Research on the Method of Massive Data Storage Management

[1st] Wencheng Zhang [a]*, [2nd] Chao Li[b], [3rd] Junyi Duan [c]

[a]19939888948@163.com,[b] 1817995954@qq.com,[c] 19939886561@163.com

The People's Liberation Army 63892, Luoyang City, Henan Province, China

Abstract. With the deepening of informatization, the explosion of data generated by various industries presents data storage challenges due to data retention and the expectation of on-demand access to storage applications from any device.Based on the requirements of mass data storage, management and use, this paper proposes a distributed data storage architecture solution based on microservices, focusing on data storage, disaster recovery and backup capability, data consistency and data labeling.The solution is based on SpringCloud microservice architecture, MinIO object storage system, MySQL database and MangoDB database as data storage carriers, and uses SpringCloud+Vue front-end separation technology and Redis memory database high-performance data management and sharing technology.To realize the requirements of high-speed data storage and management, it provides a new idea for the storage and management of massive multi-format data.

Keywords: Data storage, micro-service, object storage

1 INTRODUCTION

In recent years, the importance of data in digital transformation has been elevated to an unprecedented level, and data-driven decision-making, scheduling, and operation have given enterprises intelligent wings and brought enormous business value. Data is the atom that builds a digital world that is equivalent to the physical world. Data contains the essence of business and the source of innovation. Whoever can master the ability of data can stand out in the digital competition. The first problem to be solved when applying data is data storage, which is the cornerstone of a series of data applications.

The data that enterprises and organizations need to store mainly refers to the data that needs to be collected and analyzed, mainly including business data, process data, log data, time series data, and other formats of data. The storage of data is divided into structured data storage, as well as semi-structured and unstructured data storage according to data types. With the growth of business, enterprises and organizations have deployed various application systems, resulting in a very complex data type. The production business system generates a large amount of structured data, video surveillance systems or conference systems generate a large number of

* Brief introduction of the first author: Zhang Wencheng (1997 -), Unit 63892, Research intern, Main research direction: command information system test and software evaluation, No. 17 Leshan Road, Jianxi District, Luoyang City, Henan Province.

ICEMBDA 2023, October 27-29, Tianjin, People's Republic of China

DOI 10.4108/eai.27-10-2023.2342009

audio and video files, and file sharing systems generate a massive amount of small files. The construction of information technology is becoming increasingly integrated, and traditional IT architecture cannot cope with the current situation of the geometric growth of enterprise and organizational data, and the continuous extension and iteration of application services. The main problems faced by traditional data storage systems are: (1) The traditional single architecture system has poor scalability, difficult development and operation, and high code coupling, which are becoming increasingly prominent. (2) Traditional data storage mainly focuses on structured data storage, with less semi-structured and unstructured data. In recent years, the explosive growth of unstructured data in target data has led to the problem of massive multi format data storage. (3) Information silos: Traditional storage provides decentralized storage solutions for various application systems, which are independent and unrelated to each other, and cannot achieve data sharing and interoperability[1].

This article proposes a distributed storage and management function for highly available file objects and structural data based on the SpringCloud microservice architecture, MinIO cluster, MongoDB cluster, and Redis cluster. It integrates and extends previously dispersed or separated service projects and business systems into a unified platform system to provide external service support. Data storage applications can call corresponding services or service interfaces as needed, Thus achieving the centralization and intensification of services.

2 DEMAND ANALYSIS

With the continuous deepening of global informatization, the scale of internet users has significantly increased, while the amount of data generated by enterprises and organizations continues to increase. Enterprises and organizations face the generation, transmission, processing, and retention of large-scale data. The main data storage requirements faced include multiple aspects:

(1)The growth of data volume: The explosive growth of data volume is the main driving factor of current data storage demand. The large-scale data storage demand covers structured data (such as database data), unstructured data (such as text, images, audio and video, etc.), and real-time data streams (such as sensor data, log data) and other types of data.

(2)Data disaster recovery backup requirements: In order to protect critical data from hardware failures, human errors, natural disasters, and other factors, enterprises and organizations need to conduct data disaster recovery backup. Data disaster recovery is the process of ensuring that data can be quickly restored and accessed after a disaster occurs. Data backup is the process of copying data to another location or storage medium for recovery in the event of original data loss or damage. The demand for disaster recovery backup requires enterprises and organizations to have sufficient storage capacity to store replicas and redundant data.

(3) The demand for real-time data processing: Many application scenarios require real-time processing and analysis of data. For example, financial transactions require real-time data monitoring and identification of abnormal transactions, logistics companies need to track cargo location information in real time, and IoT applications need to process data from sensors in real time. The demand for real-time data processing requires data storage systems to have high

performance and low latency[2], be able to handle large-scale real-time data streams, and provide timely feedback and response.

(4) The demand for multi regional data transmission and storage: With the expansion of enterprises' business and multi regional distribution, cross regional data transmission and storage have become an important requirement. This includes cross regional disaster recovery backup, cross regional data sharing, etc. Enterprises and organizations need solutions that can efficiently transmit and store data to meet the mobility and availability of data between different regions.

(5) Data archiving and analysis requirements: Enterprises and organizations typically need to archive old or infrequently used data to save storage space and provide support for subsequent data analysis and mining. Data archiving is the process of removing data from the primary storage system and storing it on low-cost long-term storage media. Data archiving can be based on standards such as age, access frequency, and business value of the data. Archived data needs to maintain data integrity and retrievability, and can be quickly restored for use when needed. The demand for data archiving enables enterprises and organizations to optimize primary storage space and meet regulatory business requirements.

2.1 Program overview

The distributed microservice data storage solution includes three major service modules: data storage management terminal, data storage service API interface, and data storage cluster. Based on object storage clusters, relational database clusters, and memory database clusters, high-availability distributed storage and management functions of files and structural data are realized. The bottom layer is the data storage cluster, including object storage cluster and relational database cluster. On top of this, a resource management layer is built to visually manage the data storage cluster and at the same time visually manage the stored data to form a data storage management terminal; On the one hand, a visual data storage service is formed based on the API interface. The API interface service can be connected to other systems and provide external functions such as data addition, deletion, modification, data remark, and labeling.

Data storage cluster: the data storage service is deployed on multiple servers using the cluster mode, and through the load balancing scheme, the upper API interface service calls the load balancing server address to achieve dynamic access. It mainly includes object storage cluster server and structured data database cluster. In the object storage cluster mode, when a file is uploaded or deleted to one of the object storage servers, other file servers automatically synchronize file operations. At the same time, it can be realized that when a certain object storage server goes down, the corresponding file data will be returned from the available file server when the upper API operates the file. When the down server returns to normal operation, the file server will be automatically merged into the cluster to provide data operation services for the upper API.

Data storage management terminal: The storage management terminal comprehensively manages metadata and tags, object queries, permission policies, life cycles, etc. in a visual form. In addition, the storage management terminal integrates configuration management, capacity management, monitoring management, alarm center, log center and other functions.

Data storage service API interface service: API interface service provides data storage services for data service terminals and serves as a service for docking with other systems. On the one hand, it provides service support for data cloud service terminals. On the other hand, it combines the user permission system of the terminal to provide data collection terminals. Provides functions such as adding, deleting, modifying data, and labeling data.

Tables can be very difficult for people using screen reader technology to understand unless they include markup that explicitly defines the relationships between all the parts (i.e.: headers and data cells). *A key to making data tables accessible to screen reader users is to clearly identify column and row headers*[3]. In Word, authors should identify which row or rows contain column headers. Below are the steps to do this:

1. Select that table's row, then right-click the row and select "Table Properties";
2. In the *Table Properties* window, click the *Row* tab and select the box that says "Repeat as header row at the top of each page."

Or

Apply the "table head" style by highlighting the respective row and applying the "**TableHead**" style found in the "Body Element" section of the ACM Master Article Template.

2.2 Design ideas

Data in business scenarios are mainly divided into two categories: structured data, semi-structured and unstructured data, as well as other types of data that may appear in the future.

In order to cope with the uncertainty of future data demand scenarios, microservice technology is used during system design to integrate the underlying file object storage system and database storage system into microservices, and provide external services as a whole. If there are new demand scenarios, only It is necessary to develop corresponding functions or add new components in microservices.

In order to cope with the potential high concurrent calls of the microservice system in the future, which will cause the performance of the microservice system itself to be tight, the microservice system itself will also be deployed in cluster mode. In the design plan, the service terminal and the data cloud service interface are separated, the data cloud service is treated as a separate service, and the underlying file system and relational data storage system adopt cluster mode.

In order to meet the needs of different applications, data cloud service terminals and data cloud service API interface services are implemented based on microservice technology; at the same time, various service modules provided externally should also be microservice-based. When the application side needs which service modules, Just enable the corresponding microservice.

To accommodate readers with color vision differences, figures should still be usable when printed in grayscale. Refer to elements of the figure with non-color terms, for example "indicated as squares" instead of "indicated in blue". Use different patterns in bar charts, different line patterns in graphs, and different shapes in plots to distinguish groups of elements and reinforce color differences.

2.3 System structure

As shown in Figure 1, the system is designed and implemented based on the microservice architecture. At the same time, the service interface provided for the application side is implemented in the form of microservices, supporting the application side to call the service interface according to functional requirements; through service permission control, different business application domains It realizes business isolation among the information and improves the reliability of system and data; through the dynamic scaling and expansion of microservice applications, it achieves high expansion of business functions and meets the business needs in multiple scenarios.

2.3.1 Data storage layer.

The file storage cluster is built using the SDK officially provided by MinIO for secondary encapsulation to realize MinIO cluster management, including the cluster's MinIO content resource management (which can be regarded as replacing MinIO's own management interface) and the monitoring and management of MinIO cluster information, such as MinIO node status, online and offline control, etc.; the file storage service itself is also a microservice, and it needs to call the public Redis cluster and relational library cluster to achieve high availability and satisfy persistent storage.

The structured data cluster is built using a MongoDB database cluster. The MongoDB cluster is used to store file index data and structured data. The MySQL database is used to store client data such as users, roles, permission data, etc. The Redis cluster is responsible for caching and multiple resource management. The service backend node ensures data consistency through Redis cluster to achieve high availability.

2.3.2 Data service interface layer.

Mainly responsible for converting data into service APIs for direct use by online applications[4]. API services provide data storage services for data service terminals and serve as services to connect to other systems, forming top-level applications such as: reports, data screens, rule engines, etc. for direct use. Data can manage services and service usage methods such as resource servitization, service usage authentication, flow control, authentication, black and white lists, etc., and provide external functions such as data addition, deletion, modification, data labeling, etc., and also provides service browsing. , service application, call volume query and other enterprise-level management capabilities.

The API interface is a restful interface, which can be called through http requests and is not restricted by language. It only needs to call the network request function of the development language. Later, the restful interface can also be encapsulated into a javasdk or other language sdk, which is more convenient to call.

2.3.3 Data storage terminal (application layer).

The data storage terminal is a graphical web application for users to conveniently manage data storage services. It mainly provides a wealth of management functions at bucket, folder and file levels, metadata and tags, object query, permission policies, and life cycle management. , helping customers using data storage to lower their usage threshold.

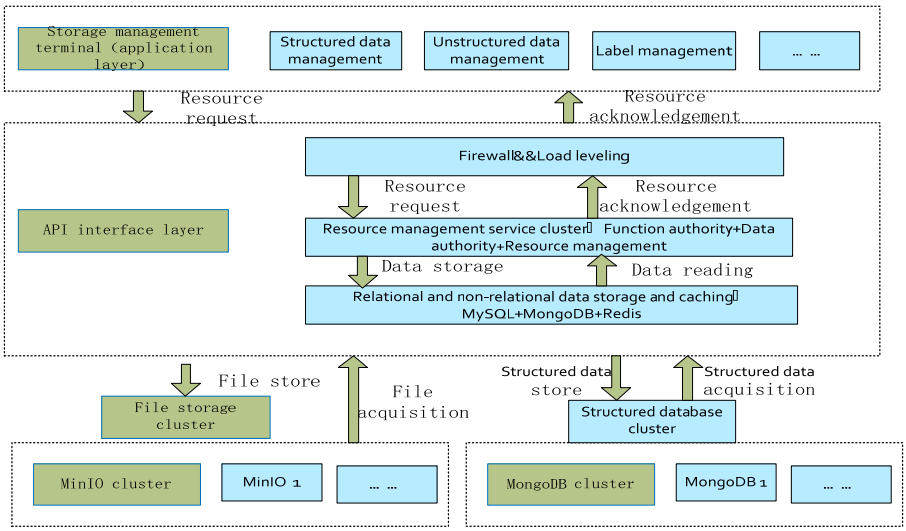

Figure 1: System architecture diagram

Adopting the industry's advanced SpringBoot architecture design concept and using the SpringCloud distributed microservice framework, the service network is split into a finer granularity, which is conducive to resource reuse, can more accurately formulate optimized service plans, and improves system maintainability. During the development process, the low-coupling characteristics of Spring Cloud are fully utilized, and different microservice modules can be developed in parallel and independently, greatly shortening the development cycle.

2.4 Technical Support

2.4.1 MongoDB.

MongoDB is a popular open source document database known for its high performance, extensibility, and flexibility. It belongs to the category of NoSQL (non-relational database). Compared with the traditional relational database (such as MySQL, Oracle), it uses different data models and query languages. MongoDB uses documents to store and represent data.

Document database: a document is a structure similar to JSON (JavaScriptObjectNotation) that uses key-value pairs to store data. A document can contain different types of fields, including strings, integers, arrays, nested documents, and so on. High performance: MongoDB has excellent performance features and can handle a large number of concurrent read and write operations. It provides low-latency access by manipulating data in memory and supports horizontal scaling to handle larger loads.

Scalability: MongoDB can easily scale horizontally on multiple servers to cope with large-scale data storage needs. Through data slicing (sharding) technology, data can be stored on multiple nodes to achieve load balancing and high availability.

Powerful query language: MongoDB uses a flexible query language to retrieve and manipulate data, called MongoDB query language (MQL). MQL supports a wide range of query operations,

including conditional query, sorting, projection, aggregation and other functions. High availability: MongoDB supports master-slave replication and automatic failover mechanisms to ensure high availability of data. Through replication, data can be automatically synchronized on multiple nodes, and when the primary node fails, the system can automatically switch to the standby node.

2.4.2 MinIO.

MinIO is a very popular open source object storage server, which is perfectly compatible with Amazon's S3 protocol and has very friendly support for K8s. It is designed for cloud native workloads such as AI. MinIO can provide data workload, including building high-performance cloud native data machine learning, big data analysis, mass storage infrastructure and so on.

Access control: Minio provides a variety of access control policies that allow fine-grained access control over users, groups, buckets and objects to ensure that only authorized users can access data. For example, you can use IAM (IdentityandAccessManagement) administrative tools to create users and groups, and use policies (Policy) to restrict user access to buckets and objects.

Data encryption: Minio supports data encryption, which can be protected using server-side encryption (SSE) or client-side encryption (CSE). Server-side encryption encrypts data when it is stored, while client-side encryption requires encryption and decryption when uploading and downloading data. In addition, Minio also supports custom encryption methods, and you can choose the appropriate encryption algorithm according to the needs of the enterprise.

Prevent data loss: Minio uses a distributed architecture where data is replicated to different nodes multiple times, ensuring that data is not lost even if one node fails. In addition, Minio also supports data erasure code (ErasureCode) technology, which stores data on different nodes, and can recover data even if multiple nodes fail at the same time.

Secure network transmission: Minio uses TLS/SSL to protect the security of data during network transmission and to ensure that the data will not be eavesdropped or tampered with during transmission. In addition[5], Minio also supports self-signed certificates, which can independently issue digital certificates according to the needs of enterprises to protect the security of data.

2.4.3 Redis.

Redis is an open source API that is written in ANSI C language, supports the network, can be memory-based and persistent, Key-Value database, and provides multiple languages.

Unlike MySQL databases, Redis data is stored in memory. Its read and write speed is very fast and can handle more than 100000 read and write operations per second. Therefore, redis is widely used in caching. In addition, Redis is often used to do distributed locks. In addition, Redis supports transactions, persistence, LUA scripting, LRU-driven events, and multiple clustering scenarios. When Redis works, the data is stored in memory, in case the server is powered off, all data will be lost. In view of this situation, Redis uses persistence mechanism to enhance data security. To put it bluntly, it is to save the data in memory to the hard disk[6].

Caching: using Redis, you can build a caching server with excellent performance. The query request first looks up the required data in Redis, and returns directly if it can be queried (hit), which greatly reduces the pressure on the relational database.

Temporary data storage location: using token (token) as the user's identity when logging in to the system, this token can be temporarily stored in Redis.

Streaming data deduplication: there is a data type in Redis that is set, much like the Set collection in Java, which does not allow duplicate data to be stored. With this feature, we can use set type to store streaming data in Redis to achieve the purpose of deduplication.

3 KEY TECHNOLOGY ANALYSIS

3.1 Data storage reliability control

Data reliability mainly considers that data should be accurate, complete, consistent and reliable. As shown in Figure 2,this includes ensuring the accuracy of data entry, data cleaning and deduplication, and handling errors and outliers in the data.

1. Structured data storage permission verification: Whether the user inputs data directly through the system or through the API interface, the user's data entry permission will be verified. If the user does not have data entry permission, he or she will not be able to enter data.
2. Structured data storage format restrictions: When users upload structured data, the data service system will limit the parameters of the uploaded data and verify the data format. Users can only pass in data with specific format parameters to specific data modules. If If the incoming data does not meet the specifications, it will be rejected and an exception value will be thrown. The user can modify the data format according to the exception value prompt or submit an application to the backend administrator to pass it into the database in a new format. This can ensure that structured data is stored in the system. quality.
3. Set tag elements when storing structured data: When structured data is stored in the database, the data creation time, user corresponding to the uploaded data, and data tags will be recorded to facilitate later data retrieval.
4. Query whether the data is duplicated when storing structured data: When the system transfers data to the database, it will first search whether the data already exists in the database. If it already exists, it will not be transferred again to prevent users from inputting multiple duplicate data.
5. Structured data storage failure log: Whether the system inserts data through the API interface or the system interface, if the data cannot be imported normally, an exception will be thrown. For users who save data through the data terminal interface, an exception prompt message will pop up directly on the interface to inform the user of the reason for the exception in data storage, allowing the user to combine the exception information for subsequent processing. At the same time, the system will later record the exception record information of the data stored by the user. . For data stored through the API interface, if the data storage fails, the API interface system will automatically record log information. The log information will record which API interface the stored abnormal data comes from, the

calling device IP address or device name, exception time, etc. Information to facilitate subsequent troubleshooting.

6. File data is stored in incremental storage of files with the same name: For file data storage, MinIO will automatically overwrite files uploaded with the same name. In order to ensure that files with the same name are not overwritten each time they are uploaded, the system will automatically rename the uploaded file after the original file name. The file upload time will be automatically appended to distinguish different files.
7. When the file data is saved, the marking elements are set: when the file data is stored in MinIO, the creation time of the data, the user corresponding to the uploaded data, and the data label are synchronously recorded and stored in the database MongoDB to facilitate later data retrieval.
8. Large file data is stored and uploaded in parts: For uploading larger files, upload in parts to increase the file upload speed. The minimum MinIO fragment is 5M. When uploading files larger than 1G, the MinIO fragmented long transfer method can be used to achieve the upload. Automatically fragment files, wait for each fragment to be uploaded, then combine the fragmented files uploaded to the server into the final uploaded file, and check whether the md5 value of the final file is consistent with the original file. To verify and compare the storage performance of MinIO, MinIO file storage includes three typical application scenarios, see Table 1.

Table 1: Typical application scenario table

Application scenarios	File size	Application features
Log	8-10Mbyte	Massive small files, random reading and writing, more writing and less reading
Documentation	1-200Mbyte	Large files, write more and read less
Photos, videos, etc	1-1000Mbyte	Large files, random reading and writing, more writing and less reading

3.2 Data storage reliability control

To ensure that data is not lost or damaged during storage and processing, using redundant storage and backup strategies can improve data reliability.

1. Automatic redundant storage of file data: After the file data is transferred to the MinIO file cluster system, MinIO will automatically synchronize the uploaded files to each server. When a MinIO server goes down, the upper-layer API will operate the file from the available file server. The corresponding file data is returned. When the downed server resumes normal operation, the file server will automatically be incorporated into the cluster to provide data operation services for the upper-layer API.
2. Automatic repair of storage file data: When MinIO detects that a node fails or a data block is damaged, it automatically obtains the same data block from other available nodes or replicas according to the disaster recovery backup mechanism and copies it to the new node. Through automatic repair, MinIO ensures that redundant copies of data remain in a healthy state while providing high availability and data integrity. Set the automatic repair function for the MinIO cluster. In an asynchronous manner, when a file is damaged, MinIO will

automatically trigger the repair operation and perform data recovery in the background. The specific repair time depends on factors such as the size of the data in the cluster, network bandwidth, and storage device performance.

3. Automatic storage file failure check: Set up the MinIO server to periodically check the health status of the node. If a node fails or becomes inaccessible, MinIO will detect the failure and perform failover operations quickly.

3.3 Data query reliability control

1. Use MongoDB for file retrieval: after the file is uploaded to MinIO, the file-related information such as bucket bucket name, file label, upload user, object object key (file path and file name), file name, file type and other file external information will be stored in the MongoDB database. When a user queries a file, it retrieves the file information from MongoDB. When the user wants to view a specific file, the system directly uses bucket and object object keys to MinIO a specific file object information in the acquired external information of the file.
2. Paging query reduces the query volume: A single large number of queries will cause excessive load on the server. For query requirements, query results can be returned by paging. For example, if a query returns 10 million data items at a time, paging query can be used to achieve each query. A page of 1,000 items is divided into 10,000 pages, so that each request only needs to return 1,000 pieces of data, which greatly reduces the query pressure on the server.
3. Adding indexes improves query efficiency: Indexes can speed up database retrieval. By creating appropriate indexes, the database can locate and retrieve data faster, reducing query execution time. Especially in large data sets and complex query scenarios, indexes can significantly improve query performance. Using Redis to cache query data can store query results in memory, thus speeding up queries. By using Redis cache, the load on the database can be reduced. When the query results already exist in the cache, there is no need to query the database again, thus reducing the load on the database.

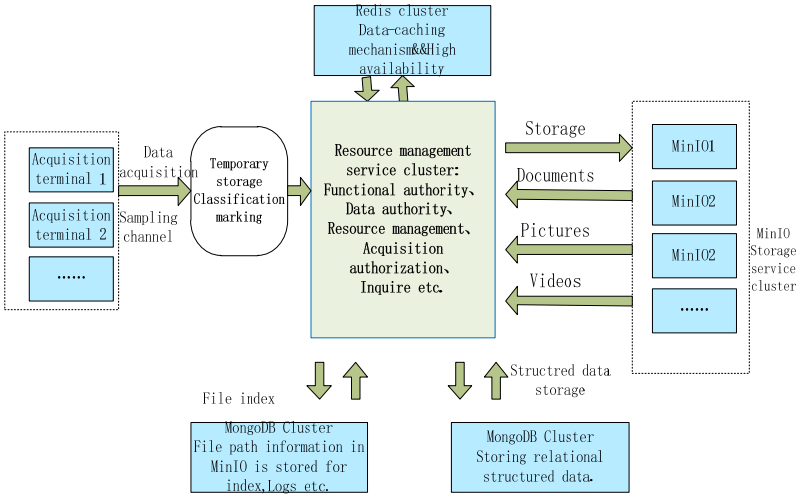

Figure 2: System operation flow chart

3.4 Achieve high availability and fault tolerance of Minio

1. Distributed architecture: MinIO is designed with a distributed architecture and can scale horizontally to provide high availability and high performance storage services. Data redundancy and fault tolerance are provided by dividing the data into multiple fragments and distributed storage.
2. Data redundancy and fault tolerance mechanism: MinIO provides a highly available storage scheme to ensure the reliability and persistence of data through data redundancy and fault tolerance mechanism. It supports multi-copy replication and fault recovery of data to ensure the availability of data.
3. Multi-node deployment: the availability and fault tolerance of MinIO can be improved through multi-node deployment. Data can be distributed on multiple nodes for higher availability and fault tolerance.
4. Load balancing: load balancing technology can be used to distribute requests to multiple MinIO nodes to improve system availability and performance.

By adopting distributed architecture, data redundancy and fault-tolerant mechanism, multi-node deployment and load balancing, we can achieve high availability and fault tolerance of Minio, so as to meet the needs of enterprise applications. There are no centralized management and control nodes in the whole system, each data node has the ability to undertake the functions of any data node, and the nodes cooperate and communicate with each other through internal efficient distributed protocols. This decentralized and stateless fully distributed data processing architecture is the key for the system to achieve horizontal and linear scalability, which effectively ensures that the whole system has no single point of failure and no performance bottleneck. This design allows users to purchase or expand capacity for the first time without considering independent metadata servers or independent network management servers[7].

Under the same test verification conditions, compare the performance indicators before and after using the massive data storage solution, including file access latency and write rate. 20 repeated experiments were conducted for each scenario, and the results in 6 aspects were averaged. The performance test results are shown in Figure 3. From the experimental results, the storage performance has been improved after using the massive data storage technology solution, among which a large number of small The effect is most obvious in file scenarios. Such scenarios are common in daily life and can effectively improve user experience.

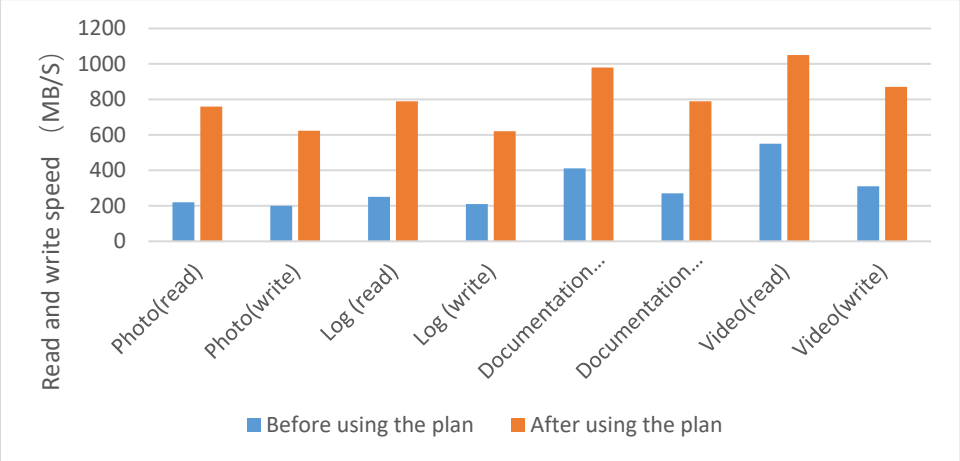

Figure 3: Storage solution verification comparison

The massive data storage solution can improve the overall flexibility of the data storage management platform and minimize I/O access latency[8]. Compared with connecting separate application servers and storage arrays through dedicated storage networks, it provides flexibility for the data storage management platform. While ensuring scalability, it also eliminates issues such as capacity planning, performance planning, expansion, disk failure repair, data reliability, and management complexity that are encountered during traditional array storage deployment and management.

4 CONCLUSION

The data storage distributed architecture solution based on micro-service realizes the storage management of massive data, uses the SpringCloud+Vue front and rear separation technology architecture, modularizes all kinds of data and adds API interface services, which can realize the customized service of data storage management, fully consider the data reliability, and provide a guarantee for the safe and reliable storage of massive data. The distributed MinIO cluster is deployed to provide data storage services for the massive structured and unstructured data generated in the production process of enterprises and organizations, and the combination of relational database and object storage is adopted to improve the performance of massive data storage and management.

The overall solution is cluster deployment, micro-service architecture development, through Redis clusters to share cached data, can meet the overall high availability. The data storage part is also a cluster deployment, which can achieve data disaster recovery. The relevant technologies used are open source technologies. According to the complex data types of business systems and the needs of massive unstructured data storage, relevant enterprise organizations can carry out customized development according to their own needs. This scheme provides a certain reference value for enterprises and organizations to store and manage massive data.

REFERENCES

[1] Wu Zhenyu, Zhu Yingxia, Li Daotong, etc. Discussion on the design method of large-scale object storage resource pool [J]. Telecommunications Engineering Technology and Standardization, 2021 (09).

[2] Liu Suru, Chen Xinxiang, Wu Jinchao. Research on object Storage of Geographic Information Spatial data Management [J]. Geospatial Information, 2022 (03).

[3] Wang Jiazhu, Fan Zhonglei, Bi Qiang, etc. Dynamic load balancing method based on monitoring in object storage system [J]. Microelectronics and computers, 2022 (12).

[4] Yan Yi, Xu Wei. A massive moving target storage method based on Redis database [J]. Electronic Mass, 2022 (08).

[5] Lin Zhe. Design and implementation of an authentication and authorization scheme for service architecture [J]. Computer programming skills and maintenance, 2022 (11).

[6] Zeng Jia. Design of micro-service architecture based on Spring Cloud [J]. Electronic Technology, 2023552 (01): 54-55.

[7] Hujin;Jin Heo,etc. Minio A Study on Improving the Performance of Flask-based Web Server Using File Server [J].Proceedings of the Korean Society of Telecommunications. Volume , Issue . 2018.

[8] Razzaq Abdul;Ghayyur Shahbaz A. K. A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges[J] Computer Applications in Engineering Education. Volume 31 , Issue 2 . 2022. PP 421-451.