American Research Journal of Computer Science and Information Technology

Volume 7, Issue 1, 68-72 Pages
Research Article | Open Access
ISSN (Online)- 2572-2921
DOI : 10.21694/2572-2921.24015

AMERICAN RESEARCH JOURNALS
An Academic Publishing House

Analysis and Design of High-Load Systems Based on Microservice Architecture

Denis Shelmanov
Frontend Engineer, INUI Gaming, Serbia, Belgrade.

ABSTRACT

Within the framework of this work, methods of designing highly loaded systems based on a microservice architecture were studied, which allows the creation of flexible and scalable IT infrastructures to ensure resistance to intense loads. By splitting into independent modules, the microservice architecture improves reliability and reduces the dependency between services.

The main objectives of the research include the development of a methodology for system decomposition, optimizing resource use, and ensuring fault tolerance, which is achieved through the use of Event Sourcing and CQRS patterns that provide state management and asynchronous event processing. Additionally, to achieve high adaptability and scalability, containerization, and orchestration of services through Kubernetes are used, which allows for dynamic load distribution and simplifies service management in conditions of intensive operation. The results of the analysis confirmed that the implementation of a microservice architecture minimizes system response time, improves data consistency, and reduces the risk of failures.

In turn, the findings demonstrate that the microservice approach to the design of high-load systems provides flexibility, independent scaling, and adaptability, which is critically important for modern information platforms focused on continuous operation and prompt response to changes in load.

KEYWORDS: *microservice architecture, high-load systems, fault tolerance, containerization, Kubernetes, Event Sourcing, CQRS, scalability, independent deployment.*

INTRODUCTION

High-load systems represent complex architectural solutions essential for supporting large information platforms such as social networks, online stores, and data streaming services. As the number of users, data volume, and requests increase, the system load intensifies, requiring high resilience and flexibility. Traditional monolithic systems in these conditions often encounter scalability and fault-tolerance issues, making the transition to more flexible architectural approaches, such as microservices architecture, a necessary step to ensure reliable performance.

The relevance of this research topic is driven by the need to develop resilient and flexible systems for large platforms operating under high loads. The shift to microservices architecture not only addresses scalability and resilience challenges but also optimizes resource management processes. In the context of continually increasing demands for performance and availability, automation and service orchestration methods, such as containerization and the use of Kubernetes, are particularly important. These technologies enable dynamic scaling, which is crucial for applications operating under conditions of unpredictable load growth.

The aim is to analyze and develop approaches to designing high-load systems based on microservices architecture.

MATERIALS AND METHODS

To conduct a comprehensive study of this topic, literature analysis, data systematization, and a comparative method were employed, along with an examination of companies' practical experiences.

Microservices architecture (MSA) has become a key approach in developing complex and scalable systems due to its flexibility and ability to simplify the management of large codebases [1]. Desai V., Koladia Y., and Pansambal S. [1] highlight the advantages of MSA over monolithic systems, noting improved scalability, flexibility, and resilience to changes. However, they emphasize the need for careful management and technology selection for the successful implementation of microservices.

The quality attributes of MSA were extensively studied in a systematic review by Li S. et al. [2]. The authors found that, despite improving system scalability and maintainability, MSA introduces new complexities in the areas of security, testing, and monitoring. This necessitates the development of new methods and tools for effective management of these aspects.

Practical aspects of MSA implementation are discussed in an online resource [3], which covers design patterns and best practices. It emphasizes that the correct application of architectural patterns and adherence to development principles are critical for the successful adoption of MSA and realizing its benefits [3].

Ao B. [4] proposes an integrated service system for multipurpose business scenarios based on MSA. The author concludes that the use of microservices in multicomponent energy systems enhances management efficiency and flexibility in adapting to various business processes.

In the context of high-load systems, Kornuta V. et al. [6] demonstrate the application of MSA for information systems using the example of the MedicinePlanner service. The main conclusion is that microservices improve the scalability and reliability of the system, which is especially important for handling large volumes of data in the medical field [6].

Load balancing in MSA is a critical task addressed in the work by Cao and co-authors [7]. They proposed an algorithm for an API gateway in a microservices architecture designed for smart cities. The results show that the developed algorithm enhances request distribution efficiency and optimizes resource usage, contributing to the stable operation of urban services [7].

Rath C. K., Mandal A. K., and Sarkar A. [8] explore a scalable microservices-based Internet of Things (IoT) architecture to ensure device compatibility. They conclude that the microservices approach provides modularity and flexibility for IoT systems, facilitating the integration of various devices and technologies [8].

Despite its advantages, MSA faces challenges in event management and ensuring system consistency. Laigner R. et al. [9] conducted an empirical study identifying the main difficulties associated with event handling and proposed solutions to improve system reliability and consistency.

The optimization of cloud systems based on MSA is examined in the work of Daradkeh T., and Agarwal A. [10]. They present a model and methods for optimizing a cloud-based elastic management system founded on microservices. The main conclusion is that MSA enhances elasticity and efficient resource management in cloud environments [10].

Design patterns such as event sourcing and CQRS are applied in the middleware model proposed by Youssef M. et al. [11]. This approach improves the scalability and manageability of complex distributed systems based on a multi-micro-agent system [11].

A comparison of MSA with serverless architecture is conducted in the study by Fan C. F., Jindal A., and Gerndt M. [12]. They find that MSA provides better performance for complex, long-running applications, while serverless architecture is effective for simple and event-driven tasks [12].

The application of microservices-based recommendation systems is demonstrated through examples. The article [13] provides a detailed analysis of Spotify's recommendation system. The main conclusion is that Spotify combines collaborative filtering, natural language processing, and audio analysis to create personalized recommendations, enhancing user experience [13]. Similarly, [14] examines LinkedIn's job recommendation system. The platform applies machine learning algorithms to personalize job suggestions based on users' professional skills, experience, and interests, increasing relevance and search efficiency [14].

The next section will examine the practical aspects of this topic in detail.

RESULTS AND DISCUSSION

Microservices architecture enables small development teams to select the optimal technology stack best suited for each service. The rapid development of technology and frequent updates to development tools allow for quicker adoption of new, more efficient solutions, which is particularly relevant in microservices architecture [3]. Microservices architecture adheres to a set of principles, such as high cohesion, low coupling, separation into business-oriented services, flexibility, and reusability. For example, MSA utilizes modularity, allowing components to be easily adapted to changes. Security requirements, including operating system protection, secure data management, and adherence to information security measures, are also essential for enhancing system reliability and resilience.

This approach is widely adopted by leading companies to support scalable, resilient, and adaptive systems in high-demand environments. Examples of organizations utilizing microservices architecture include:

Amazon – The AWS platform offers services based on microservices architecture, allowing the company to provide users with flexible and scalable solutions. AWS microservices enable Amazon to deploy new features faster while maintaining high performance for millions of users [12].

Spotify – The music streaming service uses microservices to manage recommendations, process user data, and create playlists. The microservices approach allows Spotify to quickly update recommendation algorithms and deliver current playlists to users without affecting other functions [13].

LinkedIn – LinkedIn uses microservices to manage social graphs, job recommendations, and user profiles. This approach helps the company ensure high availability and reliability while working with millions of users, which is especially important when handling large volumes of data in real time [14].

These cases demonstrate the benefits of microservices in terms of performance, adaptability, and resilience. Microservices provide companies with the flexibility to scale and maintain fault tolerance, enabling rapid responses to shifting market demands and evolving user needs.

In addition to these strategic advantages, the flexible development framework within microservices architecture (MSDH) further enhances system quality and maintainability. By implementing rigorous design criteria, such as framework cleanliness, consistency, component reuse, and the independence of business logic from underlying systems, MSDH improves both the development process and subsequent system maintenance. This setup allows individual specialists or small teams to develop specific functions independently, supporting flexible task distribution and efficient project scaling.

Figure 1 illustrates how MSDH simplifies the addition of new functions, facilitates future maintenance, reduces complexity during scaling, and improves module interaction through APIs, providing more efficient architectural development [4].

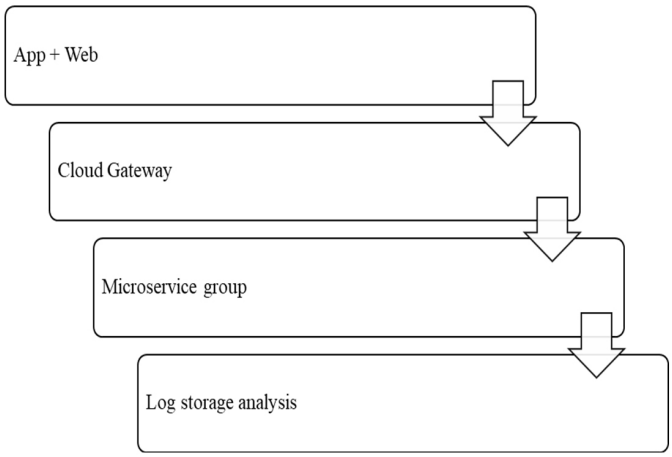

Fig.1. Basic architecture of MSHD microservices [4]

However, choosing patterns requires consideration of numerous factors, from throughput to fault tolerance. The use of asynchronous message transmission systems, such as Apache Kafka or RabbitMQ, allows for load balancing and offloading of core services, reducing their reactivity and thereby maintaining resilience under increased requests [5].

The use of container technologies like Docker and orchestration with Kubernetes greatly simplifies this process, enabling real-time deployment of additional microservice instances. Kubernetes automatically manages resource allocation, ensuring service continuity and minimizing downtime. This approach enhances the system's fault tolerance, while the autonomy of each microservice minimizes the need for scaling infrastructure nodes not involved in data processing [6].

Integrating a large number of independent services requires specialized solutions for managing routing, load distribution, and data encryption. In this context, service meshes become essential tools. Platforms like Istio or Linkerd provide comprehensive tools for automating interservice communication, facilitating security policy compliance, load balancing, and routing. Additionally, service meshes support dynamic request routing management, ensuring quality of service under high traffic volumes [7].

Under high-load conditions, systems must maintain consistent states even in the event of failures. The use of the Event Sourcing architectural pattern allows for tracking events and restoring system state in case of unexpected situations. Another key technique is Command Query Responsibility Segregation (CQRS), which enables the separation of data update and read processes, improving request handling efficiency. Data storage systems must account for consistency type—strong or eventual—depending on the specific requirements of each microservice [8].

Asynchronous interaction between microservices using Apache Kafka is one of the popular methods for load distribution in high-load systems. For example, Uber actively uses Kafka for asynchronous messaging between its microservices, providing high speed and reliability in data transmission. Kafka allows Uber to manage real-time events, such as driver location tracking and route optimization. The following code example demonstrates how to publish a message in Kafka using the Kafka-python library:

```
from kafka import KafkaProducer
import json
producer = KafkaProducer(
   bootstrap_servers=['localhost:9092'],
   value_serializer=lambda v: json.dumps(v).
encode('utf-8')
)
def send_event(event):
   producer.send('events_topic', event)
   producer.flush()
# Sending an event
send_event({"event_type": "location_update", "driver_id":
123, "latitude": 40.7128, "longitude": -74.0060})
```

In this example, KafkaProducer sends a driver location update. This event can be processed by a separate microservice that analyzes and optimizes routes based on current traffic conditions [9].

Scalability with Kubernetes is particularly useful for creating services with high elasticity requirements, as seen in Netflix. The company actively uses Kubernetes and Docker to scale its microservices, which handle large volumes of multimedia data. The Kubernetes platform allows Netflix to adjust to fluctuating loads by launching additional instances of required microservices and maintaining fault tolerance. A sample manifest for deploying a microservice in Kubernetes is provided below:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-microservice
Spec:
 replicas: 3
 selector:
  matchLabels:
   app: my-microservice
 template:
  metadata:
   labels:
    app: my-microservice
  spec:
   containers:
   - name: my-container
    image: my-microservice-image:latest
    ports:
    - containerPort: 8080
    resources:
     limits:
      memory: “512Mi”
      cpu: “500m”
     requests:
      memory: “256Mi”
      cpu: “250m”
```

This YAML manifest deploys three replicas of a microservice with resource consumption limits. The platform automatically manages instance distribution and restarts instances if one fails, minimizing the impact of failures on overall system performance [10].

For organizations handling large volumes of transactional data, such as Goldman Sachs, CQRS and Event Sourcing models enhance data consistency and system reliability. In its transaction management system, Goldman Sachs separates commands (write operations) and queries (read operations) through a CQRS architectural approach, improving performance with asynchronous command processing and enabling high-speed data handling. A simplified example of code for CQRS with Event Sourcing is provided below:

```
from collections import defaultdict
# Event storage
events = defaultdict(list)
# Event Sourcing to save state
def save_event(aggregate_id, event):
   events[aggregate_id].append(event)
# Apply events to restore state
def apply_events(aggregate_id):
   state = {}
   for event in events[aggregate_id]:
      if event[‘type’] == ‘update_balance’:
         state[‘balance’] = state.get(‘balance’, 0) +
event[‘amount’]
   return state
# Saving events
save_event(1, {‘type’: ‘update_balance’, ‘amount’: 100})
save_event(1, {‘type’: ‘update_balance’, ‘amount’: -50})
# Applying events
print(apply_events(1))  # Outputs state {‘balance’: 50}
```

This code demonstrates the basic principles of Event Sourcing. Instead of storing each state separately, the system saves events describing changes. This allows states to be recreated based on all events and is beneficial for high-load systems requiring precise transaction history and flexibility in data handling [11].

These examples illustrate how leading organizations leverage microservices, including advanced architectural patterns like Event Sourcing and CQRS, to enhance data consistency, performance, and system resilience in high-load environments. Microservices architecture, with its modular and decoupled design, provides companies with the flexibility to scale, adapt, and maintain robust fault tolerance. By adopting this approach, organizations are better equipped to meet evolving market demands and user needs, fostering sustainable growth and operational agility across various domains.

CONCLUSION

The analysis and design of high-load systems based on microservices architecture confirm its significant advantages for creating resilient and flexible applications capable of adapting to changing loads and user demands. The study showed that a microservices approach to system decomposition facilitates easier component management and enhances service independence, which is essential for ensuring reliability and minimizing failure risks. The use of containerization and orchestration technologies, such as Docker and Kubernetes, enables dynamic scaling and deployment automation, reducing resource costs and maintaining high performance under intensive system operation.

The results demonstrated that implementing Event Sourcing and CQRS patterns enhances data consistency and query processing efficiency, which is critically important for high-load platforms handling large volumes of information. Thus, the application of microservices architecture, combined with modern management and control tools, provides stability, scalability, and flexibility, making it an optimal choice for building applications in environments with growing performance and availability requirements.

The conclusions of this study confirm that using a microservices approach, strengthened by containerization and orchestration, effectively addresses scalability, fault tolerance, and system component independence challenges.

REFERENCES

1. Desai V., Koladia Y., Pansambal S. Microservices: architecture and technologies //Int. J. Res. Appl. Sci. Eng. Technol. – 2020. – vol. 8. – No. 10. – pp. 679-686.

2. Li S. et al. Understanding and addressing quality attributes of microservices architecture: A Systematic literature review //Information and software technology. – 2021. – vol. 131. – p. 106449.

3. Microservices Architecture. [Electronic resource] Access mode: https://medium.com/design-microservices-architecture-with-patterns/microservices-architecture-2bec9da7d42a (accessed 10/23/2024).
4. Ao B. Integrated service system for multi-power business scenarios based on micro-service architecture //Reviews of adhesion and adhesives. – 2023. – vol. 11. – No. 2.
5. Liaghat B. et al. Short-term effectiveness of high-load compared with low-load strengthening exercise on self-reported function in patients with hypermobile shoulders: a randomized controlled trial // British Journal of Sports Medicine. – 2022. – Vol. 56. – No. 22. – pp. 1269-1276.
6. Kornuta V. et al. Using Microservice Architecture for High-Load Information Systems on the Example of MedicinePlanner Service //2022 12th International Conference on Advanced Computer Information Technologies (ACIT). – IEEE, 2022. – pp. 437-442.
7. Cao X., Zhang H., Shi H. Load Balancing Algorithm of API Gateway Based on Microservice Architecture for a Smart City //Journal of Testing and Evaluation. – 2024. – Vol. 52. – No. 3. – pp. 1663-1676.
8. Rath C. K., Mandal A. K., Sarkar A. Microservice based scalable IoT architecture for device interoperability // Computer Standards & Interfaces. – 2023. – Vol. 84. – pp. 103697.
9. Laigner R. et al. An Empirical Study on Challenges of Event Management in Microservice Architectures // arXiv preprint arXiv:2408.00440. – 2024.
10. Daradkeh T., Agarwal A. Modeling and optimizing micro-service based cloud elastic management system // Simulation Modelling Practice and Theory. - 2023. – Vol. 123. – p. 102713.
11. Youssfi M. et al. Multi-Micro-Agent System middleware model based on event sourcing and CQRS patterns // Smart Trajectories. – CRC Press, 2022. – pp. 25-46.
12. Fan C. F., Jindal A., Gerndt M. Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application //CLOSER. – 2020. – pp. 204-215.
13. Inside Spotify's Recommendation System: A Complete Guide to Spotify Recommendation Algorithms. [Electronic resource] Access mode: https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022 (accessed 10/23/2024).
14. LinkedIn Jobs Recommendation Systems. [Electronic resource] Access mode: https://pyimagesearch.com/2023/08/07/linkedin-jobs-recommendation-systems / (accessed 10/23/2024).

Citation: Denis Shelmanov, "Analysis and Design of High-Load Systems Based on Microservice Architecture", American Research Journal of Computer Science and Information Technology, Vol 7, no. 1, 2024, pp. 68-72.