CAP & PACELC theorem

19 August 2023 07:28 PM

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

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.

Consistency :

- Consistency means that every successful read request receives the result of the most recent write request.
- 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 every request receives a non-error response, without any guarantees on whether it reflects the most recent write request.
- 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.

In a distributed system, there is always the risk of a network partition. If this happens, the system needs to decide either to continue operating and compromise *data consistency*, or stop operating and compromise *availability*.

However, there is no such thing as trading off *partition tolerance* to maintain both *consistency* and *availability*.

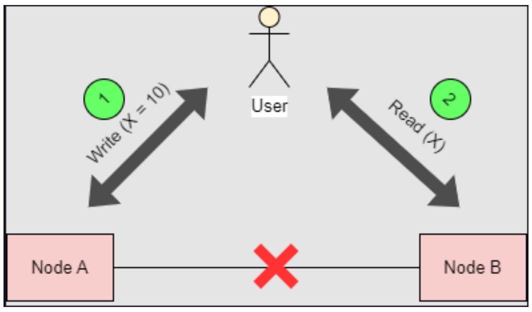
Note : It is very important to understand that *partition tolerance* is not a property we can abandon.

Final statement of the CAP theorem :

According to the final statement of the CAP theorem, a distributed system can be either *consistent* or *available* in the presence of a network partition.

Proof :

Let's attempt to prove this theorem simplistically and schematically. Let's imagine a distributed system consisting of two nodes, as shown in the illustration.



This distributed system can act as a plain register with the value of a variable *X*.

Now, let's assume that there is a network failure that results in a network partition between the two nodes of the system at some point. A user of the system performs a write, and then a read—even two different users may perform these operations.

We will examine the case where a different node of the system processes each operation. In that case, the system has two options:

- It can fail one of the operations, and break the *availability* property.

- It can process both the operations, which will return a stale value from the read and break the *consistency* property.

It cannot process both of the operations successfully, while also ensuring that the read returns the latest value that is written by the write operation. This is because the results of the write operation cannot be propagated from node A to node B due to the network partition.

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.

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.

Ex:- MongoDB, Redis, Hbase, etc..,

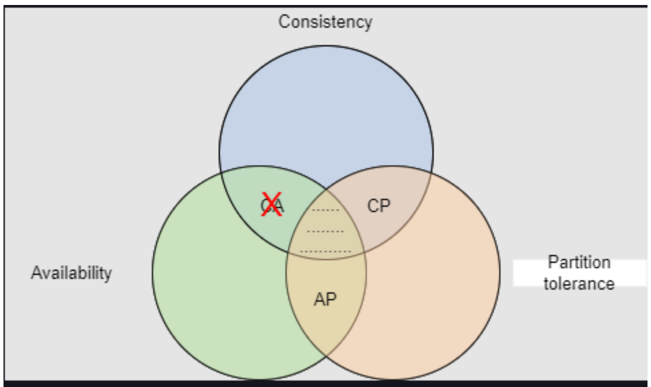
AP database: An AP database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, the AP databases typically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

Ex:- Cassandra, Couch DB, Amazon DynamoDB, Cosmos DB, etc..,

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.

Ex:- RDBMS, PostrgreSQL, etc..,

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.



Categories of distributed systems according to the CAP theorem

MISC :

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 (link resides outside ibm.com) 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.

What is missing in the CAP theorem?

We cannot avoid partition in a distributed system, therefore, according to the CAP theorem, a distributed system should choose between consistency or availability. ACID (Atomicity, Consistency, Isolation, Durability) databases, such as RDBMSs like MySQL, Oracle, and Microsoft SQL Server, chose consistency (refuse response if it cannot check with peers), while BASE (Basically Available, Soft-state, Eventually consistent) databases, such as NoSQL databases like MongoDB, Cassandra, and Redis, chose availability (respond with local data without ensuring it is the latest with its peers).

One of the major pitfalls of the **CAP Theorem** was it did not make any provision for **Performance** or **Latency**, in other words, CAP Theorem didn't provide trade-offs when the system is under normal functioning or non-partitioned. Let's try to understand this by an example:

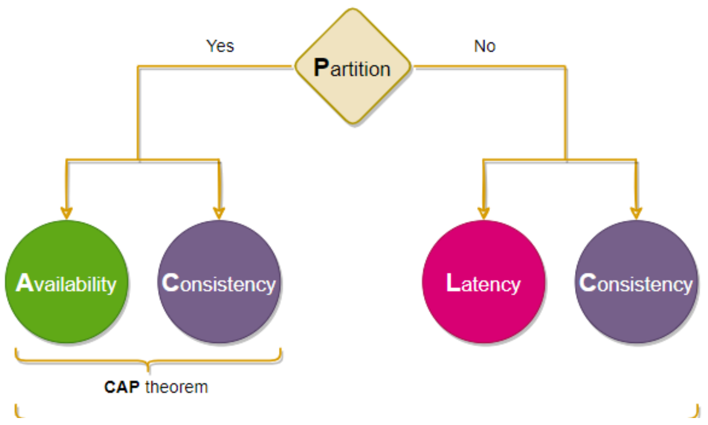
Consider a scenario, in which you are making a request to Signup up and getting the confirmation after an hour. According to the CAP theorem, this system is available but such latency is unacceptable in any real-world application.

One place where the CAP theorem is silent is what happens when there is no network partition? What choices does a distributed system have when there is no partition? PACELC theorem to the rescue.

PACELC theorem :

The PACELC theorem states that in a system that replicates data:

- if there is a partition ('P'), a distributed system can trade-off between availability and consistency (i.e., 'A' and 'C');
- else ('E'), when the system is running normally in the absence of partitions, the system can trade-off between latency ('L') and consistency ('C').



PACELC theorem

PACELC Theorem

The first part of the theorem (PAC) is the same as the CAP theorem, and the ELC is the extension. The whole thesis is assuming we maintain high availability by replication. So, when there is a failure, CAP theorem prevails. But if not, we still have to consider the tradeoff between consistency and latency of a replicated system.

Trade-off between latency and consistency :

When no network partition is present during normal operation, there's a different trade-off between *latency* and *consistency*.

To guarantee *data consistency*, the system will have to delay write operations until the data has been propagated across the system successfully, thus taking a *latency* hit.

An example of this trade-off is the single-master replication scheme. In this setting, the *synchronous replication* approach would favour *consistency* over *latency*. Meanwhile, *asynchronous replication* would reduce *latency* at the cost of *consistency*.

Categorization of distributed systems based on PACELC theorem :

Each branch of the PACELC theorem creates two sub-categories of systems.

The first part of the theorem defines the two categories we have already seen: CP and AP.

The second part defines two new categories: **EL** and **EC**.

These sub-categories are combined to form the following four categories:

- AP/EL
- CP/EL
- AP/EC
- CP/EC

A system from the AP/EL category prioritizes *availability* during a network partition and *latency* during a normal operation.
In most cases, systems are designed with an overarching principle in mind: usually either performance and availability, or data consistency. As a result, most of the systems fall into the AP/EL or CP/EC categories.

There are still systems we cannot strictly classify into these categories. This is because they have various levers that can tune the system differently when needed. Still, this theorem serves as a good indicator of the various forces at play in a distributed system.

MISC :

- Dynamo and Cassandra are PA/EL systems: They choose availability over consistency when a partition occurs; otherwise, they choose lower latency.
- BigTable and HBase are PC/EC systems: They will always choose consistency, giving up availability and lower latency.
- MongoDB can be considered PA/EC (default configuration): MongoDB works in a primary/secondaries configuration. In the default configuration, all writes and reads are performed on the primary. As all replication is done asynchronously (from primary to secondaries), when there is a network partition in which primary is lost or becomes isolated on the minority side, there is a chance of losing data that is unreplicated to secondaries, hence there is a loss of consistency during partitions. Therefore it can be concluded that in the case of a network partition, MongoDB chooses availability, but otherwise guarantees consistency. Alternately, when MongoDB is configured to write on majority replicas and read from the primary, it could be categorized as PC/EC.