Article

Distributed Data Processing and Real-Time Query Optimization in Microservice Architecture

Jin Li [1,*]

[1] Morgan Stanley, 65 Irby Ave NW, Atlanta, GA, 30305, United States
[*] Correspondence: Jin Li, Morgan Stanley, 65 Irby Ave NW, Atlanta, GA, 30305, United States

Abstract: Today, the rapid advancement of information technology, the Internet, and big data has promoted the widespread application of microservice architecture in contemporary software engineering, especially in distributed data processing and real-time query optimization, which shows great potential for development. This article explores the optimization path of data distributed processing and real-time queries under microservice architecture. A series of real-time query performance optimization strategies have been proposed based on the characteristics of microservice architecture, such as caching mechanism for query results, optimization of indexes, monitoring and analysis of query performance, as well as the use of asynchronous processing and message queues. By adopting appropriate technology and architecture design, microservice architecture improves the operational performance and fast response to queries of distributed systems, meeting the constantly evolving needs for efficient data processing.

Keywords: microservice architecture; distributed data processing; real time query optimization; query cache; performance monitoring

Received: 02 May 2025
Revised: 08 May 2025
Accepted: 23 May 2025
Published: 27 May 2025

1. Introduction

With the continuous advancement of cloud computing and intelligent technology, distributed systems have become popular in many fields, especially in the areas of big data processing and real-time query requirements, where the growth momentum is particularly evident. Microservice architecture occupies a core position in contemporary distributed system design due to its excellent scalability, mobility, and reliability. However, the introduction of microservices has also brought new challenges in distributed data processing and real-time query optimization. How to achieve high efficiency in data storage, querying, and processing under microservice architecture has become a core issue in ensuring efficient system operation. Researchers have explored various technologies for distributed data processing under microservice architecture and proposed optimization solutions to enhance real-time query performance. This paper discusses the practical application of distributed database technology, aiming to solve the problems of low query efficiency and data consistency in distributed scenarios.

2. Core Components of Microservice Architecture

The implementation of microservice architecture relies on a series of core components and technologies, as shown in Figure 1. The registration and query of services are the core components of this architecture. Service registration nodes (such as Eureka and Consul) can complete real-time registration and search of services, ensuring dynamic updates and correct guidance of communication between services. The API portal plays an important role as a unified interface to handle front-end requests, perform tasks such as routing allocation, load balancing, security verification, and traffic control, and simplify the interaction process between the front-end and microservices. Tools such as Zuul and

Kong are typical representatives of API portals [1]. In the microservice system, configuration management hubs such as Spring Cloud Config and Consul can be used to centrally maintain configuration information for various microservices, making maintenance and updates of configuration information more convenient. Message queue technologies such as Kafka and RabbitMQ play a bridging role in asynchronous communication between microservices, achieving decoupling between services. In addition, centralized management of logs and system monitoring are also important components of microservice architecture. By using log aggregation systems (such as ELK Stack) and monitoring systems (such as Prometheus, Grafana), developers can track and troubleshoot service status in real time, ensuring stable system operation and excellent performance.

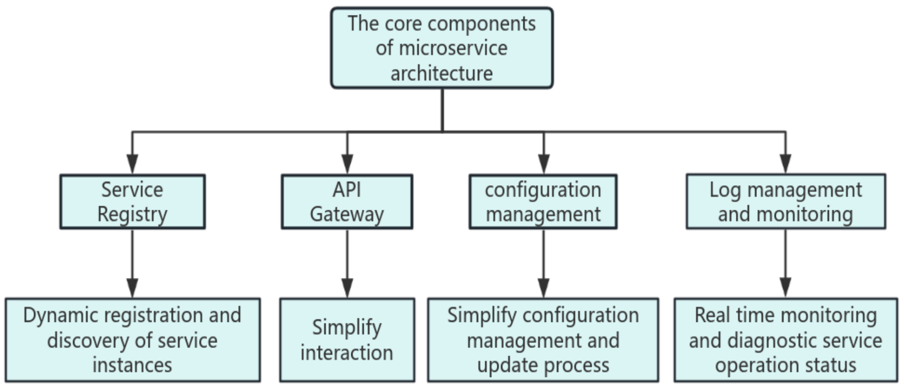
The core components of microservice architecture
Service Registry
API Gateway
configuration management
Log management and monitoring
Dynamic registration and discovery of service instances
Simplify interaction
Simplify configuration management and update process
Real time monitoring and diagnostic service operation status

Figure 1. Core Components of Microservice Management.

3. Distributed Data Processing Technology

3.1. Fundamentals of Distributed Data Processing

Distributed data processing refers to the technology of storing and processing data in a decentralized manner across multiple physical or virtual nodes. Due to the emergence of massive amounts of information, relying solely on a single processing unit is no longer sufficient for data storage, processing, and analysis. Distributed processing has become a critical coping strategy. This technology divides data into several parts to achieve storage and parallel computing across multiple nodes, enhancing the speed of data processing and system scalability [2].

In a distributed data management framework, information is subdivided into numerous data units, which are evenly distributed across numerous servers. This layout can reduce bottleneck issues in stored procedures and optimize the efficiency of storage resource utilization. The information units processed by each server maintain independence, and they must cooperate with each other through network connections to ensure data consistency and integrity. The distributed data management framework mainly consists of three basic parts: data storage layer, data processing layer, and data transmission layer. The data storage layer focuses on long-term storage of information, the data processing layer is responsible for processing information, and the data transmission layer is responsible for information transmission between servers. Hadoop, as a representative distributed data processing platform, relies on the MapReduce algorithm to achieve distributed computing of massive data. Under this framework, data is subdivided into numerous blocks and stored in HDFS. Processing tasks are dispersed across multiple nodes, each performing Map and Reduce steps. During the execution process, nodes can call local data blocks and interact with other nodes as necessary to improve data processing efficiency. Thanks to its distributed structure, Hadoop has demonstrated outstanding performance and stability in handling massive amounts of data [3].

3.2. Distributed Database Technology

Distributed database technology, as the cornerstone of distributed data processing, is committed to dispersing the storage and management functions of data across numerous physical servers, enhancing system availability, scalability, and fault recovery capabilities. In contrast, traditional relational databases often suffer from performance issues when dealing with large amounts of data due to their inherent structural and design constraints. This distributed database overcomes the performance limitations of monolithic databases by splitting data into numerous fragments (i.e. shards) and distributing these fragments across different servers.

In distributed database architecture, consistency, stability, and data persistence are achieved through a series of distributed consistency algorithms. Typical algorithms such as Paxos and Raft synchronize data between numerous replicas to address issues such as server failures. Distributed databases also propose a distributed transaction processing mechanism. Given that data is stored in various nodes, transaction processing needs to span across different database entities, which requires distributed database systems to be able to manage transaction consistency across nodes. Two stages commit protocol (2PC) and three stage commit protocol (3PC) are two commonly used distributed transaction processing protocols that ensure the coordination and consistency of all participating nodes during transaction execution through a step-by-step verification mechanism [4,5].

For example, Google's Spanner database relies on advanced technologies such as distributed protocols and global time synchronization to provide a distributed storage solution with strict consistency on a global scale. This database utilizes a distributed transaction control mechanism, Paxos algorithm, and TrueTime interface to synchronize the data status of different regions, ensuring that even in complex network distribution situations, the system can still demonstrate excellent reliability and fast data reading performance. In mathematics, the performance of distributed databases is measured by query latency and throughput. If L is the average query latency, N is the number of database nodes, and T is the throughput, then the overall query performance of the system can be expressed by the following formula:

$$L = \frac{C}{T \cdot N} \quad (1)$$

In formula (1), C is a constant related to factors such as data distribution and network transmission. This formula indicates that, with reasonable data allocation and optimized query strategies, the system's query latency can decrease as the number of nodes increases; however, excessive node expansion may introduce additional overhead that impacts latency [6].

3.3. Distributed Data Processing Framework

Distributed data processing framework is one of the core technologies for processing massive amounts of data and improving computational efficiency. This architecture achieves parallel processing of tasks across multiple nodes by partitioning data into multiple parts and storing these parts dispersed across numerous nodes, accelerating data processing speed and enhancing system scalability. Most of these architectures are based on distributed file storage systems (such as HDFS) and distributed computing paradigms (such as MapReduce, Spark), and are equipped with advanced task allocation and resource management capabilities. The core concept of distributed data processing architecture is to distribute data to numerous computing nodes, with each node independently processing a fragment of the data. Under these architectures, data processing typically involves cutting the data into smaller fragments (known as sharding) and then utilizing parallel computing paradigms to perform the processing. Each node communicates and cooperates with each other through the Internet, and integrates the processing results. Distributed architecture can effectively handle large amounts of static data and can also handle real-time data streams, demonstrating its high flexibility and versatility [7].

Apache Flink represents an advanced architecture for distributed processing of large-scale datasets, which excels in real-time data stream processing. Unlike conventional batch processing systems such as Hadoop, Flink adopts the concept of stream processing, enabling it to process and provide real-time feedback on real-time data streams. In the Flink architecture, data is transmitted as independent events, and each processing node independently processes the data stream it receives and maintains data synchronization through traffic management mechanisms. Flink is known for its low latency and excellent throughput, and has been widely used in industries such as finance and supply chain. Especially on e-commerce platforms, Flink can analyze consumer behavior data and recommend products or adjust pricing strategies to businesses in real-time. Compared with traditional batch processing systems, Flink can complete data processing tasks and output analysis results in hundreds of milliseconds, improving the response speed and accuracy of business decisions on e-commerce platforms.

4. Real Time Query Optimization under Microservice Architecture

4.1. Query Cache and Index Optimization

In microservice architecture, optimizing query performance is particularly crucial as the amount of information expands and the frequency of service calls increases. Query caching and index optimization are key means to improve database query efficiency, reducing database burden and accelerating response speed.

By temporarily storing frequently accessed information in memory, query caching can reduce duplicate queries to the database and lower access latency. In microservice systems, this type of cache is typically deployed on distributed cache servers (such as Redis, Memcached). Faced with frequent queries, the system can directly read data from the cache to avoid querying the database every time. By building indexes on database columns, data retrieval efficiency can be improved. Among the types of indexes, there are single field indexes, multi field indexes, and full-text search indexes. Multi field indexing is particularly effective for retrieval operations involving multiple conditions, as it can integrate different fields into one index and reduce the data scope of database retrieval. Under the microservice architecture, databases are usually deployed in various service nodes, and proper index construction can enhance data retrieval efficiency across service nodes and reduce response time [8].

Combining query caching with index optimization can further improve the efficiency of data retrieval. Once the query cache can match the request, the data will be directly read from the cache. When the cache does not match, index optimization is used to execute the query. By appropriate caching and indexing settings, query response speed and resource utilization can be balanced. For example, on an e-commerce platform, when users search for product information, the system first searches for popular product information in the cache. If there is no relevant data in the cache, the system will use multi field indexes to efficiently query product categories and prices. Set the query time as:

$$Tq = \frac{Nd}{C} + Iq \tag{2}$$

In formula (2), Tq represents the query time, Nd is the amount of data to be scanned, C is the cache hit rate, and Iq is the index query time. Optimizing cache and indexing can reduce query time and improve system performance.

4.2. Query Performance Monitoring and Analysis

In microservice systems, query performance is one of the key factors affecting system response speed and user experience. With service decomposition and data distribution, a query may need to cross numerous microservices and databases. Real-time monitoring and in-depth analysis of query performance have become core to ensure system stability and responsiveness. Monitoring and analyzing query performance helps developers identify system performance bottlenecks and provides decision-making basis for further optimization of the system.

Monitoring query performance typically involves evaluating indicators from multiple dimensions, such as query response time, database workload, cache efficiency, and probability of query failure. These indicators enable developers to instantly grasp performance barriers during query execution and quickly identify potential performance risks. For example, if the query response time is too long, it may indicate that the database index needs to be optimized or cache failure leads to frequent direct queries. If the database pressure is too high, it may be due to overly dense database interactions between services, which may require optimizing query logic or improving load balancing efficiency.

In order to achieve real-time monitoring of query efficiency, commonly adopted methods include deep log analysis, distributed tracing techniques (such as Jaeger, Zipkin, etc.), and performance monitoring systems (such as Prometheus, Grafana, etc.). This type of tool can track the process of requests from beginning to end, meticulously record the feedback time and processing path of each query, and generate intuitive query efficiency analysis charts. With the help of these monitoring data, the R&D team has the ability to identify core issues that affect query efficiency and take targeted measures to improve them. In a certain e-commerce platform, the query performance monitoring Table 1 is shown below, which displays the response time and database load of different query types:

Table 1. Query Performance Monitoring Table.

Query type	Average response time (ms)	Cache hit rate(%)	Database load(%)	Error rate(%)
User Information Inquiry	120	95	30	0.5
Product search query	200	80	50	1.0
Order details inquiry	150	85	45	0.3
Payment information inquiry	250	70	60	1.5

By analyzing the data in Table 1, the development team can take measures such as optimizing indexes, increasing cache, or performing query sharding to improve query efficiency. Through continuous query performance monitoring and data analysis, query optimization under microservice architecture can be iteratively improved, helping the system achieve higher performance and better user experience.

4.3. Asynchronous Processing and Message Queuing

In microservice systems, the performance of real-time retrieval is often limited by the constraints of database queries and synchronous service calls. Introducing asynchronous processing and message queue mechanisms is a feasible strategy to improve efficiency in order to accelerate system response time and enhance processing capabilities. This strategy enables the system to delay time-consuming operations and immediately return responses to users upon receiving their requests. After asynchronous processing is completed, the system will inform the user of the result through callback functions or notification mechanisms. This approach reduces the coupling between services and enhances the scalability and flexibility of the system. Asynchronous processing can reduce response latency when handling complex queries and large-scale data processing tasks. Message queue technology (such as Kafka, RabbitMQ, etc.) is the mainstream means of implementing asynchronous processing. In the application of microservice architecture, message queues adopt a publish-subscribe model to achieve low coupling communication between services. The tasks are transmitted to the corresponding consumer services in the form of messages, which are processed asynchronously by the consumers, improving the scalability and robustness of the entire system. On an e-commerce platform, whenever a consumer requests a payment instruction, the payment process requires collaboration with multiple microservices such as inventory control, order processing, and payment interfaces. With the help of message queue mechanism, the system is able to asynchronously transmit payment instructions to relevant microservices without waiting for their response directly. Once the payment transaction is completed, the relevant modules will asynchronously refresh inventory data, update order status, and send notifications to other services through a message queue. This design ensures that the processing delay of a single microservice does not affect the progress of the entire consumer request, improving the system's processing capacity and response efficiency. The request processing time of the system consists of synchronous processing time T and asynchronous processing time Ta, and the overall performance of the system can be expressed by the following formula:

$$Ttotal = \frac{N}{C} + Ts + \frac{Na}{Ca} + Ta \quad (3)$$

In formula (3), N is the total number of requests, C is the cache hit rate, Ts is the synchronous operation time, Na is the number of asynchronous tasks, Ca is the processing capacity of asynchronous tasks, and Ta is the processing time of each asynchronous task. By reasonably designing the configuration of asynchronous processing and message queues, the delay caused by synchronous operations can be reduced, and the overall response speed and processing capability of the system can be improved.

5. Conclusion

In the contemporary field of software engineering, microservice architecture, as a critical design framework, has been widely promoted to various large-scale distributed systems. It has become the preferred solution for enterprises to achieve the digital transformation process with its high flexibility and convenient scalability. With the advancement of cloud technology and big data, the distributed storage and processing capabilities of data have been enhanced, providing strong support for processing large-scale data in microservice architectures. Unlike traditional single architecture, in microservice architecture, each service unit is often equipped with its own data storage method. Therefore, when designing a data processing system, it is necessary to focus on data consistency, stability, and scalability. By utilizing distributed database technology, we can effectively address issues such as data redundancy and performance limitations, and improve the overall operational efficiency of the system.

References

1. H. Raza, W. Abbasi, K. Aurangzeb, N. M. Khan, M. S. Anwar, and M. Alhussein, "Parameter estimation of the systems with irregularly missing data by using sequentially parallel distributed adaptive signal processing architecture," *Alexandria Eng. J.*, vol. 82, pp. 139–144, 2023, doi: 10.1016/j.aej.2023.09.051.
2. R. Chen, G. Cai, J. Chen, and Y. Hong, "Integrated method for distributed processing of large XML data," *Cluster Comput.*, vol. 27, no. 2, pp. 1375–1399, 2024, doi: 10.1007/s10586-023-04010-0.
3. M. A. Poltavtseva and V. A. Torgov, "Applying distributed ledger technology to auditing and incident investigation in big data processing systems," *Autom. Control Comput. Sci.*, vol. 56, no. 8, pp. 874–882, 2022, doi: 10.3103/S0146411622080193.
4. A. Alexandrescu, "Parallel processing of sensor data in a distributed rules engine environment through clustering and data flow reconfiguration," *Sensors*, vol. 23, no. 3, p. 1543, 2023, doi: 10.3390/s23031543.
5. S. Cui, "Online education based on distributed multi-layer data processing technology," *Procedia Comput. Sci.*, vol. 228, pp. 688–700, 2023, doi: 10.1016/j.procs.2023.11.080.
6. H. Miyajima, N. Shigei, H. Miyajima, and N. Shiratori, "Scalability improvement of simplified, secure distributed processing with decomposition data," *Nonlinear Theory Its Appl., IEICE*, vol. 14, no. 2, pp. 140–151, 2023, doi: 10.1587/nolta.14.140.
7. L. Wang, B. Yu, F. Chen, C. Li, B. Li, and N. Wang, "A cluster-based partition method of remote sensing data for efficient distributed image processing," *Remote Sens.*, vol. 14, no. 19, p. 4964, 2022, doi: 10.3390/rs14194964.
8. E. Fakiris, G. Papatheodorou, D. Christodoulou, Z. Roumelioti, E. Sokos, M. Geraga, et al., "Using distributed temperature sensing for long-term monitoring of pockmark activity in the Gulf of Patras (Greece): Data processing hints and preliminary findings," *Sensors*, vol. 23, no. 20, p. 8520, 2023, doi: 10.3390/s23208520.