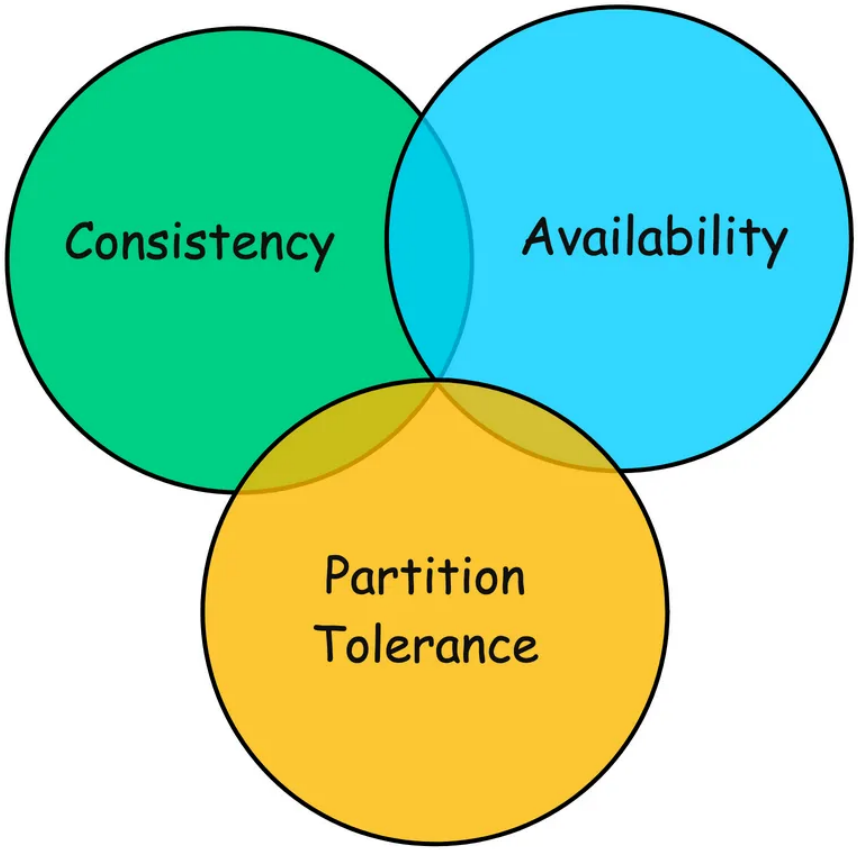
CAP Theorem Explained

#24 System Design - CAP Theorem

ASHISH PRATAP SINGH

JUL 31, 2024

The **CAP theorem**, introduced by Eric Brewer in 2000, provides a fundamental framework for understanding the **trade-offs** that must be made when designing distributed systems.



CAP stands for **Consistency**, **Availability**, and **Partition Tolerance**, and the theorem states that:

> **It is impossible for a distributed data store to simultaneously provide all three guarantees.**

- **Consistency** (C): Every read receives the most recent write or an error.
- **Availability** (A): Every request (read or write) receives a non-error response, without guarantee that it contains the most recent write.
- **Partition Tolerance** (P): The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

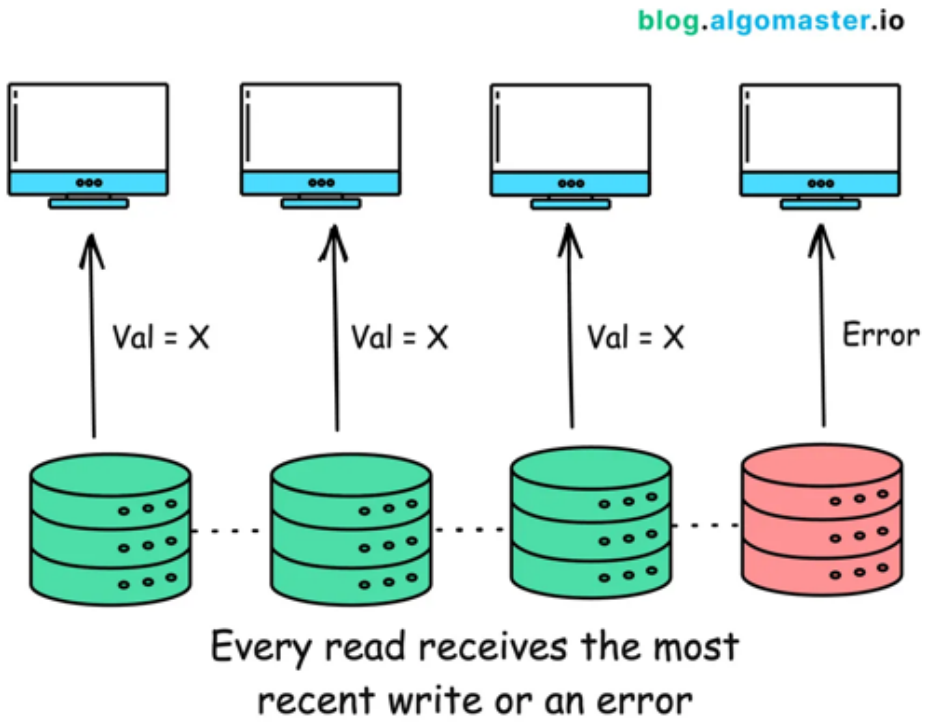
In this article, we will explore the the 3 pillars of CAP theorem, trade-offs, and practical design strategies for building resilient and scalable distributed systems.

---

3 Pillars of CAP

1. Consistency

Consistency ensures that **every read receives the most recent write or an error**. This means that all working nodes in a distributed system will return the same data at any given time.



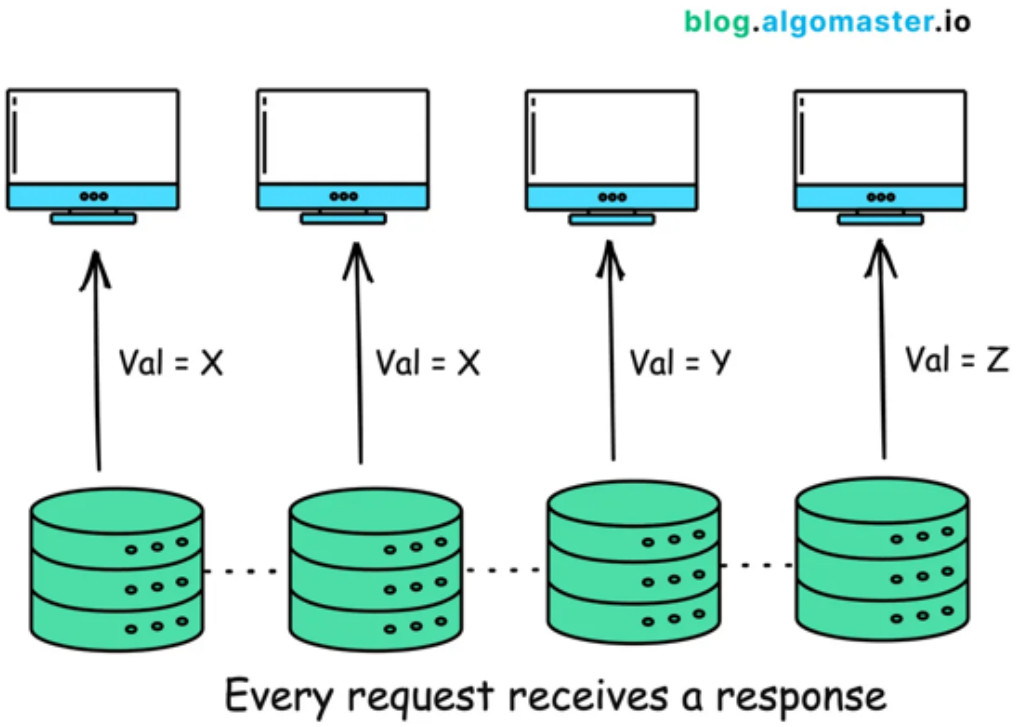
> In a consistent distributed system, if you write data to node A, a read operation

> from node B will immediately reflect the write operation on node A.

Consistency is crucial for applications where having the most up-to-date data is critical, such as financial systems, where a balance inquiry must reflect the most up-to-date state of an account.

2. Availability

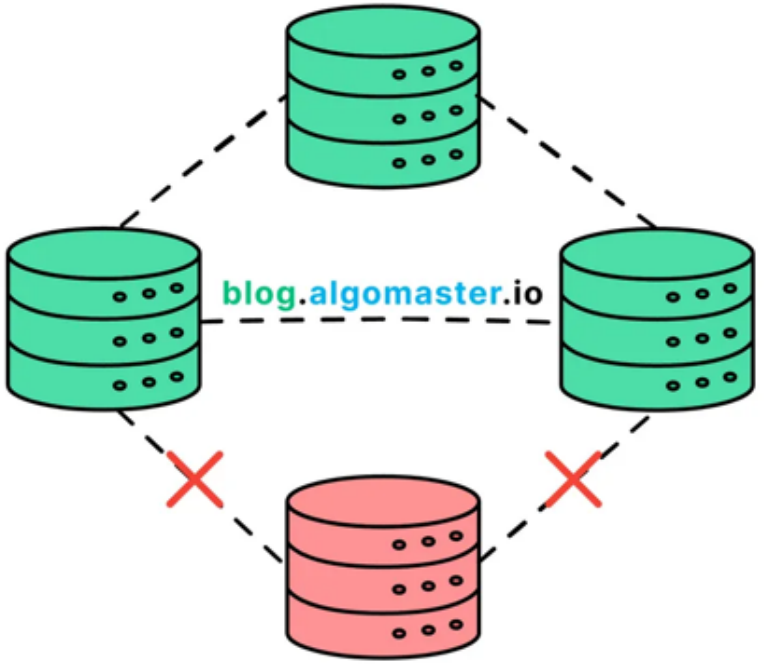
Availability guarantees that **every request** (**read or write**) **receives a response**, without ensuring that it contains the most recent write. This means that the system remains **operational** and **responsive**, even if the response from some of the nodes don't reflect most up-to-date data.



Availability is important for applications that need to remain operational at all times, such as online retail systems.

3. Partition Tolerance

Partition Tolerance means that the **system continues to function despite network partitions** where nodes cannot communicate with each other.



> A **network partition** occurs when a network failure causes a distributed system to split into two or more groups of nodes that cannot communicate with each other.

When there is a network partition, the system must choose between **Consistency** and **Availability**.

Partition Tolerance is essential for distributed systems because network failures can

and do happen. A system that tolerates partitions can maintain operations across different network segments.

The CAP Trade-Off: Choosing 2 out of 3

The CAP theorem asserts that in the presence of a network partition, a distributed system must choose between **Consistency** and **Availability**.

Let's explore these scenarios:

CP (Consistency and Partition Tolerance):

These systems prioritize consistency and can tolerate network partitions, but at the cost of availability. During a partition, the system may reject some requests to maintain consistency.

Traditional relational databases, such as MySQL and PostgreSQL, when configured for strong consistency, prioritize consistency over availability during network partitions.

> **Banking systems** typically prioritize **consistency over availability** since data accuracy is more critical than availability during network issues.

> Consider an ATM network for a bank. When you withdraw money, the system must ensure that your balance is updated accurately across all nodes (consistency) to prevent overdrafts or other errors.

AP (Availability and Partition Tolerance):

These systems ensure availability and can tolerate network partitions, but at the cost of consistency. During a partition, different nodes may return different values for the same data.

NoSQL databases like Cassandra and DynamoDB are designed to be highly available and partition-tolerant, potentially at the cost of strong consistency.

> **Amazon**'s shopping cart system is designed to always accept items, prioritizing availability.
>
> When you add items to your Amazon cart, the action almost never fails, even during high traffic periods like Black Friday.

CA (Consistency and Availability):

In the absence of partitions, a system can be both consistent and available. However,

network partitions are inevitable in distributed systems, making this combination impractical.

Example Systems: Single-node databases can provide both consistency and availability but aren't partition-tolerant. In a distributed setting, this combination is theoretically impossible.

Practical Design Strategies

Designing distributed systems requires carefully balancing these trade-offs based on application requirements.

Here are some practical strategies:

1. Eventual Consistency

For many systems, strict consistency isn't always necessary.

Eventual consistency can provide a good balance where updates are propagated to all nodes eventually, but not immediately.

> **Example:** Systems where immediate consistency is not critical, such as DNS and content delivery networks (CDNs).

2. Strong Consistency

A model ensuring that once a write is confirmed, any subsequent reads will return that value.

> **Example:** Systems requiring high data accuracy, like financial applications and inventory management.

3. Tunable Consistency

Tunable consistency allows systems to adjust their consistency levels based on specific needs, providing a balance between strong and eventual consistency.

Systems like Cassandra allow configuring the level of consistency on a per-query basis, providing flexibility.

> **Example:** Applications needing different consistency levels for different operations, such as e-commerce platforms where order processing requires strong consistency but product recommendations can tolerate eventual consistency.

4. Quorum-Based Approaches:

Quorum-based approaches use voting among a group of nodes to ensure a certain level of consistency and fault tolerance.

> **Example:** Systems needing a balance between consistency and availability, often used in consensus algorithms like Paxos and Raft.

Beyond CAP: PACELC

While CAP is foundational, it doesn't cover all scenarios.

Daniel Abadi proposed the PACELC theorem as an extension by introducing **latency** and **consistency** as additional attributes of distributed systems.

- If there is a partition (P), the trade-off is between availability and consistency (A and C).
- Else (E), the trade-off is between **latency** (L) and **consistency** (C).

This theorem acknowledges that even when the system is running normally, there's a tradeoff between latency and consistency.

In conclusion, the CAP Theorem is a powerful tool for understanding the inherent trade-offs in distributed system design. It's not about choosing the "best" property, but rather about making informed decisions based on the specific needs of your application.

By carefully evaluating the CAP trade-offs, you can architect robust and resilient systems that deliver the right balance of consistency, availability, and partition tolerance.

Thank you for reading!

If you found it valuable, hit a like ❤️ and consider subscribing for more such content every week.

If you have any questions or suggestions, leave a comment.

P.S. If you're enjoying this newsletter and want to get even more value, consider

becoming a [paid subscriber](#).

As a paid subscriber, you'll unlock all **premium articles** and gain full access to all [premium courses](#) on [algomaster.io](#).

There are [group discounts](#), [gift options](#), and [referral bonuses](#) available.

Checkout my [Youtube channel](#) for more in-depth content.

Follow me on [LinkedIn](#), [X](#) and [Medium](#) to stay updated.

Checkout my [GitHub repositories](#) for free interview preparation resources.

I hope you have a lovely day!

See you soon,
Ashish

235 Likes · 16 Restacks

Previous

Next

Discussion about this post

Comments Restacks

Write a comment...

Hervé BAKONGO 7 Oct 2024

Liked by Ashish Pratap Singh

Thanks, It is the Best article I have read about CAP THEOREM.

LIKE (2) REPLY SHARE

Ankit Singh 3 May

Liked by Ashish Pratap Singh

This is great, one of the best articles on CAP I have read so far.

LIKE (1) REPLY SHARE

4 more comments...