CAP Theorem

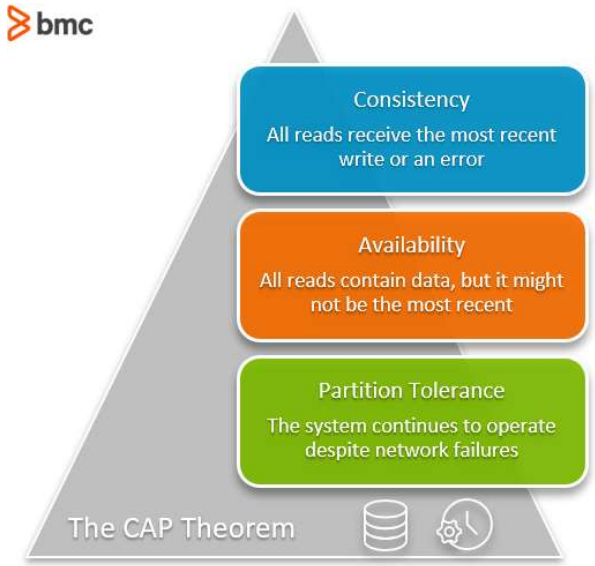
06 June 2024 22:08

CAP Theorem for Databases: Consistency, Availability & Partition Tolerance

What is the CAP theorem?

The CAP Theorem is comprised of three components (hence its name) as they relate to distributed data stores:

- **Consistency**: All reads receive the most recent write or an error.
- **Availability**: All reads contain data, but it might not be the most recent.
- **Partition Tolerance**: The system continues to operate despite network failures (ie; dropped partitions, slow network connections, or unavailable network connections between nodes.)



In the theorem, partition tolerance is a must. The assumption is that the system operates on a distributed data store so the system, by nature, operates with network partitions. Network failures will happen, so to offer any kind of reliable service, partition tolerance is necessary—the P of CAP.

That leaves a decision between the other two, C and A. When a network failure happens, one can choose to guarantee consistency or availability:

- High consistency comes at the cost of lower availability.
- High availability comes at the cost of lower consistency.

Choosing consistency and availability comes when choosing which database type to go with, such as SQL Vs NoSQL.

NoSQL:

NoSQL databases do not require a schema, and don't enforce relations between tables. All its documents are JSON documents, which are complete entities one can readily read and understand. They are widely recognized for:

- Ease-of-use
- Scalable performance
- Strong resilience

- Wide availability

Examples of NoSQL databases include:

- Cloud Fire store
- Firebase Real-time DB
- MongoDB
- Mark Logic
- Couchbase
- Cloud DB
- Amazon DynamoDB

Consistency in databases:

Consistent databases should be used when the value of the information returned needs to be accurate.

Financial data is a good example. When a user logs in to their banking institution, they do not want to see an error that no data is returned, or that the value is higher or lower than it actually is. Banking apps should return the exact value of a user's account information. In this case, banks would rely on consistent databases.

Examples of a consistent database include:

- Bank account balances
- Text messages

Database options for consistency:

- MongoDB
- Redis
- HBase

Availability in databases:

Availability databases should be used when the service is more important than the information.

An example of having a highly available database can be seen in e-commerce businesses. Online stores want to make their store and the functions of the shopping cart available 24/7 so shoppers can make purchases exactly when they need.

Database options for availability:

- Cassandra
- DynamoDB
- Cosmos DB

Some database options, like Cosmos and Cassandra, allow a user to turn a knob on which guarantee they prefer—consistency or availability.