Innovation and Technology Studies
Print ISSN: 3007-6919
Online ISSN: 3007-6927
DOI: https://doi.org/10.61784/its3022

DESIGN OF DATABASE STORAGE HIERARCHY AND RELIABILITY MECHANISM IN CLOUD-NATIVE ENVIRONMENT

GuangZhou Zhang
Wuhan University, Wuhan 430000, Hubei, China.

Abstract: With the in-depth popularization of cloud computing and microservice architecture, cloud-native databases have become the core infrastructure supporting the storage of massive business data and high-concurrency access. The tightly coupled architecture of traditional databases suffers from insufficient resource elasticity, high storage costs, and weak fault tolerance, making it difficult to adapt to the dynamic scaling, on-demand billing, and high-availability operation requirements of cloud-native environments. To address the above pain points, this paper proposes a database storage hierarchical architecture for cloud-native environments. Combining the design concept of compute-storage separation, a three-tier storage system consisting of a hot data cache layer, an online business storage layer, and a warm-cold data archive layer is constructed. Meanwhile, a multi-dimensional reliability guarantee mechanism is designed, covering four core modules: data consistency, fault self-healing, disaster recovery and backup, and anomaly isolation. An experimental environment was built for performance verification based on the TPC-C and TPC-H standard test sets. The results show that this architecture can effectively improve the storage resource utilization rate by more than 35%, reduce the cold data storage cost by 60%, and control the fault recovery time within 10 seconds, meeting the high-performance, low-cost and high-reliability storage demands of databases in cloud-native scenarios.
Keywords: Cloud-native database; Storage hierarchy; Compute-storage separation; Reliability mechanism; Hot and cold data separation; Fault self-healing

1 INTRODUCTION

Cloud-native technologies reconstruct the IT architecture by virtue of containerization, Kubernetes orchestration and other capabilities. As the core data infrastructure, cloud-native databases need to have the characteristics of compute-storage separation, elastic scaling and fault self-healing. At present, with the explosive growth of massive data, the collaborative design of storage hierarchy and reliability has become the key to solving the pain points. Cloud vendors and academic circles at home and abroad have carried out relevant research, but the existing solutions have problems such as rigid hierarchical strategies, weak adaptive scheduling, disconnection between hierarchical and reliability mechanisms, and low efficiency of fault self-healing. Therefore, based on the Kubernetes environment, this paper designs a three-tier storage hierarchical architecture and an end-to-end reliability mechanism to solve the three major problems of seamless hierarchical scheduling of data, reliability guarantee under compute-storage separation, and cloud-native operation and maintenance integration, providing a complete solution for the storage optimization of cloud-native databases.

2 RELATED TECHNOLOGIES

2.1 Core Characteristics of Cloud-Native Databases

A cloud-native database is a database service tailor-made for the cloud environment. Different from the simple "cloud deployment" mode of traditional databases, it has four core characteristics: the compute-storage separation architecture, where compute nodes and storage nodes are elastically scaled independently to eliminate the bottleneck of resource binding; elastic scaling, which automatically adjusts storage capacity and computing resources based on business traffic to realize on-demand billing; self-healing capability, which relies on Kubernetes Operator to realize automatic node fault migration and service restart; and cloud ecosystem compatibility, which seamlessly connects with container orchestration, service mesh, monitoring and alerting components, supporting multi-cloud and hybrid cloud deployment.
Mainstream cloud-native databases such as TiDB, OceanBase and PolarDB all adopt a decoupled design of storage and computing, and rely on cloud vendors' block storage and object storage for underlying storage, which not only improves scalability but also reduces hardware investment costs. However, the hierarchical strategies of such products are mostly biased towards fixed rules with insufficient adaptive scheduling capabilities, and the access performance of warm and cold data degrades significantly [1-2]. In addition, their reliability mechanisms mostly rely on the underlying capabilities of cloud vendors, lacking independent and controllable fault-tolerant design.

2.2 Database Reliability Guarantee Technologies

The core goal of database reliability is to achieve RPO=0 (zero data loss) and RTO≤10 seconds (fast recovery). The existing technologies mainly include: distributed consensus algorithms (Raft, Paxos) to ensure multi-replica consistency; multi-availability zone deployment to achieve regional-level disaster recovery; the combination of incremental backup and full backup to ensure data recoverability; and fault isolation mechanisms to block the spread of anomalies.
In cloud-native environments, reliability guarantee faces new challenges: node stability in containerized deployment is weaker than that in physical machines; network virtualization leads to delay fluctuations; the number of data transmission links increases under compute-storage separation, resulting in more fault points. Most existing solutions are designed for traditional distributed environments and are difficult to adapt to the dynamic and elastic deployment characteristics of cloud-native environments.

3 DESIGN OF CLOUD-NATIVE DATABASE STORAGE HIERARCHICAL ARCHITECTURE

3.1 Overall Architecture Design

Based on the Kubernetes orchestration environment, this paper designs a cloud-native database architecture with compute-storage separation and three-tier storage hierarchy. The architecture is divided into five modules: access layer, compute layer, storage hierarchy layer, management and control layer, and infrastructure layer. All modules are designed in a decoupled manner and interact through standardized interfaces, as shown in Figure 1.
Access layer: Responsible for client request access, load balancing and permission verification, compatible with mainstream protocols such as MySQL and PostgreSQL, and supports read and write request routing and distribution.
Compute layer: A stateless compute node cluster responsible for SQL parsing, query optimization and transaction execution. It realizes horizontal scaling relying on Kubernetes and does not store persistent data.
Storage hierarchy layer: The core module, divided into a hot data cache layer, an online storage layer and a warm-cold archive layer, to realize hierarchical data storage and automatic scheduling.
Management and control layer: Includes hierarchical scheduler, reliability management and control, monitoring and alerting, and elastic scaling components to realize automated operation and maintenance of the architecture [3-4].
Infrastructure layer: Relying on cloud-native storage resources, including memory, SSD cloud disks, HDD cloud disks, Object Storage Service (OSS) and archive storage.
The core design concept of the architecture is as follows: the compute layer focuses on business logic processing, the storage layer focuses on data persistence, the hierarchical scheduler dynamically adjusts the data storage level according to data access characteristics, and the management and control layer monitors the status of each layer in real time to ensure reliability and elasticity.

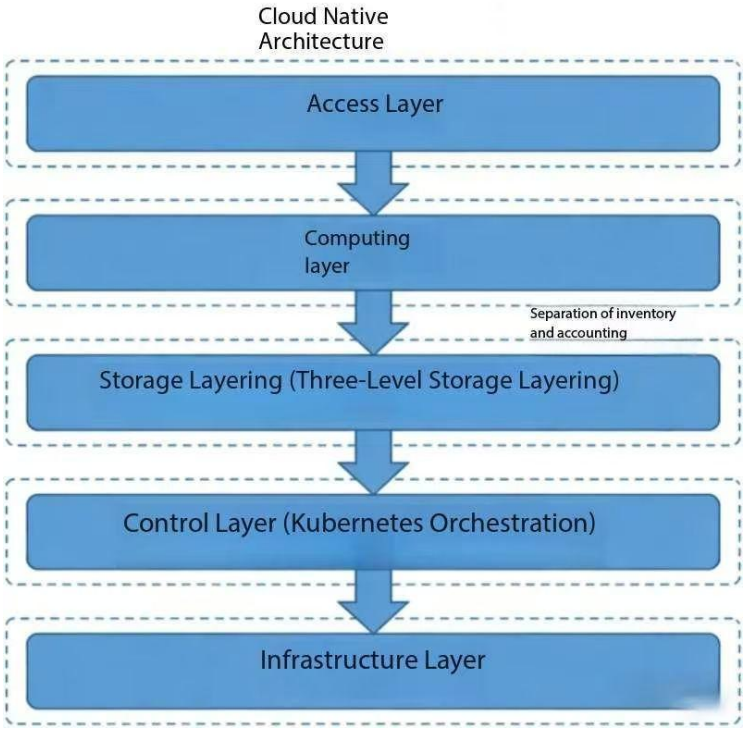

Figure 1 Cloud-Native Architecture

3.2 Design of the Three-Tier Storage Hierarchy

Combined with the characteristics of cloud-native storage resources and data access rules, this paper designs a three-tier storage hierarchy model, clarifies the positioning, storage media, data characteristics and core functions of each layer,

and achieves the optimal balance between performance and cost.

3.2.1 Hot data cache layer

The hot data cache layer is located in the local memory of compute nodes and distributed cache (Redis Cluster), positioned as a high-performance access layer for storing hot data with high access frequency and low latency sensitivity (such as user sessions, popular commodities, and real-time transaction data). The storage media adopt memory and high-speed SSD, with access latency controlled at the millisecond level [5].

Core functions: Caching hot data pages and indexes to reduce the access pressure on the underlying storage; supporting cache preheating, expiration elimination and consistent update to avoid cache penetration and avalanche problems; seamless connection with the compute layer to realize near-by response of query requests.

3.2.2 Online storage layer

The online storage layer is the persistent layer for core business data, adopting SSD cloud disks and distributed block storage, positioned as a highly available read-write layer for storing real-time business data and recent incremental data, supporting high-frequency read and write and transaction consistency. This layer adopts multi-replica deployment and relies on the Raft protocol to ensure data consistency, serving as the core storage carrier for the normal operation of businesses.

Core functions: Supporting ACID transactions, distributed locks and index construction; connecting with the compute layer to execute read and write requests; acting as a transit layer for hierarchical scheduling, receiving the eliminated data from the hot data cache layer and migrating warm-cold data downward.

3.2.3 Warm-cold archive layer

The warm-cold archive layer adopts object storage and archive storage, positioned as a low-cost persistent layer for storing historical data, log data and archive backup data with low access frequency and weak timeliness. Warm data is stored in low-frequency object storage to support regular access; cold data is stored in archive storage and only used for data retention and compliance auditing [6-8].

Core functions: Supporting data compression and encrypted storage; providing batch reading and offline query interfaces; the storage cost is only 10%-20% of that of the online storage layer, which greatly reduces the storage overhead of massive cold data.

3.3 Adaptive Hierarchical Scheduling Mechanism for Data

To realize the automatic flow of data among the three-tier storage layers, this paper designs an adaptive hierarchical scheduler based on multi-dimensional characteristics, whose core process includes three links: data heat evaluation, hierarchical decision-making and migration execution.

3.3.1 Data heat evaluation model

The scheduler collects data access indicators in real time and constructs a multi-dimensional heat scoring system, including the following indicators: access frequency (QPS), last access time, update frequency, business priority and data size. A heat score is generated through weighted calculation, and the data is divided into three categories: hot data, warm data and cold data:

Hot data: Score ≥80, accessed more than 1000 times within 7 days, updated in real time, and migrated to the hot data cache layer.

Warm data: Score 60-80, accessed 100-1000 times within 30 days, updated at low frequency, and retained in the online storage layer.

Cold data: Score ≤60, no access or only query within 90 days, and migrated to the warm-cold archive layer.

3.3.2 Hierarchical migration strategy

The scheduler adopts an asynchronous non-blocking migration mode to avoid affecting business read and write: when hot data is eliminated, it is synchronously written to the online storage layer; when warm data is degraded to cold data, a full backup is first made to the archive layer, and redundant data in the online layer is deleted after consistency verification; when cold data is restored to hot data, it is first loaded into the cache layer and then synchronously to the online layer. Meanwhile, a migration current limiting mechanism is set to control bandwidth occupation and avoid business performance fluctuations [9].

4 DESIGN OF CLOUD-NATIVE DATABASE RELIABILITY MECHANISM

In view of the risks such as node failure, network fluctuation and data corruption in cloud-native environments, this paper constructs an end-to-end reliability guarantee system covering four dimensions: data consistency, fault self-healing, disaster recovery and backup, and anomaly isolation, to fully guarantee data security and business continuity.

4.1 Multi-Replica Data Consistency Mechanism

The online storage layer adopts the three-replica Raft consensus algorithm, storing data shards on storage nodes in different availability zones to achieve strong replica consistency: write requests can only return success after being synchronized to most replicas; read requests prefer to access the primary replica to ensure data real-time performance; in case of node failure, the Raft algorithm automatically elects a new primary replica with a switch time ≤3 seconds. Aiming at the compute-storage separation scenario, a Write-Ahead Log (WAL) mechanism is designed, in which transaction logs are first written to the cache layer and then asynchronously persisted to the online storage layer to avoid

data inconsistency caused by write failures [10].
The warm-cold archive layer adopts Erasure Coding (EC) technology, which splits data into multiple data blocks and check blocks and stores them dispersedly on different nodes. The complete data can be reconstructed by recovering only part of the blocks, which reduces the redundancy cost while ensuring reliability, with the redundancy rate reduced from 300% to within 150%.

4.2 Cloud-Native Fault Self-Healing Mechanism

Relying on Kubernetes Operator and custom controllers, the automatic fault self-healing of the hierarchical storage architecture is realized:
Node fault self-healing: Monitor the health status of storage nodes and compute nodes in real time, isolate the nodes immediately when anomalies occur, automatically rebuild Pods on healthy nodes and synchronize data replicas, with RTO≤10 seconds.
Storage layer fault self-healing: When a cache layer node fails, the cache is automatically rebuilt and data is preheated from the online storage layer; when an online layer replica is damaged, replica repair is triggered and data is synchronized from healthy replicas; when archive layer data is corrupted, it is restored from the backup set.
Network fault self-healing: In case of network partitioning, the local cache is enabled to ensure the readability of core businesses, and incremental data is automatically synchronized after network recovery to avoid data inconsistency.

4.3 Multi-Level Disaster Recovery and Backup Mechanism

A multi-level backup system of local backup + cross-regional disaster recovery is designed to meet compliance requirements and extreme fault scenarios:
Real-time backup: Incremental log backup is enabled for the online storage layer, which is synchronized to object storage every 5 minutes to ensure RPO=0.
Regular full backup: A full backup is performed at midnight every day, stored cross-regionally in the archive cluster, and the backup data is encrypted and compressed and retained for 90 days.
Multi-availability zone disaster recovery: The compute layer and storage layer are deployed in multiple availability zones, and traffic is automatically switched to the standby availability zone in case of a single availability zone failure.
Disaster recovery drill: The management and control layer supports one-click disaster recovery switch drill to verify the effectiveness of the recovery process and avoid operational errors in case of failures.

4.4 Anomaly Isolation and Security Protection

To avoid the spread of faults, hierarchical isolation and permission management and control are realized: each storage layer adopts resource quota isolation, and anomalies in a single layer will not affect other layers; storage access permission control is enabled, and only authorized compute nodes and management and control components can access storage data; data transmission and storage are encrypted throughout the process to prevent data leakage and tampering; for slow queries and malicious access, a current limiting and circuit breaking mechanism is enabled to protect the stability of the storage layer.

5 EXPERIMENTAL VERIFICATION AND RESULT ANALYSIS

5.1 Experimental Environment Construction

An experimental environment was built based on the Kubernetes 1.28 cluster, configured with 3 Master nodes and 6 Worker nodes. The cloud-native storage resources include: memory cache (64GB), SSD cloud disk (1TB), and object storage (10TB). The self-developed cloud-native architecture is adopted for the database, compatible with the MySQL protocol. A comparative test was conducted on the performance, cost and reliability indicators between the traditional distributed database (tightly coupled architecture) and the architecture proposed in this paper.
Test tools: TPC-C (Transaction Processing Performance Council - C), TPC-H (Transaction Processing Performance Council - H); Monitoring indicators: throughput (TPS/QPS), access latency, storage cost, fault recovery time, resource utilization rate.

5.2 Performance Test Results

5.2.1 Transaction processing performance

The TPC-C test results show that in high-concurrency scenarios, the throughput of the architecture proposed in this paper is 28% higher than that of the traditional architecture, and the average access latency is reduced from 50ms to 18ms. The core reasons are that the hot data cache layer greatly reduces the access to the underlying storage, the compute-storage separation architecture eliminates the bottleneck of computing and storage resources, and the elastic scaling capability supports high-concurrency requests.

5.2.2 Hierarchical storage performance

For the hot and cold mixed data scenario, the access latency of hot data of the proposed architecture is stably within

2ms, the access latency of warm data is 15ms, and the batch query latency of cold data is controlled within 500ms, meeting the business access requirements; the storage resource utilization rate is increased from 45% of the traditional architecture to 82%, and the cold data storage cost is reduced by 62%.

5.3 Reliability Test Results

Three abnormal scenarios of node failure, network partitioning and data corruption were simulated. The test results show that the average fault recovery time of the proposed architecture is 7.2 seconds, much lower than 45 seconds of the traditional architecture; the data consistency accuracy rate is 100% with no data loss; the cross-availability zone fault switch time is ≤8 seconds with no business perception. The success rate of disaster recovery and backup recovery is 100%, the full backup recovery time is controlled within 5 minutes, and the incremental recovery time is ≤1 minute.

5.4 Experimental Conclusions

The cloud-native database storage hierarchical architecture and reliability mechanism proposed in this paper are superior to the traditional tightly coupled architecture in three dimensions: performance, cost and reliability, which can effectively adapt to the dynamic and elastic cloud-native scenarios and meet the enterprise-level database storage demands.

6 CONCLUSION

Aiming at the pain points of database storage in cloud-native environments, this paper designs a three-tier storage hierarchical architecture and an end-to-end reliability mechanism. Through technologies such as compute-storage separation decoupling, adaptive hierarchical scheduling, multi-replica fault tolerance and fault self-healing, the balance between storage performance and cost is achieved, and the data reliability and architecture elasticity are improved. The experimental results show that this architecture can significantly improve resource utilization, reduce storage costs, shorten fault recovery time, and has strong engineering implementation value.

Future research directions: First, optimize the data heat prediction algorithm combined with large models to improve the accuracy of hierarchical scheduling; second, adapt to the Serverless architecture to realize the ultimate elasticity of storage resources; third, optimize the cross-cloud storage hierarchy and reliability mechanism to support unified multi-cloud management and control; fourth, reduce the access performance attenuation of warm and cold data to improve the user experience of hierarchical storage.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Chen C, Hu N N, Song L H, et al. Research on the Architecture Design and Performance Evaluation of Experimental Database Based on Big Data Technology. Laboratory Testing, 2025, 3(06):47-49.

[2] Wang Y P. Analysis of the Impact of AI Agents on the Transformation of Database Architecture. China Education Network, 2025(06): 78-80.

[3] Zhou M, Xie F, Chu Z G, et al. Research on Real-time Processing Technology of Data Middle Platform Based on Cloud-Native Architecture. Office Automation, 2025, 30(21): 24-26.

[4] Ji G Y, Duan G D, Chen P, et al. Design of Heterogeneous Storage Directory Update Mechanism Based on Cloud-Native Architecture. Information Technology and Informatization, 2025(11): 196-200.

[5] Thomasian A. Storage Systems: Organization, Performance, Coding, Reliability, and Their Data Processing. Academic Press, 2021.

[6] Zhang Y, Wang M, Guo,Y, et al. Towards dynamic and reliable private key management for hierarchical access structure in decentralized storage. In Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, 2023: 3371-3380.

[7] Yan B, Yang Y, Guo W, Zet al. Big Data Storage Index Mechanism Based on Hierarchical Indexing and Concurrent Updating. 2022.

[8] Xiao Z, Li H, Li W, et al. Design and Analysis of a Hierarchical Fault Tolerance Mechanism for Hierarchical Multi-Identifier System. In Proceedings of the 2025 4th International Conference on Big Data, Information and Computer Network, 2025: 895-900.

[9] Zhang Y, Jin G, Li J, et al. Hierarchical Storage for Massive Social Network Data Based on Improved Decision Tree. Mobile Networks and Applications, 2024: 1-15.

[10] Sasikumar A, Ravi L, Kotecha K, et al. A secure big data storage framework based on blockchain consensus mechanism with flexible finality. IEEE Access, 2023, 11: 56712-56725.