Software Architecture Evolution and Technology Research

Ruiqi Zeng(✉), Yiru Niu, Yue Zhao, and Haiyang Peng

Science and Technology on Communication Security Laboratory, No. 30 Research Institute of China Electronics Technology Group Corporation, China Electronics Technology Cyber Security Co., Ltd., Chengdu, China
zengruiqi@sina.com, physea@mail.ustc.edu.cn

Abstract. In order to solve the problem of the complexity of software system and the efficiency of large-scale collaboration, software architecture design methods have gradually developed. In the process of software architecture evolution, users and concurrency are increasing day by day, and software technology is constantly evolving and improving accordingly. Although there are many researches and technical discussions on software architecture methodology, there are few papers that fully discuss the development of technology in terms of the evolution of the architecture. This paper takes the evolution of software architecture as the research object, discusses problems, solutions and related technologies involved in the evolution of software architecture, and comprehensively explains how the system gradually evolves from a single architecture to a complex high-concurrency architecture.

Keywords: Architecture design · Architecture evolution · High concurrency

1 Introduction

With the rapid development of computer science and programming, software design can be applied to everything, from daily life to aerospace. The development method has also evolved from a single-person development model to a large-scale collaborative engineering model.

Various uncertainties in management and technology have increased explosively with the explosive growth of large-scale collaboration efficiency and software complexity. As a result, the quality of software development cannot be effectively guaranteed, and the cycle and cost cannot be effectively controlled.

People have been seeking to find solutions to these problems. However, in the software engineering bible "The Mythical Man-Month" published in 1975, Fred Brooks said that there is no silver bullet that can solve all problems1.

Since then, people have developed project research and development process management to control the uncertainty of management activities, and also developed software architecture design methods to control technical uncertainty. At the same time,

S. Liu and X. Ma (Eds.): ADHIP 2021, LNICST 416, pp. 708–720, 2022.
https://doi.org/10.1007/978-3-030-94551-0_54

they have continuously summarized and improved in practice, to effectively guide and to the greatest extent guarantee the quality, cycle and cost of software development.

In terms of software architecture design, software architecture is often evolving. With the increasing number of users and concurrency, the actual software system will also experience a process from simple in the early stage to complex in the later stage. In the process of software architecture expansion, various problems will arise, and corresponding technologies to solve these problems will emerge accordingly. The emergence of these technologies ensures the availability, stability, scalability and concurrency of the software during the evolution of the system2.

This paper explains problems that may be encountered in software evolution, such as how to provide high concurrent access when resources are limited, how to quickly query database tables, how to reduce user access latency, how to improve analysis efficiency for unstructured data, how to carry out rapid expansion of the system, how to carry out rapid upgrade and deployment of the system, how to carry out all-round monitoring of the system and so on. For these problems, discussed how to solve them, the related technologies used, and explained the effects of these technologies in solving practical problems.

The rest of this paper is organized as follows. Section 2 introduces the evolution of architecture and technologies involved. Section 3 gives a general overview of a complex architecture. Section 4 is a summary.

2 Architecture Evolution Technology

When the initial user and concurrency were very small, the simplest software architecture was often used. With the increase of users and concurrency, system performance encountered bottlenecks. At this time, various technologies need to be used to expand and enhance various parts of the architecture. Then the system became more and more complex. This is the software architecture evolution.

2.1 Stand Alone Deployment

In the early stage of system construction, there were few concurrent visits and there were no requirements for performance. At this time, the application system and the database are often placed on the same server. The entire deployment structure is a stand-alone deployment, as shown in (Fig. 1).

The problem with stand-alone deployment is that the application system and the database of the same server will compete for the resources. When the number of visits increases, the server will become a performance bottleneck, so architecture evolution is required.

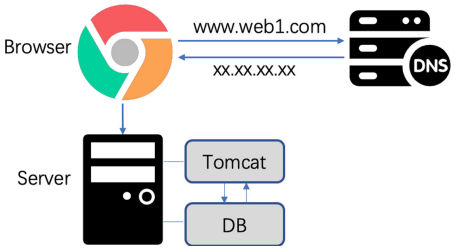

Fig. 1. Stand alone deployment

2.2 Separation of Application and Database

In order to cope with the lack of performance of the single-machine deployment server, the application system and the database are usually placed on two different servers, which can significantly improve their performance. The separated architecture is shown in (Fig. 2).

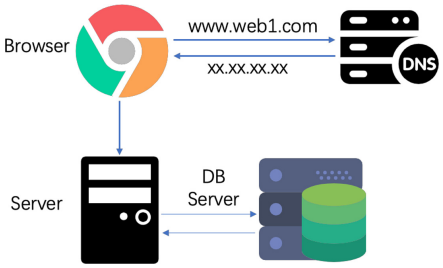

Fig. 2. Separation of application and database

When concurrency continues to grow, the read and write capabilities of the database will become a bottleneck, requiring continued architectural adjustments.

2.3 Introducing Local Cache and Distributed Cache

In order to solve database performance problems, caching technology is introduced, which includes adding a local cache on the application server side and adding a distributed cache externally3. The cache is mainly used to store popular html pages or frequently used data. Through caching technology, most requests can be intercepted before they reach the database, which greatly improves efficiency. The architecture at this time is shown in (Fig. 3).

Although cache can handle most of the requests, with the growth of user access, the stand-alone server, where the application is deployed, will under increasing pressure of concurrency, resulting in slower response, so the architecture will continue to evolve.

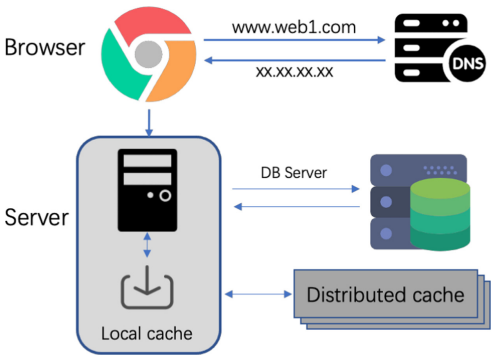

Fig. 3. Introducing local cache and distributed cache

2.4 Introducing Reverse Proxy to Realize Load Balancing

When the application pressure is too great, multiple instances of the application can be deployed, and reverse proxy software can be used for load balancing to reduce the pressure on each server and improve the comprehensive concurrency capabilities4. Generally, you can use Nginx as a load balancer. The evolved architecture is shown in (Fig. 4).

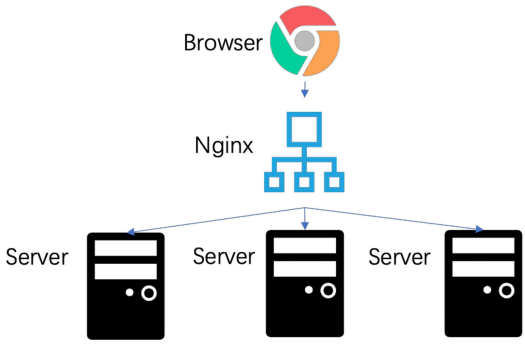

Fig. 4. Introducing reverse proxy to realize load balancing

Although the load balancer greatly increases the amount of concurrency, more concurrency means more database access, and the database deployed on a single machine becomes the bottleneck of system performance.

2.5 Database Read/Write Separation

In order to reduce the pressure on the database, the database is divided and multiple databases are used for business processing. It is mainly divided into reading DB and writing DB. The reading DB is used when reading data, and the writing DB is used when writing data5. There can be multiple reading DB, and synchronization mechanism is used to synchronize data from writing DB to reading DB. The latest written data that

needs to be queried can be written to the cache, and obtained through the cache. The architecture is shown in (Fig. 5).

There are some specialized middleware technologies that can support the separation of database read and write, and the client can access the lower-level database through the middleware. After reading and writing are separated, problems such as data synchronization need to be solved.

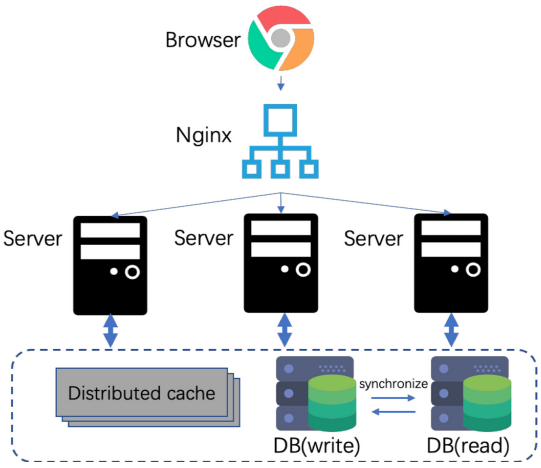

Fig. 5. Database read write separation

When the business continues to increase gradually, there will be a big gap in the number of visits between different businesses. Different businesses directly compete for the database, which will affect each other's performance, so the architecture needs to be upgraded again.

2.6 Database Divided by Business

In order to solve the problem of competition between different businesses on database, database is divided by business, so that data of different businesses are stored in different databases. Resource usage is isolated according to business, and competition for resources is reduced. For businesses with high traffic, more servers can be deployed to support them. The architecture is shown in 0. This architecture needs to solve problem of correlation analysis across DB (Fig. 6).

Under this architecture, as number of users grows, the stand-alone writing DB will gradually reach performance bottleneck.

2.7 Split a Large Table into Small Tables

As the amount of data in written database becomes larger and larger, the query efficiency will become lower and lower. In order to improve the efficiency of a single table, it is

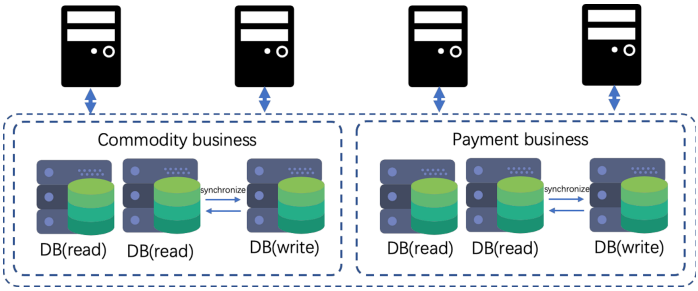

Fig. 6. Database is divided by business

necessary to split the tables. Sub-table is to disperse data of one table into multiple tables to increase the efficiency of data query and operation efficiency.

As long as the amount of table data for real-time operation is small enough and request can be distributed to these small tables on multiple database servers as evenly as possible, the database performance can be improved through horizontal expansion. At this time, the system architecture is shown in (Fig. 7).

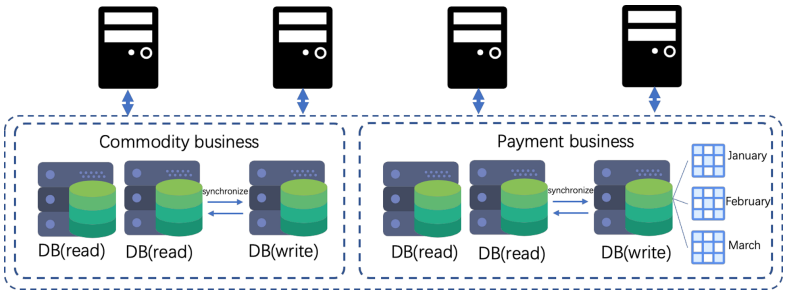

Fig. 7. Split a large table into small tables

Under this architecture, both the database and application system can be scaled horizontally, and the supportable concurrency is greatly improved. However, as the number of users continues to grow, a single-machine load balancer will eventually become a performance bottleneck.

2.8 Use LVS or F5

Since the bottleneck is in Nginx, it is impossible to achieve performance improvement by adding multiple layers of Nginx, so a more powerful load balancing technology is required6.

LVS and F5 are load balancing solutions that work at the fourth layer of network. LVS is software, running in kernel state of operating system, and can forward TCP requests or higher-level network protocol requests, so it supports more protocols, and its performance is also much higher than Nginx. F5 is a load balancing hardware, similar to

the capabilities provided by LVS, with higher performance than LVS, but it is expensive. The enhanced system architecture is shown in (Fig. 8).

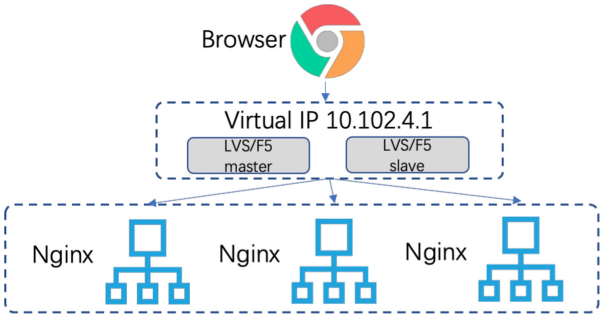

Fig. 8. Use LVS or F5

The number of users continues to grow, reaching tens of millions or even hundreds of millions. Because users are distributed in different regions and their distance from server room is different, the delay for users to access server will also be significantly different, which causing the response to some users is too slow, so the architecture still needs to evolve.

2.9 Load Balancing of Data Centers by DNS Polling

By distributing business requests to different data centers or computer rooms, you can continue to increase concurrent visits.

In DNS server, one domain name can be configured to multiple IP addresses, and each IP address corresponds to a virtual IP in different computer rooms. This method can realize load balance between computer rooms, the system can achieve horizontal expansion at computer room level, and concurrency of tens of millions to hundreds of millions can be solved by adding computer room7. The architecture is shown in (Fig. 9).

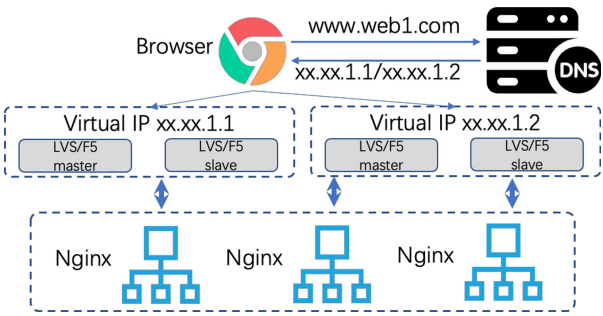

Fig. 9. Load balancing of data centers by DNS polling

With enrichment of data and development of business, the requirements for retrieval and analysis become more and more abundant, new needs require new technologies.

2.10 Introduce NoSQL Database and Search Engine Technology

For massive file storage, it can be solved by distributed file system HDFS. For key value data, it can be solved by HBase and Redis8. For full-text retrieval scenarios, it can be solved by search engines such as ElasticSearch. For multi-dimensional analysis scenarios, it can be solved by Kylin or Druid9. The enhanced architecture is shown in (Fig. 10).

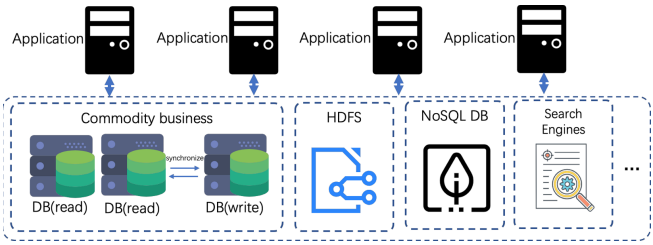

Fig. 10. Introduce NoSQL database and search engine technology

Abundant technologies have solved a wealth of needs, and the business dimensions can be greatly expanded. As a result, one application contains too much business code, so its business upgrade and iteration become difficult.

2.11 Big Application Split into Small Application

In order to solve the difficult upgrade and iteration problem caused by a large application, application code is divided according to business sectors, so that the responsibilities of a single application are clearer, the maintenance cost is lower, and each module can be upgraded and iterated independently10. The architecture after application split is shown in (Fig. 11).

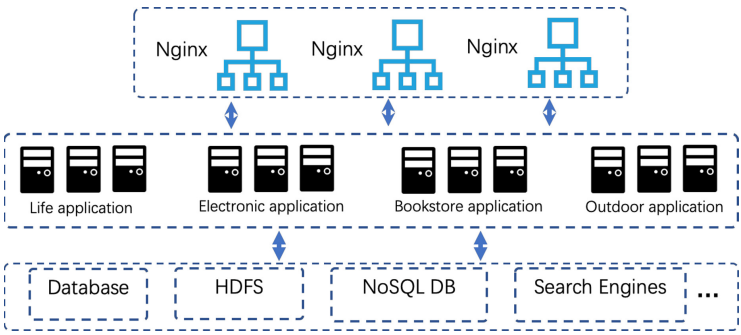

Fig. 11. Big application split into small application

After application is split, there are shared modules between different applications, and separate management of each application will cause the same code to be redundant, which lead to that when a public function need upgraded, all application code need upgraded. To improve reusability, the common code needs to be stripped out.

2.12 Separating Public Functions into Micro Services

In order to solve the problem of reuse and efficiency of public functions, these services need to be deployed separately. For functions that are used by multiple applications such as user management, orders, payment, and authentication, their codes are extracted to form a single service, which is made into a microservice mode11. Applications and services access public services through multiple methods such as HTTP, TCP, or RPC requests, and each individual service can be managed by a separate team (Fig. 12).

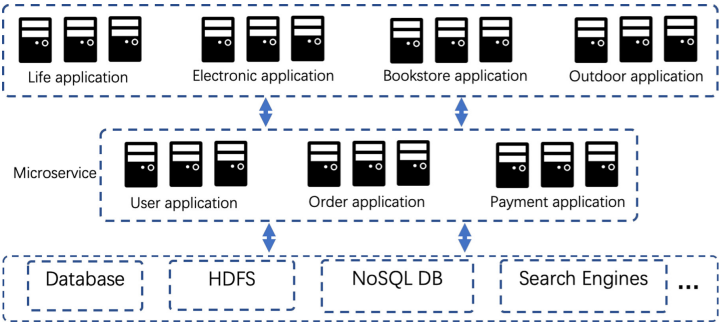

Fig. 12. Separating public functions into micro services

Although microservices improve service stability and availability, when applications and services access each other, their call chains will become very complicated and logic will become chaotic, so new technologies are needed to solve this problem.

2.13 Introduce ESB to Shield Access Difference of Service Interface

In order to improve efficiency of call chain between services, ESB is used to perform unified access protocol conversion, applications and services access and call each other through ESB, so as to reduce degree of system coupling12. The ESB technical architecture is shown in (Fig. 13).

When the number of applications and services continues to increase, the deployment of applications and services becomes complicated, and the number of services deployed on the same server increases sharply, and service operating environment is prone to conflicts. In addition, for scenarios that require dynamic expansion and contraction, the performance of service needs to be scaled horizontally, which makes operation and maintenance very difficult.

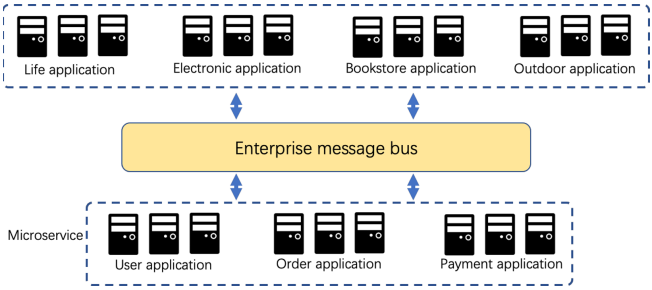

Fig. 13. Introduce ESB to shield access difference of service interface

2.14 Introduction of Containerization Technology

In order to solve operation, maintenance and deployment problems of large-scale microservice systems, containerization technology is needed.

Containerization technology can make application deployment fast and efficient, and the deployment environment can be reused13. At the same time, container orchestration technology provides a set of mechanisms for deploying, maintaining, expanding and repairing containerized microservices, making container management very efficient. At present, the most popular containerization technology is Docker, and the most popular container management tool is Kubernetes14. Applications/services can be packaged as Docker images, and images can be dynamically distributed and deployed through Kubernetes15. The containerized deployment architecture is shown in (Fig. 14).

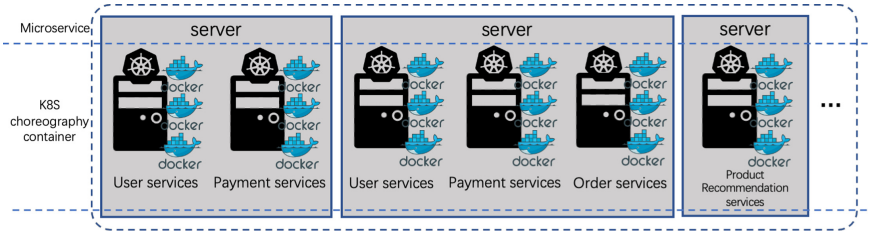

Fig. 14. Introduction of containerization Technology

Containerized deployment solves the problem of service dynamic expansion and contraction, and greatly improves operation and maintenance efficiency. However, hardware resources need to be managed, the costs of machine itself, operation and maintenance are extremely high, and resource utilization is low.

2.15 Using Cloud Platform System

In order to solve the problem of high cost of physical resources and dynamic expansion of hardware resource, the system can be deployed on a public cloud to use massive public machine resources.

On cloud platform, hardware resources (such as CPU, memory, network, etc.) can be dynamically applied for on demand, a general operating system, common technical components (such as Hadoop technology stack, MPP database, etc.) and even well-developed applications are provided for users.16. The cloud platform architecture is shown in (Fig. 15).

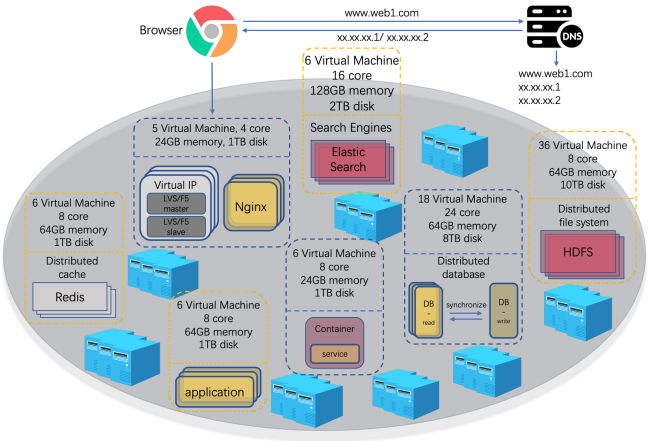

Fig. 15. Using cloud platform system

Since then, the entire technical architecture has solutions from the problem of high concurrent access to service architecture and system implementation.

3 Global View

0is a global view of architecture evolution. In actual application, the architecture design of a system with clear performance indicators needs to be able to support the performance of system, while leaving interfaces for architecture extending.

In our system design, we adopted front-end and back-end separation technology, local cache technology, business database separation, database read-write separation technology, load balancing technology, NoSQL database, microservice-related technology, and containerization technology. Through the use of these technologies, the high concurrency requirements of our system have been met, and our system can provide 10,000 concurrent access for outside system (Fig. 16).

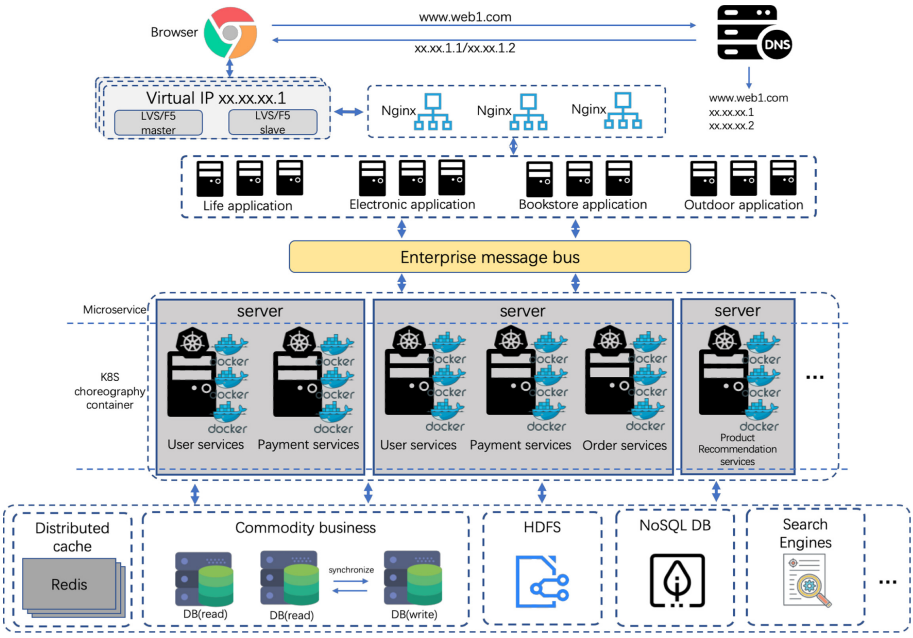

Fig. 16. Global view

4 Conclusion and Outlook

This paper discusses the technologies involved in the evolution of the architecture. Firstly, we explain why the software architecture methodology emerged, then we illustrate problems, methods, and technologies involved in software architecture evolution by describing the relatively complete process of architectural evolution. Finally, a comprehensive display and summary of software architecture evolution are presented. This paper covers all aspects of architecture evolution, discusses the usage scenarios and effects of various technologies, and provides a comprehensive reference for high-concurrency software design.

In future work, we will continue to pay attention to the development of related technologies. For example, the evolution of microservice technology to service mesh, the containerized deployment of heterogeneous systems, etc., and continue to discuss the application scenarios of related technologies and their promotion of architecture evolution.

Acknowledgment. Foundation Item: Supported by Sichuan Science and Technology Program (No. 2020YFG0292).

References

1. Brooks, F.: The Mythical Man-Month, 20th Anniversary Edition (1995)
2. Abbas, G., Imran, M., Hafeez, Y., et al.: Improving software architecture design decision by selecting set of solutions. In: 2020 3rd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET) (2020)
3. Jing, Z., Wu, G., Hu, X., et al.: A distributed cache for hadoop distributed file system in real-time cloud services. IEEE (2012)
4. Moharir, M., Shobha, G., Oppiliappan, A., et al.: A study and comparison of various types of load balancers. In: 2020 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE). IEEE (2020)
5. Xue-jun, LI, Qian-jun, et al.: Application and Research of Read-write Separation Technology in Active-active Database
6. Randhawa, N.S., Dhami, M., Singh, P.: Enhanced load balancer with multi-layer processing architecture for heavy load over cloud network. Department of Electronics and Communication Engineering, Swami Vivekanand Institute of Engineering & Technology, Banur, India; Department of Information Technology, Chandigarh Engineering College, Landran, Mohali, India; Department of Information Technolog
7. Hong, Y.S., No, J.H., Kim, S.Y.: DNS-based load balancing in distributed web-server systems. In: IEEE Workshop on Software Technologies for Future Embedded & Ubiquitous Systems, & the Second International Workshop on Collaborative Computing, Integration. IEEE (2006)
8. Cattell, R.: Scalable SQL and NoSQL data stores. ACM SIGMOD Rec. **39**(4), 12–27 (2010)
9. Daniel, B.K.: Big Data and data science: a critical review of issues for educational research. British J. Educ. Technol. **50**(1) (2019)
10. Bernstein, D.: Containers and cloud: from LXC to docker to kubernetes. IEEE Cloud Comput **1**(3), 81–84 (2014)
11. Nadareishvili, I., Mitra, R., Mclarty, M., et al.: Microservice Architecture (2016)
12. Hua, J.X., Xia, Q.U.: Application Research on the Enterprise Service Bus Technology in Application System Integration of Universities. Journal of Xi'an University (Natural Science Edition) (2019)
13. Develop with Docker [EB/OL] [2020 - 09 - 10]. https://docs.docker.com/develop/
14. Orzechowski, M., Balis, B., Pawlik, K., Pawlik, M., Malawski, M.: Transparent deployment of scientific workflows across clouds - kubernetes approach. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 9–10 (2018). https://doi.org/10.1109/UCC-Companion.2018.00020
15. Ruiqi, Z.Y., Zhao, H., et al.: A novel construction technology of enterprise business deployment architecture based on containerized microservices (2020)
16. Sugam, S., Chang, V., Sunday, T.U., et al.: Cloud and IoT-based emerging services systems. Cluster Comput. **22**, 1–21 (2019)