Proceedings of CONF-SEML 2026 Symposium: Computational Analysis and Modeling in Complex Intelligent Systems
DOI: 10.54254/2755-2721/2026.GU33499

From CAP to CALM: Evolution of Consistency Model in Cloud Native Distributed System

Hao Chang

International School of Technology, Henan University, Zhengzhou, China
2410240534@henu.edu.cn

Abstract. The core of distributed data correctness assurance is consistency trade-off. Aiming at the limitations of traditional models in cloud native environment, this paper discusses the evolution and application of consistency theory. Firstly, this paper reviews the paradigm shift from the consistency, availability, and partition tolerance (CAP) theorem to the consistency as logical monotonicity (CALM) theorem, and analyzes the key significance of "initiative to avoid coordination" in reducing system delay. Secondly, this paper introduces the fine-grained classification framework of "coordinated spectrum" in the cloud native environment in detail, and quantitatively analyzes the multidimensional trade-off between consistency strength and system throughput in the micro service and edge computing scenarios. Finally, a set of optimization paths based on monotonicity design is proposed to reduce synchronization overhead by identifying the logical monotonicity of application logic. The research shows that monotonic design is a breakthrough to improve the efficiency of large-scale distributed systems, and provides theoretical support and practical guidance for state management under cloud native architecture.

Keywords: CAP theorem, Consistency model, Cloud native system, CALM theorem

1. Introduction

Distributed system has become the core infrastructure in cloud computing, Internet of things, big data and other fields. Its core feature is that the data is distributed on multiple physically isolated autonomous nodes. However, limited by unreliable network latency, node failure and highly concurrent cross regional access, how to ensure the consistency of multi replica data has always been one of the most serious challenges in the distributed field [1,2].

Back to the theoretical starting point, Eric Brewer's CAP conjecture in 2000 and its subsequent formal proof delineated the physical boundary for the distributed system, that is, when the network partition occurs, the system must make a trade-off between Consistency, Availability and Partition Tolerance [3]. Although CAP theorem has laid the foundation of distributed design, its limitations have become increasingly prominent. Stonebraker and other scholars pointed out that network partitions are not frequently faulted in modern high reliable data centers, and overemphasizing partition fault tolerance may lead to the system sacrificing consistency under normal conditions [4]. Subsequently, researchers tried to find optimization space under the boundary of CAP, such as improving fault tolerance while ensuring strong consistency through consensus algorithms such as

DOI: 10.54254/2755-2721/2026.GU33499

Paxos and Raft, or seeking a breakthrough in the field of final consistency by using vector clock and conflict free replication data types (CRDTS), so as to meet the low latency requirements under massive concurrency [5,6].

In the era of cloud nativity, the boundaries of distributed systems are further expanded. Microservice architecture transforms single database transaction into complex cross service distributed transaction. The traditional two-phase commit (2PC) protocol is weak in scalability. At the same time, Serverless architecture requires high flexibility in state management, while edge computing puts forward strict requirements for local autonomy in weak network environment. In response to these challenges, the Consistency As Logical Monotonicity (CALM) theorem proposed by Joe Hellerstein reveals a new path [6]. The theorem points out that if the program logic is monotonous, consistency can be achieved without coordination, which marks the paradigm shift of consistency research from "passive trade-off" to "active avoidance of coordination" [7].

This paper aims to sort out the evolution from CAP theorem to CALM theorem through systematic literature review. Therefore, the following chapters are arranged as follows. The second chapter analyzes CAP and its derivative theory PACELC. Chapter three focuses on the coherence model spectrum. The fourth chapter analyzes the new challenges in the cloud native environment. The fifth chapter discusses the theoretical value and evolution of CALM theorem.

2. Theoretical basis: CAP theorem and its evolution

2.1. Structure proposal of CAP theorem

In 2000, Brewer proposed the CAP conjecture, pointing out that the distributed system cannot perfectly meet the consistency, availability and partition fault tolerance at the same time [3]. Consistency refers to that all nodes see the same data at the same time, availability refers to that non fault nodes return reasonable responses within a reasonable time, and partition fault tolerance refers to that the system can continue to operate when the network partition occurs. In 2002, Gilbert and Lynch proved in the asynchronous network model that when the network partition is unavoidable, the system can only meet one of the consistency and availability at most [5].

2.2. Controversy and criticism of CAP

Stonebraker pointed out that CAP is often abused. Many fault types, such as application errors and repeatable DBMS errors, do not belong to the CAP framework. Network partitioning in LAN environment is extremely rare. It is questionable to sacrifice consistency for rare scenarios. Subsequent studies clarified that CAP is not two out of three, but one out of two between C and A when partitioning [5]. Giving up consistency does not mean data confusion, and the financial system is not all strongly consistent.

2.3. Extension of CAP

The limitation of CAP is that it only discusses the trade-offs when partitioning, and ignores the trade-offs when the system is running normally. Therefore, Abadi et al. proposed the PACELC theorem, that is, when partitioning, there is a trade-off between availability and consistency, otherwise there is a trade-off between latency and consistency. Synchronous replication ensures strong consistency but increases latency, while asynchronous replication reduces latency but may

DOI: 10.54254/2755-2721/2026.GU33499

return old data. This extension upgrades the consistency discussion from binary selection in partitioning to multi-dimensional trade-offs in the whole time period [5].

3. Consistency model spectrum

3.1. Strong consistency and order constraint model

As shown in Figure 1, the consistency model defines the rigor of synchronization between data replicas in a distributed system, forming a continuous spectrum between performance and correctness. At the top of the spectrum is the strong consistency model represented by Linearizability. Linear consistency requires that all operations be atomized at an instant and must strictly follow the global real-time order, which means that once a process reads a new value, subsequent read operations of all processes must return the value. This model is usually implemented through consensus protocols such as Paxos or Raft. Although it provides developers with an intuitive experience similar to a single machine, its high coordination cost makes it mainly limited to key scenarios with high requirements for fault tolerance, such as financial transactions and distributed locks. At the same time, although sequence consistency also requires all processes to see the consistent operation execution sequence, it relaxes the hard constraint on the global real-time time and only needs to meet the sequence in the program logic [2]. However, in large-scale clusters across regions, the synchronization delay caused by maintaining the global order is still the main obstacle to system scalability.

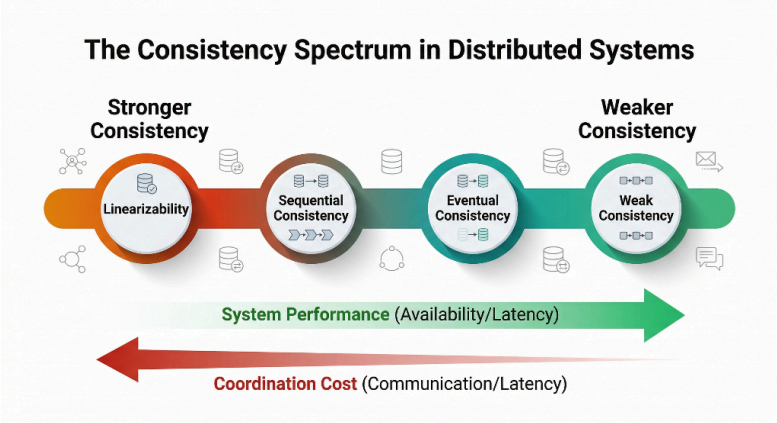

Figure 1. Tradeoff relationship of distributed consistency model spectrum (picture credit: original)

3.2. Weakening trade-offs and final consistency evolution

In order to obtain higher response speed in a distributed environment, researchers have proposed a variety of weakening constraint models at the middle and low end of the spectrum. Causal Consistency is based on the "first occurrence" relationship of Lamport. It only maintains the order of operations with causal correlation, and allows concurrent operations to be executed in different orders on different replicas, which is of great value in geographic replication systems. On this basis, the client center consistency is further refined to ensure the data consistency of a single user when switching access nodes through the semantics of monotonous reading, monotonous writing, "read what you write" and "write follow read". The final consistency at the end of the spectrum is the core of BASE theory, which only ensures that all copies converge uniformly after a long enough time without updates. In order to shorten inconsistent windows and solve conflicts, modern systems often

DOI: 10.54254/2755-2721/2026.GU33499

introduce conflict free replication data types (CRDTS) to realize automatic merging by using the semi lattice property in mathematics [6]. This model evolution from strong to weak enables the system designer to select the most appropriate balance point in the consistency spectrum according to the business tolerance for delay.

4. New challenges in cloud native environment

4.1. Core features of cloud native architecture

Cloud native is characterized by containers, microservices, and declarative APIs. Containerization realizes environment consistency through Docker, and Kubernetes provides elastic scalability and self-healing CAP abilities. Microservice architecture splits the monomer into small and autonomous services, but at the same time, cross service distributed transactions become a new pain point. The service grid uses sidecar mode to manage services and sink infrastructure to solve the problem of communication reliability [8].

4.2. Multi cloud and mixed cloud environment

The adoption rate of Multi Cloud Architecture in current enterprises continues to rise, aiming to avoid supplier lock-in and optimize costs. Alonso et al. Classified multi cloud applications into replication and distribution. The latter requires components to coordinate across the management domains and network boundaries of different cloud vendors, resulting in an exponential increase in the complexity of consistency implementation [1]. In addition, the heterogeneity of different cloud platforms in infrastructure such as network latency and storage semantics makes the global state synchronization face severe challenges. The high bandwidth cost and unstable delay fluctuation caused by cross cloud data transmission further force the system to make a more rigorous trade-off between consistency strength and operation cost.

4.3. New challenges of cloud native to consistency

Micro service splitting makes cross service distributed transactions difficult. Saga mode achieves final consistency through local transactions and compensation, but the complexity is much higher than ACID [5]. Serverless places the state outside, and the cold start of the function may lead to the loss of state. It is difficult to implement cross function distributed transactions [9]. Edge computing will sink the computing power, requiring node autonomy, but autonomy also brings the risk of data bifurcation [10].

4.4. Toughness design mode and consistency

Distributed tracking helps locate performance bottlenecks, fuse mode prevents fault diffusion through rapid failure, and chaos engineering actively injects the toughness of fault test system. However, there are complex conflicts between these modes and consistency assurance, and incomplete transactions may be left when the fuse is triggered, resulting in data inconsistency [11].

DOI: 10.54254/2755-2721/2026.GU33499

5. From CAP to CALM: theoretical evolution and paradigm shift

5.1. Proposal of CALM conjecture

Hellerstein put forward the CALM conjecture in 2010. The core insight is that consistency and logical monotony are intrinsically linked. The output of monotonic programs will not decrease when the input increases, and such programs can achieve consistency without coordination [6]. Bloom language integrates CALM principles into the design to help programmers identify the parts that need to be coordinated.

5.2. Proof of CALM theorem

Amelot et al. Introduced the relational transducer network model in 2013, and formally proved that deterministic queries have a consistent and coordinated distributed implementation if and only if the query is monotonous. This is the first strict proof of CALM conjecture, which sublimates it from engineering intuition to formal theorem.

5.3. Finer grained monotonicity classification

In 2015, Amelot et al. Further introduced fine-grained monotonicity classification [6]. Monotonicity corresponds to the situation that the node has only basic network knowledge, semi monotonicity corresponds to the situation that the node perceives the data partitioning strategy, and disjoint monotonicity corresponds to the scene of domain guided partitioning. Different system configurations correspond to different computable problem boundaries.

5.4. Generalized CALM theorem

Baccaert and Ketsman extended the CALM theorem to non deterministic computing in 2026, introduced the concept of distributed behavior and configuration constraint framework, and proved the generalized CALM theorem. They proposed the coordination spectrum [7] and classified the problems according to the required coordination degree, from the monotonous behavior of nodes without coordination to all computable behaviors requiring strong coordination, forming a continuous spectrum.

5.5. Summary of evolution from CAP to CALM

This evolution reflects the deepening of the theory from "passive trade-off" to "active avoidance". As shown in Table 1, CAP delimits the physical boundary of zoning, PACELC complements the delay trade-off under normal conditions, and CALM theorem reveals the feasible path to avoid coordination cost through logical monotonicity identification, and promotes the design perspective from binary opposition to multidimensional coordination spectrum.

DOI: 10.54254/2755-2721/2026.GU33499

Table 1. Comparison of evolution of distributed consistency theory

Theoretical model	Trade off dimension	Core insights	Design paradigm
CAP	C vs A (partitioning)	Partition time consistency and availability cannot be combined	Passive trade-offs
PACELC	L vs C (normal conditions)	Strong consistency also has delay cost under normal conditions	Overall trade-offs
CALM	Coordination vs Monotonicity	Monotonic logic can reach agreement without coordination	Active avoidance

6. Conclusion

This paper reviews the theoretical evolution from CAP to CALM, analyzes the new challenges of cloud native environment to consistency, and introduces coordinated spectrum as a fine-grained classification framework. The CAP theorem lays the theoretical foundation, but the core is trade-offs. The consistency model spectrum provides a complete picture. The cloud native architecture not only improves elasticity, but also puts new pressure on consistency. The CALM theorem reveals the internal relationship between monotonicity and consistency without coordination.

In terms of practical enlightenment, cloud native application designers should choose a consistency model according to the importance of data, and give priority to monotonic design to minimize coordination requirements. Developers can separate the parts that need coordination from those that do not need coordination by identifying the monotonous parts.

From CAP to CALM, consistency research has gone through more than 20 years. CALM theorem shows that it can surpass trade-offs under certain conditions. Understanding this evolution is the basis for system designers to make wise decisions in a complex distributed world.

References

[1] Alonso, J., Orue-Echevarria, L., Casola, V., Torre, A. I., Huarte, M., Osaba, E., & Lobo, J. L. (2023). Understanding the challenges and novel architectural models of multi-cloud native applications–a systematic literature review. Journal of Cloud Computing, 12(1), 6.

[2] Aldin, H. N. S., Deldari, H., Moattar, M. H., & Ghods, M. R. (2019). Consistency models in distributed systems: A survey on definitions, disciplines, challenges and applications. arXiv preprint arXiv: 1902.03305.

[3] Brewer, E. (2010, July). A certain freedom: thoughts on the CAP theorem. In Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing (pp. 335-335).

[4] Stonebraker, M. (2010). Errors in database systems, eventual consistency, and the CAP theorem. Communications of the ACM, BLOG@ ACM.

[5] Muñoz-Escoí, F. D., de Juan-Marín, R., García-Escrivá, J. R., González de Mendívil, J. R., & Bernabéu-Aubán, J. M. (2019). CAP theorem: revision of its related consistency models. The Computer Journal, 62(6), 943-960.

[6] Laddad, S., Power, C., Milano, M., Cheung, A., Crooks, N., & Hellerstein, J. M. (2022). Keep CALM and CRDT on. arXiv preprint arXiv: 2210.12605.

[7] Baccaert, T., & Ketsman, B. (2026). A generalized CALM theorem for non-deterministic computation in asynchronous distributed systems. Information Systems, 102691.

[8] Oyeniran, O. C., Modupe, O. T., Otitoola, A. A., Abiona, O. O., Adewusi, A. O., & Oladapo, O. J. (2024). A comprehensive review of leveraging cloud-native technologies for scalability and resilience in software development. International Journal of Science and Research Archive, 11(2), 330-337.

[9] Besozzi, V., Della Bartola, M., Dazzi, P., & Danelutto, M. (2025). High-Performance Serverless Computing: A Systematic Literature Review on Serverless for HPC, AI, and Big Data. IEEE Access, 13, 195611-195656.

DOI: 10.54254/2755-2721/2026.GU33499

[10] Zhao, J., Quan, H., Xia, M., & Wang, D. (2023). Adaptive resource allocation for mobile edge computing in internet of vehicles: A deep reinforcement learning approach. IEEE Transactions on Vehicular Technology, 73(4), 5834-5848.
[11] Arif, T., Jo, B., & Park, J. H. (2025). A comprehensive survey of privacy-enhancing and trust-centric cloud-native security techniques against cyber threats. Sensors, 25(8), 2350.