partitions by maintaining consistency, while compromising on availability.

MongoDB is a *single-master* system—each replica set can have only one primary node that receives all the write operations. All other nodes in the same replica set are secondary nodes that replicate the primary node's operation log and apply it to their own data set. By default, clients also read from the primary node, but they can also specify a read preference that allows them to read from secondary nodes.

When the primary node becomes unavailable, the secondary node with the most recent operation log will be elected as the new primary node. Once all the other secondary nodes catch up with the new master, the cluster becomes available again. As clients can't make any write requests during this interval, the data remains consistent across the entire network.

Cassandra and the CAP theorem (AP)

Apache Cassandra is an open source NoSQL database maintained by the Apache Software Foundation. It’s a wide-column database that lets you store data on a distributed network. However, unlike MongoDB, Cassandra has a masterless architecture, and as a result, it has multiple points of failure, rather than a single one.

Relative to the CAP theorem, Cassandra is an AP database—it delivers availability and partition tolerance but can't deliver consistency all the time. Because Cassandra doesn't have a master node, all the nodes must be available continuously. However, Cassandra provides *eventual consistency* by allowing clients to write to any nodes at any time and reconciling inconsistencies as quickly as possible.

As data only becomes inconsistent in the case of a network partition and inconsistencies are quickly resolved, Cassandra offers “repair” functionality to help nodes catch up with their peers. However, constant availability results in a highly performant system that might be worth the trade-off in many cases.

Microservices and the CAP theorem

Microservices are loosely coupled, independently deployable application components that incorporate their own stack—including their own database and database model—and communicate with each other over a network. As you can run microservices on both cloud servers and on-premises data centers, they have become highly popular for hybrid and multicloud applications.

Understanding the CAP theorem can help you choose the best database when designing a microservices-based application running from multiple locations. For example, if the ability to quickly iterate the data model and scale horizontally is essential to your application, but you can tolerate eventual (as opposed to strict) consistency, an AP database like Cassandra or Apache CouchDB can meet your requirements and simplify your deployment. On the other hand, if your application depends heavily on data consistency—as in an eCommerce application or a payment service—you might opt for a relational database like PostgreSQL.