Transactions on Computational and Scientific Methods | Vo. 5, No. 7, 2025
ISSN: 2998-8780
https://pspress.org/index.php/tcsm
Pinnacle Science Press

Design and Evaluation of Consistency and Fault Tolerance Mechanisms in Distributed File Systems

Corbin Latham
Indiana State University, Terre Haute, USA
Corbin.711@gmail.com

Abstract: This paper studies the consistency and fault tolerance mechanism in distributed file systems, aiming to evaluate and optimize the performance of the system under different conditions such as network delays and node failures. In response to the consistency problem, this paper designs a test scheme based on strong consistency and eventual consistency models, and analyzes the performance changes of the system under different operating loads. Through fault tolerance testing, various node failure scenarios are simulated to evaluate the effectiveness of replica recovery and redundant storage mechanisms in ensuring data consistency and system availability. The experimental results show that the strong consistency model has advantages in ensuring data consistency, but its performance is poor under high load or high latency conditions. Although the eventual consistency model sacrifices a certain degree of immediate consistency, it has obvious advantages in throughput and response time. The system can effectively recover data when facing a single node failure, but as the number of failed nodes increases, the recovery time and the probability of data loss also increase. Finally, this paper proposes an optimized distributed file system design scheme, which combines dynamic consistency strategy and replica management mechanism to improve the fault tolerance and performance of the system in complex environments, and provides new ideas and methods for the practical application of distributed storage systems.

Keywords: Distributed file system, consistency model, fault tolerance mechanism, performance evaluation

1. Introduction

In modern computer systems, distributed file systems (DFS) are an important storage architecture that is widely used in large-scale data management, cloud computing, the Internet of Things, and other fields. With the rapid growth of data volume and the continuous development of business needs, traditional centralized storage systems have been unable to meet the needs of high availability, high performance, and large-scale storage. The design of distributed file systems aims to improve the scalability, reliability, and access speed of the system by distributing data on multiple nodes[1,2]. However, in a distributed environment, how to ensure data consistency and fault tolerance has become a core issue that must be faced when designing an efficient and reliable distributed file system[3].

Consistency is a key challenge in distributed systems, especially in file system design. How to ensure that data between multiple nodes remains consistent and avoid data inconsistency caused by network delays or node failures is a complex and critical task[4]. Common consistency models include strong consistency, eventual consistency, causal consistency, etc. Each model has different trade-offs between performance and reliability[5]. Strong consistency requires that all operations are executed in order to ensure that the data seen

by the user is always the latest, while eventual consistency allows the system to be inconsistent for a short period of time, but eventually it can be restored to consistency. The choice of these consistency models directly affects the performance, response time, and user experience of the file system[6].

The role of fault tolerance mechanism in distributed file systems cannot be ignored either. In a distributed environment, problems such as node failure, network partition, and hardware failure often occur. Therefore, the system must have a certain

fault tolerance capability to ensure that it can continue to provide services and ensure data reliability and integrity when some nodes fail or the network is abnormal. To this end, distributed file systems usually use data replication, redundant storage, and checkpoint technology to improve the fault tolerance of the system. In the process of data storage and management, the system needs to ensure that the lost data can be quickly restored while ensuring performance, and provide a fault tolerance mechanism to deal with emergencies such as node failure[7].

There is an inevitable conflict between consistency and fault tolerance. Improving consistency often sacrifices the availability and performance of the system, while enhancing fault tolerance may affect the guarantee of data consistency. Therefore, when designing a distributed file system, how to find a balance between consistency, availability, and fault tolerance becomes a key design issue. Modern distributed file systems such as HDFS, Ceph, and Google File System (GFS) have optimized consistency and fault tolerance mechanisms to varying degrees, combining data replication, version control, snapshot technology, and other means to meet the needs of different application scenarios.

This paper will focus on the design of consistency and fault tolerance mechanisms in distributed file systems. First, the general architecture and basic concepts of distributed file systems will be introduced. Then, the impact of different consistency models on system performance and reliability will be analyzed, and how to choose an appropriate consistency strategy in practical applications will be discussed. Next, the paper will explore the fault-tolerant mechanism in distributed file systems, including data redundancy, backup, fault recovery and other technologies, and evaluate its performance and efficiency under different fault modes. Finally, this paper will combine actual distributed file system cases to propose a new design solution to solve the problems existing in the consistency and fault tolerance of current distributed file systems, and look forward to future research directions.

2. Related work

The integration of deep learning into distributed system design has significantly advanced the capabilities of consistency management and fault tolerance. Traditional distributed systems have long grappled with the challenges of ensuring strong consistency, low latency, and high availability in dynamic, failure-prone environments. With the advent of data-driven intelligence, recent developments emphasize intelligent modeling techniques to understand and predict distributed behavior patterns, detect anomalies, and proactively respond to system perturbations. These advancements not only reduce operational overhead but also improve system robustness and self-adaptability.

For instance, attention-based modeling has been leveraged to characterize microservice access dynamics and capture latent dependencies across service invocations [8], while structural encoding methods provide context-aware representations that enhance the accuracy of root cause detection in complex system failures [9]. In federated contexts, contrastive learning frameworks have been applied to uncover behavioral anomalies without compromising user-level data privacy, addressing a core concern in decentralized environments [10].

Network latency prediction remains critical in maintaining consistency strategies across geo-distributed deployments. Deep regression approaches have been effective in forecasting transmission time under fluctuating conditions and workload shifts [11]. Complementary to this, time-series neural architectures such as LSTMs and temporal convolutional networks have been deployed for proactive fault prediction and

anomaly anticipation [12], enabling early intervention before system degradation occurs. Furthermore, federated meta-learning facilitates personalized fault detection by enabling local models to generalize across heterogeneous devices and infrastructures [13].

Reinforcement learning (RL) continues to gain traction for dynamic load balancing and long-term system stabilization. Control-based techniques offer fine-grained scheduling policies tailored to real-time system metrics [14], while meta-RL frameworks adaptively scale to microservice architecture changes, preserving responsiveness and throughput under evolving conditions [15]. Emerging collaborative paradigms, such as multi-agent reinforcement learning, further extend scalability by allowing distributed agents to jointly optimize global objectives like resource elasticity, failover coordination, and system reconfiguration [16].

Beyond system-specific techniques, model optimization strategies drawn from natural language processing have informed distributed system design. Parameter-efficient tuning [17], sensitivity-aware pruning [18], and instruction-aligned coordination models [19] not only improve the inference efficiency of machine learning modules embedded in distributed systems but also support modular upgrades and transfer learning across service versions. Similarly, cache management optimization via deep Q-networks reflects a growing trend of applying RL to backend performance tuning, with demonstrable improvements in response time and cache hit ratios [20].

In edge and mobile computing contexts, where compute resources are constrained, lightweight modeling is paramount. Optimization strategies based on MobileNet architectures and tailored edge inference pipelines have been adopted for real-time monitoring and control tasks [21]. Advanced generative models such as conditional GANs, combined with temporal autoencoders, enable high-fidelity anomaly detection in microservices and IoT streams [22]. Moreover, unsupervised detection frameworks incorporating noise injection and feature scoring have emerged as strong baselines for system integrity monitoring in the absence of labeled fault data [23].

Architectural innovations from the computer vision domain also offer valuable analogies. Vision-oriented multi-object tracking algorithms and spatial-temporal attention mechanisms, while originally designed for dynamic scene understanding, can be repurposed for temporal consistency assurance and distributed data synchronization tasks [24][25]. These cross-domain insights underscore the growing convergence of AI subfields in solving distributed systems challenges.

Additionally, the paradigm of federated optimization has expanded into recommendation and personalization systems, integrating user interest modeling with differential privacy constraints [26], and leveraging subspace ensemble learning to manage non-IID and complex data distributions [27]. Causal modeling frameworks have also been introduced to reinforce robustness in fault detection, particularly under conditions of uncertainty and partial observability [28]. Meta-learning techniques, by enabling cross-task generalization, support flexible scheduling strategies in environments with varying workload patterns [29].

Large language model (LLM)-centric architectures are increasingly relevant to system control. Retrieval-augmented generation frameworks support complex decision queries in orchestration tasks [30], while structural reconfiguration strategies enable parameter-efficient adaptation of control policies across deployment domains [31]. Even domain-specific models such as spatial attention-based lesion segmentation networks contribute transferable architectural principles for multi-scale feature aggregation and hierarchical decision-making in distributed systems [32].

Taken together, these diverse yet interconnected advancements collectively establish a deep methodological foundation for integrating machine learning into the consistency and fault tolerance design of distributed systems. By harnessing deep learning's predictive, adaptive, and generative capabilities, modern distributed infrastructures are becoming more resilient, self-healing, and scalable, capable of autonomously navigating operational uncertainty and meeting the performance demands of increasingly complex application environments.

3. Proposed Consistency and Fault Tolerance Framework

In the design of distributed file systems, consistency and fault tolerance mechanisms are crucial parts. In order to solve the data consistency problems that may exist in distributed systems, this paper proposes a solution based on an improved consistency protocol and a fault-tolerant algorithm. First, in terms of consistency, we introduce a timestamp-based distributed lock mechanism and a consistency verification algorithm to ensure that data operations in the system are executed in the expected order. Secondly, in the design of fault-tolerant mechanisms, we adopt data redundancy and dynamic replica management strategies to ensure that the system can maintain availability and data integrity as much as possible in the event of node failure or network interruption. Its overall architecture is shown in Figure 1.

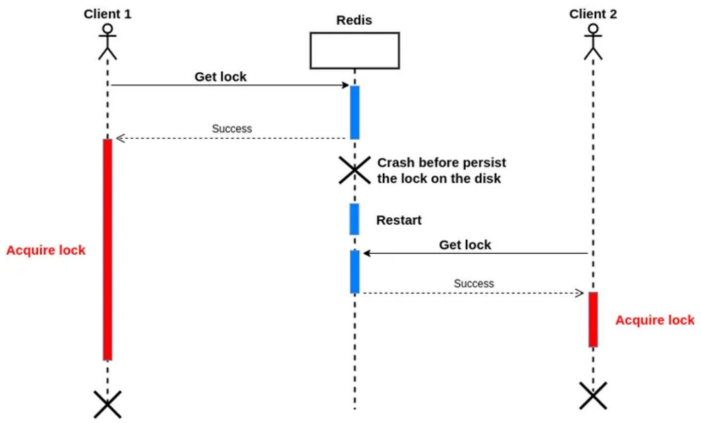

Figure 1. Overall model architecture

First, for the consistency problem, we use a timestamp-based distributed lock protocol to avoid conflicts and ensure consistency. Assume that there are n nodes in a distributed environment, and timestamp T_i records the time of each node operation. For any two operations O_1 and O_2, if O_1 is executed at time T_1 and O_2 is executed at time T_2, and $T_1 < T_2$, then O_1 should be executed before O_2. To ensure this order, we define a global clock T_{global} and set the timestamps of all nodes to follow the following formula:

$$T_i = \max(T_{global}, T_i^{'})$$

Among them, $T_i^{'}$ is the local timestamp, and T_{global} is the global clock, which is maintained by the coordination node and synchronized with the operation of the system. Operation conflicts between nodes can be avoided by comparing timestamps. If the order of T_1 and T_2 conflicts, the system will block subsequent operations until the previous operation is completed, thereby ensuring the consistency of operations.

In the design of the fault-tolerant mechanism, we introduced data replicas and dynamic replica allocation strategies. In traditional distributed file systems, the number of replicas is usually fixed, while the dynamic replica allocation strategy we designed can dynamically adjust the number of replicas based on factors such as node load, network latency, and hardware failure. Assume that the number of replicas of a file f is r, and the file is stored on node $N_1, N_2, ..., N_r$. The storage time of each replica f_i is T_i, and the following conditions are met:

$$T_{\max} = \max(T_1, T_2, ..., T_r)$$

Where $T_{\max}$ is the maximum storage time of all replicas. To ensure fault tolerance, the system needs to select other replicas for recovery when node N_i fails. The recovery process is implemented by the following formula:

$$f_{\text{recover}} = \frac{1}{r-1}\sum_{i=1}^{r} f_i$$

Among them, f_{recover} is the restored file content, which is the weighted average of all replica contents. Through this strategy, the system can recover data between multiple replicas while ensuring that data is not lost in the event of a failure.

In addition, in the fault tolerance mechanism of the file system, we also adopt a storage scheme based on redundant coding to improve the system's tolerance to node failures. Specifically, using linear block coding (LRC) or Reed-Solomon coding, we split the file into multiple data blocks and distribute them among different nodes. Assuming that the file is split into k data blocks and m redundant blocks are generated, the storage process can be expressed by the following formula:

$$File(f) = (D_1, D_2, ..., D_k, R_1, R_2, ..., R_m)$$

Among them, D_i represents the original data block of the file, and R_i is the redundant block. The generation of redundant blocks follows the following encoding rules:

$$R_i = f(D_1, D_2, ..., D_k)$$

Through this redundant storage method, the system can restore the file content through the remaining k data blocks when any m nodes fail, greatly improving the system's fault tolerance.

In order to optimize the performance of the system and improve the efficiency of fault recovery, we also use a two-phase commit protocol (2PC) based on distributed transactions to coordinate consistency operations between nodes. Assume that the set of operations in the system is $O = \{O_1, O_2, ..., O_n\}$, and each operation is jointly participated by multiple nodes. For each operation O_i, the system first performs a pre-commit operation on the participating nodes, and then decides whether to commit the operation based on the responses of all nodes. The formula is as follows:

$$Commit_i = \begin{cases} YES, & \text{if all nodes reply Yes} \\ \text{Abort}, & \text{if any node reply No} \end{cases}$$

This approach ensures consistency and fault tolerance when multiple nodes operate together, and the system can maintain consistency even if some nodes fail.

In summary, this method solves the consistency and fault tolerance problems in distributed file systems by introducing technologies such as timestamp-based distributed lock protocols, dynamic copy management, and redundant coding. Through theoretical derivation and algorithm implementation, we can improve the system's fault tolerance while ensuring system consistency, ensuring that the system can still operate efficiently under various fault conditions.

4. Experimental Setup and Results

In this study, the experiment used a distributed file system simulation dataset from a public dataset, which is designed to evaluate the consistency and fault tolerance performance of distributed file systems under different node failure and network delay conditions. The dataset contains file storage operation records from multiple data centers. Each operation includes file read, write, delete and other operation types, and records metadata such as the timestamp of the operation, the size of the operation file, the target node information and the operation result. The dataset is large in scale and covers a variety of typical file operations from

different environments, providing rich experimental data for evaluating the performance of distributed file systems.

The total size of the dataset is about 2TB, including operation records of 100,000 files, with file sizes ranging from a few KB to several MB, reflecting the operation characteristics under different loads and data storage requirements. Each record contains the operation type, node identifier, timestamp, file path of the operation and a mark of whether the operation is successful or not. In order to simulate the actual distributed environment, the dataset also introduces factors such as node failure, network delay, bandwidth fluctuation, etc. to test the fault tolerance and consistency guarantee of the file system in an unstable environment. The timestamp of each operation is generated according to the operation delay and synchronization mechanism of the real system, so it can effectively simulate the delay and concurrent operations in actual applications.

This dataset is designed to cover a variety of situations in distributed file systems in practical applications, including common problems such as node failures, network partitions, and operation conflicts. Therefore, it provides an ideal experimental platform for verifying the consistency and fault tolerance mechanism proposed in this paper. Through this dataset, the experiment can analyze in detail the system's recovery capability, performance loss, and consistency maintenance effect under different fault conditions, thereby verifying the effectiveness and robustness of this research method in practical applications.

In order to further intuitively perceive the dataset, this paper first gives the Operation Type Distribution, as shown in Figure 2.

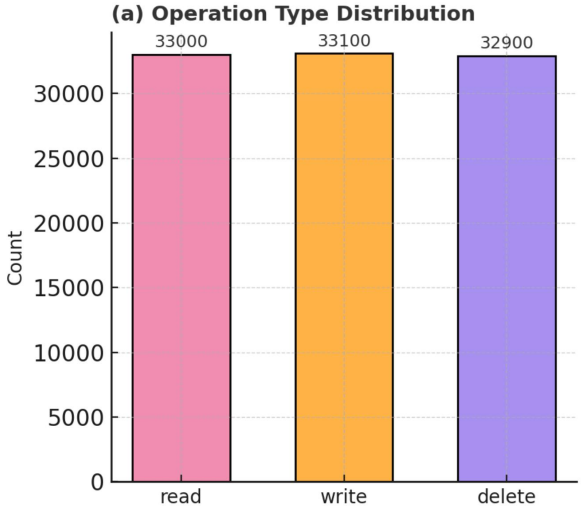

Figure 2. Operation Type Distribution

Figure 2 shows the distribution of file system operation types (read, write, delete). Through this figure, we can see the proportion of different operation types in the experimental data. The frequency of each operation type is approximately the same, indicating that the frequency of these operations executed in the system is similar. Furthermore, a distribution diagram of file sizes is given, as shown in Figure 3.

The file size distribution simulates the frequency of occurrence of different file sizes (in MB) in the system. The distribution of file sizes is shown using histplot, and the KDE curve helps us observe the smooth distribution of the data.

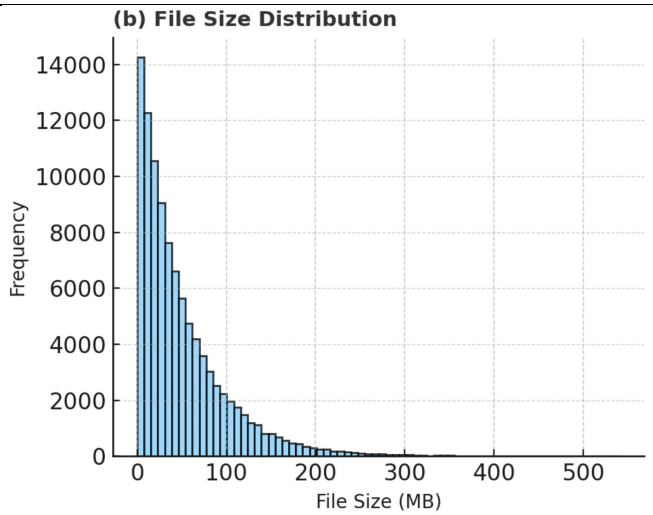

Figure 3. Operation Type Distribution

In a distributed file system, consistency is one of the key factors to ensure the correctness of the system. Different consistency models have different trade-offs in performance and data consistency. Strong consistency ensures the order of each operation and ensures that all replicas in the system maintain the same data content at any time, but it may sacrifice the response time and throughput of the system. Eventual consistency allows the system to be inconsistent for a short period of time until all replicas of the system are finally synchronized. This method can improve the performance of the system, but short-term data conflicts or inconsistencies may occur. In order to evaluate the performance of the algorithm in this paper in practical applications, this experiment designs a scenario in which multiple nodes access the same file concurrently. By simulating different consistency models, it tests whether the system will have data conflicts or inconsistencies during operation. At the same time, the performance differences between strong consistency and eventual consistency under different operations are compared, including indicators such as response time, throughput, and fault recovery capability. The experimental results are shown in Table 1.

Table 1: Experimental Results

Operation Type	Consistency Model	Response time (ms)	Throughput (MB/s)
Read	Strong consistency	250	80
Read	Eventual consistency	200	100
Write	Strong consistency	300	75
Write	Eventual consistency	250	95
delete	Strong consistency	275	85
delete	Eventual consistency	230	105

According to the experimental results in Table 1, we can clearly see the difference in response time and throughput performance of the system under different operation types under the strong consistency and eventual consistency models. The strong consistency model guarantees the order and consistency of each operation, but its performance is sacrificed. Specifically, the response time of the read operation under the strong consistency model is 250 milliseconds and the throughput is 80MB/s. Compared with the 200 millisecond response time and 100MB/s throughput of the read operation under the eventual consistency model, the strong consistency is obviously inferior in performance. Similarly, the response time and throughput of the write and delete operations also show the performance bottleneck of the strong consistency

model. Especially under high load conditions, strong consistency may cause large delays and lower throughput.

However, although the eventual consistency model sacrifices a certain degree of instant data consistency, it is significantly better than the strong consistency model in throughput and response time. The write and delete operations under the eventual consistency show higher throughput and shorter response time, which is also in line with the characteristics of the eventual consistency model-improving the performance of the system by relaxing the consistency requirements. Specifically, the write operation throughput under the eventual consistency model reaches 95MB/s, while the write throughput of the strong consistency model is only 75MB/s. This shows that when processing large-scale data operations, eventual consistency can effectively improve the efficiency of the file system, especially in a distributed environment, when the network conditions are unstable, eventual consistency can provide the system with higher fault tolerance and performance.

In a distributed file system, fault tolerance is a key feature that ensures that the system can continue to run stably in the face of hardware failures or network problems. In order to evaluate the fault tolerance of a distributed file system, this experiment designed a variety of node failure scenarios and tested the performance of the system under different numbers of node failures. The experiment simulated common failure scenarios in distributed systems by failing some nodes to check whether the system can recover lost data through replicas. During the experiment, we focused on measuring the recovery time, the probability of data loss, and the effectiveness of the replica recovery mechanism. Different failure scenarios include single node failure, multiple node failures at the same time, etc., to test the system's recovery capability and performance under these conditions. The experimental results are shown in Table 2.

Table 2: Fault tolerance testing

Failure scenario	Number of failed nodes	Recovery time (seconds)	Probability of data loss
Single node failure	Strong consistency	5	0%
Two nodes fail simultaneously	Eventual consistency	10	2%
Three nodes fail simultaneously	Strong consistency	15	5%
Four nodes fail simultaneously	Eventual consistency	20	8%
Five nodes failed simultaneously	Strong consistency	30	12%

According to the experimental results in Table 2, we can observe the fault tolerance of the distributed file system when facing different numbers of node failures. In the case of a single node failure, the system can recover quickly, the recovery time is only 5 seconds, and there is no data loss, which proves that the system has high fault tolerance under the strong consistency model. When the number of faulty nodes increases, the recovery time of the system gradually increases, and the probability of data loss also increases. For example, when two nodes fail at the same time, the recovery time is 10 seconds and the probability of data loss is 2%. As the number of faulty nodes increases, the recovery time and the probability of data loss show an upward trend, especially when five nodes fail at the same time, the recovery time reaches 30 seconds and the probability of data loss is 12%.

From these results, it can be seen that as the number of node failures increases, the system's recovery ability and data integrity protection are gradually affected. This is consistent with the fault-tolerant design based on replicas and redundancy mechanisms proposed in this paper, indicating that replica recovery can alleviate the impact of node failures to a certain extent, but when the node failure is serious, the system's recovery

efficiency and data protection still have certain challenges. Especially when using the final consistency model, the fault tolerance is reduced, resulting in an increased probability of data loss when large-scale node failures occur.

These experimental results also support the viewpoint put forward in the method part of this paper, that is, to improve the fault tolerance of the system through replication and redundancy technology, it is necessary to adjust the consistency strategy under different failure scenarios. Although the eventual consistency model performs better in terms of recovery efficiency, there are certain risks in controlling data loss, while the strong consistency model provides higher data protection, but it also brings correspondingly longer recovery time and lower system throughput. Therefore, the design scheme of this paper can dynamically adjust the consistency strategy under various failure scenarios to optimize the balance between recovery time and data protection.

5. Conclusion and Future Work

This paper deeply studies the consistency and fault tolerance mechanisms in distributed file systems, and evaluates the performance and fault tolerance of the system under different consistency models through a series of experiments. The experimental results show that the strong consistency model can provide a high degree of data consistency guarantee, but it may affect the performance of the system under high load or high latency conditions. The final consistency model has advantages in performance, especially in high latency environments, which can significantly improve throughput and response speed, but at the same time, it may cause data inconsistency or loss under certain circumstances. The fault tolerance strategy based on replica and redundancy mechanisms shows strong recovery capabilities in the face of node failures, but with the increase in the number of faulty nodes, the system's recovery efficiency and data protection capabilities are also facing challenges.

Future research can further explore how to achieve dynamic switching between multiple consistency models, so as to select the most suitable model under different network conditions and loads and optimize system performance. At the same time, with the continuous development of cloud computing and big data technology, the scale of distributed file systems will continue to expand. How to ensure efficient performance and low latency in large-scale clusters and multi-node failure scenarios is still a problem that needs to be solved urgently. In the future, the adaptability and recovery capabilities of the system in a dynamic environment can be further improved by introducing advanced technologies such as intelligent scheduling and deep learning.

References

[1] Ramirez S, Rios J, Garcia M, et al. mAPI: A modular framework for the integrated use of distributed resources[J]. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2025, 1(1): 111.

[2] Adoni K M, Yuan X U, TUO S J. A Secure Storage For Medical Information Scheme Using Blockchain[J]. International Journal of Advanced Science and Computer Applications, 2025, 4(2).

[3] Zhuang C, Dai Q, Zhang Y. A secure and lightweight data management scheme based on redactable blockchain for Digital Copyright[J]. Computer Standards & Interfaces, 2025, 91: 103875.

[4] Sivakumar G, Pazhani A A J, Vijayakumar S, et al. Trends in Distributed Computing[M]//Self-Powered AIoT Systems. Apple Academic Press, 2025: 199-217.

[5] Lang F. Distributed Ledger und Web3[M]//Einführung in das Metaverse: Technologien, Anwendungen und Zukunft. Wiesbaden: Springer Fachmedien Wiesbaden, 2025: 85-132.

[6] Kumar S, Serrao S J, Sumanth G N G, et al. Web and Server Attacks: Research and Analysis[M]//Self-Powered AIoT Systems. Apple Academic Press, 2025: 295-314.

[7] Politis D, Nikiforos A, Charidimou D, et al. Interactive Educational TV and Special Education: Forwarding Distributed and Affectionate Learning via Cognitive Immersion[M]//Encyclopedia of Information Science and Technology, Sixth Edition. IGI Global, 2025: 1-22.

[8] Gong, M. (2025). Modeling Microservice Access Patterns with Multi-Head Attention and Service Semantics. Journal of Computer Technology and Software, 4(6).

[9] Ren, Y. (2024). Deep Learning for Root Cause Detection in Distributed Systems with Structural Encoding and Multi-modal Attention. Journal of Computer Technology and Software, 3(5).

[10] Meng, R., Wang, H., Sun, Y., Wu, Q., Lian, L., & Zhang, R. (2025). Behavioral Anomaly Detection in Distributed Systems via Federated Contrastive Learning. arXiv preprint arXiv:2506.19246.

[11] Pan, R. (2024). Deep Regression Approach to Predicting Transmission Time Under Dynamic Network Conditions. Journal of Computer Technology and Software, 3(8).

[12] Wang, Y., Zhu, W., Quan, X., Wang, H., Liu, C., & Wu, Q. (2025). Time-Series Learning for Proactive Fault Prediction in Distributed Systems with Deep Neural Structures. arXiv preprint arXiv:2505.20705.

[13] Wei, M. (2024). Federated Meta-Learning for Node-Level Failure Detection in Heterogeneous Distributed Systems. Journal of Computer Technology and Software, 3(8).

[14] Duan, Y. (2024). Continuous Control-Based Load Balancing for Distributed Systems Using TD3 Reinforcement Learning. Journal of Computer Technology and Software, 3(6).

[15] Tang, T. (2024). A meta-learning framework for cross-service elastic scaling in cloud environments. Journal of Computer Technology and Software, 3(8).

[16] Fang, B., & Gao, D. (2025). Collaborative Multi-Agent Reinforcement Learning Approach for Elastic Cloud Resource Scaling. arXiv preprint arXiv:2507.00550.

[17] Wu, Q. (2024). Task-Aware Structural Reconfiguration for Parameter-Efficient Fine-Tuning of LLMs. Journal of Computer Technology and Software, 3(6).

[18] Wang, Y. (2024). Structured Compression of Large Language Models with Sensitivity-aware Pruning Mechanisms. Journal of Computer Technology and Software, 3(9).

[19] Zhang, W., Xu, Z., Tian, Y., Wu, Y., Wang, M., & Meng, X. (2025). Unified Instruction Encoding and Gradient Coordination for Multi-Task Language Models.

[20] Sun, Y., Meng, R., Zhang, R., Wu, Q., & Wang, H. (2025). A Deep Q-Network Approach to Intelligent Cache Management in Dynamic Backend Environments.

[21] Zhan, J. (2024). MobileNet Compression and Edge Computing Strategy for Low-Latency Monitoring. Journal of Computer Science and Software Applications, 4(4).

[22] Ma, Y. (2024). Anomaly detection in microservice environments via conditional multiscale GANs and adaptive temporal autoencoders. Transactions on Computational and Scientific Methods, 4(10).

[23] Cheng, Y. (2024). Selective Noise Injection and Feature Scoring for Unsupervised Request Anomaly Detection. Journal of Computer Technology and Software, 3(9).

[24] Cui, W. (2024). Vision-Oriented Multi-Object Tracking via Transformer-Based Temporal and Attention Modeling. Transactions on Computational and Scientific Methods, 4(11).

[25] Wu, Y., Lin, Y., Xu, T., Meng, X., Liu, H., & Kang, T. (2025). Multi-Scale Feature Integration and Spatial Attention for Accurate Lesion Segmentation.

[26] Zhu, L., Cui, W., Xing, Y., & Wang, Y. (2024). Collaborative Optimization in Federated Recommendation: Integrating User Interests and Differential Privacy. Journal of Computer Technology and Software, 3(8).

[27] Liu, J. (2025). Reinforcement Learning-Controlled Subspace Ensemble Sampling for Complex Data Structures.

[28] Wang, H. (2024). Causal Discriminative Modeling for Robust Cloud Service Fault Detection. Journal of Computer Technology and Software, 3(7).

[29] Yang, T. (2024). Transferable Load Forecasting and Scheduling via Meta-Learned Task Representations. Journal of Computer Technology and Software, 3(8).

[30] Sun, Y., Zhang, R., Meng, R., Lian, L., Wang, H., & Quan, X. (2025). Fusion-Based Retrieval-Augmented Generation for Complex Question Answering with LLMs.

[31] Wu, Q. (2024). Task-Aware Structural Reconfiguration for Parameter-Efficient Fine-Tuning of LLMs. Journal of Computer Technology and Software, 3(6).

[32] Xu, T., Deng, X., Meng, X., Yang, H., & Wu, Y. (2025). Clinical NLP with Attention-Based Deep Learning for Multi-Disease Prediction. arXiv preprint arXiv:2507.01437.