www.ijecs.in
*International Journal Of Engineering And Computer Science*
*Volume 14 Issue 07 July 2025, Page No. 27488-27491*
*ISSN: 2319-7242 DOI: 10.18535/ijecs/v14i07.5126*

# Approaches to Building Fault-Tolerant Distributed Systems: The CAP Theorem and Its Practical Application

**Smirnov Andrei**

Master's degree, Perm National Research Polytechnic University

**Abstract**

This article examines approaches to building fault-tolerant distributed systems while considering the constraints of the CAP theorem. It explores the impact of trade-offs between consistency, availability, and partition tolerance on architectural decisions and system performance. The study investigates real-world CAP-balancing strategies, including CP, AP, and CA models, as well as hybrid approaches that dynamically adjust consistency and availability levels based on workload characteristics and business requirements. Particular attention is given to the practical application of CAP principles in modern distributed databases, financial systems, and cloud services.

**Keywords:** CAP theorem, fault tolerance, distributed systems, consistency, availability, trade-offs, hybrid strategies, distributed databases.

## 1. Introduction

Modern distributed systems form a building block for information technologies, mainly due to their ability to scale, be fault-tolerant, and have high service availability. However, system architects face a major obstacle known as the CAP theorem that states that data consistency, availability, and partition tolerance cannot coexist. This limitation requires a delicate balance to be struck between workload properties, system design, and business demands.

An effective design paradigm for distributed systems requires a deep understanding of the real-world implications of the CAP theorem, along with an analysis of the trade-offs between its elements. Systems can either prioritize consistency (CP) or availability (AP) or occasionally experience a balance between both. Empirical examples such as distributed databases, financial transaction systems, and cloud computing services show diverse approaches to striking a balance between elements in the CAP theorem and their resultant effects on system performance and reliability.

In this regard, it is vital to explore the key trade-offs between consistency, availability, and partition tolerance while analyzing practical approaches to designing fault-tolerant distributed systems. The goal of this research is to assess the applicability of various CAP balancing strategies in relation to system requirements. The relevance of this topic is underscored by the increasing volume of data and the growing demand for efficient, scalable architectures capable of handling high workloads. The study employs methods of theoretical analysis, comparative architectural review, and case studies of real-world implementations.

## 2. Main part. Trade-offs between consistency, availability, and partition tolerance

Progress in fault-tolerant distributed systems requires a balanced blending of three basic properties: Consistency, Availability, and Partition Tolerance. This principle, outlined in the CAP theorem, identifies inherent limitations in existence within distributed systems and has a significant influence on their design framework. Formulated in 2000 by Eric Brewer and later proved mathematically, the CAP theorem states that a system cannot simultaneously provide all three properties:

- Consistency (C) maintains that all nodes in the system return the same data at any given time. This means that once a write operation is

completed, all subsequent reads from any replica will return the same value.

- Availability (A) guarantees that every request receives a valid response, even if some nodes in the system are unavailable. However, the response may not always reflect the most recent write.
- Partition Tolerance (P) signifies a system's ability to continue operating despite temporary communication failures between its components.

The CAP theorem states that in the presence of a network partition (P), a system must choose between strict consistency (C) and high availability (A). This constraint imposes fundamental architectural decisions on distributed systems, requiring developers to prioritize one of the possible trade-offs based on application-specific requirements [1].

In real-world conditions, networks cannot guarantee absolute reliability. Partitioning is inevitable, especially in globally distributed systems where nodes may be located in different geographic regions and experience temporary communication failures. As a result, partition tolerance must be considered a mandatory requirement, effectively reducing the trade-off to a choice between consistency and availability.

Systems that prioritize consistency ensure strict data update ordering but may become unavailable when network partitions occur. This approach can be commonly used in banking and financial applications, where data integrity is paramount [2]. In contrast, availability-focused systems strive to always respond to requests, even at the cost of temporary data inconsistencies. This strategy is widely employed in social networks, content streaming services, and other applications where minimizing response time is a critical requirement.

The choice between consistency and availability is not only determined by technical considerations but also by business requirements. In the e-commerce sector, high availability is more critical than strict consistency because even momentary downtime can lead to loss of revenue. However, in banking transactions, it is critical that all operations are processed in a strict order, even if this results in temporary delays in service availability [3].

Workload characteristics further impact this decision. In read-heavy systems, weaker consistency models may be acceptable, as minor temporary inconsistencies do not significantly affect functionality [4]. Conversely, in write-intensive environments, stricter consistency mechanisms become essential to ensure data integrity. Understanding the CAP theorem's trade-offs is important for designing fault-tolerant distributed systems. The selection of a balancing strategy depends on application characteristics, business priorities, and infrastructure constraints.

## 3. Strategies for building distributed systems with different cap trade-offs

Designing distributed systems requires selecting a strategy to balance consistency, availability, and partition tolerance. Depending on system priorities, architects adopt different approaches to optimize performance under specific conditions.

Systems with a **high consistency requirement** tend to offer data consistency in the presence of network partitions, but at the expense of availability. This is especially important for financial, distributed ledger, and coordination services, where slight inconsistencies can be catastrophic. These systems typically use consensus algorithms such as Paxos or Raft to guarantee correct data propagation at the expense of adding extreme latency.

A good CP system example is **Google Spanner**, a strongly consistent globally distributed database with TrueTime mechanism ensuring strong consistency [5]. **Zookeeper** is another one, which is used for coordination of the cluster and which is temporarily unavailable when network partition happens to keep data consistent. The same concept is used in most relational databases using distributed transactions, such as PostgreSQL and MySQL with clustering.

The primary drawback of the CP approach lies in potential replication delays and loss of availability during network partitions. If nodes fail to reach consensus on the updated state, the system blocks request processing until connectivity is restored. This makes CP-based systems inefficient for applications that require low-latency responses.

Another class of distributed systems prioritizes **availability and partition tolerance,** ensuring continued operation even in the presence of temporary network failures between nodes. This is achieved by relaxing strict consistency guarantees and adopting eventual consistency strategies. Such systems are widely used in globally distributed data stores, social networks, and streaming services, where it is critical for users to always access at least some version of the data [6].

A prominent example is **Amazon DynamoDB**, which prioritizes availability while allowing temporary inconsistencies between nodes that are resolved asynchronously. A similar approach is used in **Cassandra**, where consistency can be configured but is, by default, secondary to availability. The **Domain Name System (DNS)** also exemplifies an AP system, as data caching across servers can lead to temporary inconsistencies, yet the system guarantees a response to every query.

The primary limitation of AP systems is the need for conflict resolution mechanisms. When different nodes independently update data, discrepancies may arise, requiring either manual intervention or automated reconciliation. This issue is particularly prevalent in distributed NoSQL databases, where different nodes may temporarily store divergent versions of the same data object.

Systems that guarantee **both consistency and availability** can only function under conditions where network partitions do not occur. These architectures are predominantly used in centralized environments, such as traditional relational databases operating within a single data center.

Such systems ensure strict transactional integrity and remain fully available as long as they do not encounter network disruptions. Their main limitation is low fault tolerance, if a network failure occurs, the system becomes entirely unavailable, unlike AP or CP models, which accommodate trade-offs.

A typical example of a CA system is a **centralized PostgreSQL deployment** running within a single data center. It guarantees strong data consistency and high availability under normal conditions, but in the event of a network failure, availability is lost entirely.

In real-world scenarios, strict adherence to a single CAP model is rare. Most distributed systems employ **hybrid strategies** that allow for dynamic trade-offs based on workload characteristics and application requirements.

One example of this approach is **Google Spanner**, which combines elements of CP and CA models: strong consistency is maintained through globally synchronized time, while availability is preserved within local clusters. Similarly, **MongoDB** provides configurable consistency levels, allowing developers to choose between strict consistency and eventual consistency depending on the use case (table I).

**Table I. Impact of consistency on performance [7]**

| Consistency level | Coordination overhead | Typical latency | Common use cases |
|---|---|---|---|
| Strong (Linearizable) | High (consensus required) | Higher | Financial transactions, critical configs. |
| Causal | Medium (dependency tracking) | Moderate | Social media feeds, collaborative apps. |
| Eventual | Low (asynchronous updates) | Lower | Caches, product catalogs, user profiles. |

NewSQL databases such as **CockroachDB** and **TiDB** aim to integrate the benefits of both relational and distributed NoSQL systems. They employ distributed transactions to ensure consistency while allowing for availability adjustments to reduce latency. The choice of strategy depends on business requirements, workload patterns, and system architecture. For some cases, CAP balancing may dynamically adjust in response to network conditions. In certain clusters, consistency may be temporarily relaxed in favor of availability if a network partition is detected.

The CAP theorem has also laid the foundation for the development of frameworks in the field of large language models (LLM), which can be utilized in distributed systems [8]. Researchers propose a similar CAP-like scheme, but with shifted priorities towards context, accuracy, and performance (fig. 1).

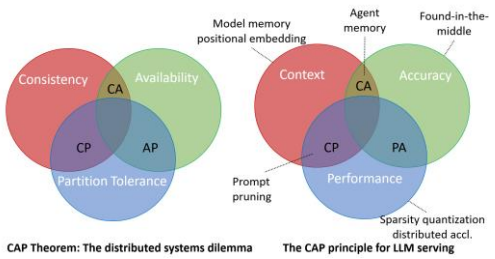


**Figure 1. Comparison of the traditional CAP theorem and the proposed CAP framework for LLM serving**

Modern distributed systems increasingly adopt adaptive CAP balancing methods, allowing priorities to shift based on real-time system conditions. This enables the development of high-performance solutions that cater to a wide range of applications, from financial platforms to globally distributed cloud services.

**Conclusion**

Designing fault-tolerant distributed systems inherently involves trade-offs between consistency, availability, and partition tolerance. The CAP theorem imposes fundamental constraints, requiring system architects to prioritize these factors based on the specific characteristics of the system. In some cases, strong consistency is paramount, as seen in banking systems and distributed transactional databases. In others, high availability takes precedence, particularly in content delivery networks and social media platforms. Hybrid solutions, which integrate elements of different strategies, enable a balanced approach by dynamically adapting system behavior to evolving conditions.

Advancements in technology, including NewSQL databases, distributed clusters, and adaptive consistency mechanisms, have helped mitigate the rigid limitations imposed by the CAP theorem, offering more flexible system architectures. But selection of an optimal strategy among CP, AP, and hybrid types remains a challenge in high-load service design even today. The deep understanding of CAP trade-offs and their implications in real-life scenarios is of prime importance in developing fault-tolerant, scalable, and efficient distributed systems in accordance with the specific demands of business and users.

**References**


1. Lee E.A., Bateni S., Lin S., Lohstroh M., Menard C. Quantifying and generalizing the CAP theorem. arXiv. 2021. arXiv preprint arXiv:2109.07771.
2. Bolgov S. Creating infrastructure for scalable fintech solutions: technical and organizational aspects // ISJ Theoretical & Applied Science. Vol. 140. № 12. P. 354-358.
3. Pandey A.K., Pandey R. Influence of CAP theorem on big data analysis. Int. J. Inform. Technol. (IJIT). 2020. V. 6(6).
4. Malygin D. S. Monitoring of Web Service Availability in Distributed Infocommunication Systems // International Research Journal. 2024. №. 3(141). DOI: 10.23670/IRJ.2024.141.31 EDN: OUGUEQ
5. Google Spanner / Google Cloud // URL: https://cloud.google.com/spanner?hl=en (date of application: 27.04.2025).
6. Khoshaba O., Grechaninov V., Molodetska T., Lopushanskyi A., Zavertailo K. Study of the Workspace Model in Distributed Structures Using CAP Theorem. InInternational scientific-practical conference 2022. P. 229-242. Cham: Springer Nature Switzerland. DOI: 10.1007/978-3-031-30251-0_18
7. Bhosale P. Data Consistency Models in Distributed Systems: CAP Theorem Revisited. IJSAT-International Journal on Science and Technology. 2023. Vol. 14(3). P. 1-11.
8. Zeng P., Ning Z., Zhao J., Cui W., Xu M., Guo L., Chen X., Shan Y. The CAP principle for LLM serving: A survey of long-context large language model serving. arXiv preprint arXiv:2405.11299. 2024.