Migration from SOA to Microservices Architecture: A Case-Based Evaluation of Performance Improvements and Architectural Trade-Offs

Sandeep Sharma[1] and Vijay Pal Singh[2]
[1] Department of Computer Engineering & Applications, Mangalayatan University, Aligarh, UP, India
[2] Department of Computer Engineering & Applications, Mangalayatan University, Aligarh, UP, India
sharmasandeep2617@gmail.com

Abstract. *This paper uses a real-world case-study approach to examine the shift from Service-Oriented Architecture (SOA) to Microservices Architecture (MSA), which focuses on performance enhancements and architectural trade-offs in popular and highly demanded software systems. It analyses key operational metrics of applications such as scalability, latency, availability, and deployment speed, using case-based scenarios from e-commerce and media streaming systems. The evaluation shows a significant increase in system agility and responsiveness when technologies like Kubernetes, Kafka, Redis, API Gateway, and CI/CD Pipelines are leveraged in MSA. However, this transition from SOA to MSA also has several challenges, including increased orchestration complexity, decentralized data management, distributed transaction handling, security enforcement across services, and governance of decentralized components. While addressing these pros and cons, this study gives practical recommendations for organizations considering similar architectural shifts. Moreover, integrating serverless computing and AI-driven observability remains an area that needs to be explored in future research to improve tooling, monitoring, and operational scalability in microservices ecosystems to increase agility and reduce the operation cost.*

Keywords: Microservices, SOA, Software Architecture, Migration, Application Modernization.

1 Introduction

To meet the growing demands for scalability, availability, lower Latency and rapid deployment, many organizations are increasingly shifting from traditional Service-Oriented Architecture (SOA) to modern Microservices Architecture (MSA) [1,2]. This architectural shift is particularly prominent in systems such as e-commerce and media streaming, where systems need high concurrency, low latency, and operational agility are needed [3,4].

SOA was once accepted as a new paradigm shift in software engineering. By breaking down large, tightly coupled systems into individual services connected through middleware—typically an Enterprise Service Bus (ESB)—SOA introduced flexibility and facilitated strategic alignment with business goals [6,7]. However, as the technological infrastructure become increasingly complex and grew, the drawbacks of SOA became noticeable. As a result, SOA deployments ended up being closely coupled that were supposed to be loosely integrated. This happened due to the use of shared databases, centralized dependencies, limited scalability, inter-service constraints, and the operational inefficiencies [7,8]. In contrast, a decentralized architecture is provided by MSA. This architecture designs systems as a group of independent services, where each service is responsible for a defined business function and manages its data [1,3]. MSA

approach promotes standalone services, continuous delivery and horizontal scaling in a efficient manner [1,5].

Despite these advantages, migrating from SOA to MSA introduces various architectural and operational complexities. common trade-offs include increased orchestration overhead, the need for decentralized security and governance, challenges in managing distributed transactions, monitoring services and infrastructure, tooling complexity, and operational overhead [6,9,10]. Therefore, organizations require a balanced understanding of performance gains and architectural trade-offs before initiating such transitions [2,5].

This paper addresses the need for SOA-to-MSA migration though case-based evaluation using performance-critical and highly-demand applications in e-commerce and media streaming domains. The study measures changes in key performance metrics, including latency, availability, scalability, and deployment speed, before and after migration. It also examines architectural limitations post migration, such as service orchestration complexity, data consistency challenges, Cache Invalidation issues, Security fragmentation, governance inconsistency, and tooling extensibility [6,9].

The finding indicates that implementing microservice architecture with the latest supporting technologies such as Kubernetes, Kafka, Redis, API Gateway, observability tools increase the system agility, responsiveness, and maintainability of the system [1,4]. However, these adaptations demand mature tooling systems, comprehensive monitoring, and disciplined cross-functional collaboration.

Using the real-world case study, this paper helps to provides the useful guidance for software architects and developers who are planning similar migration for application modernization. The study also highlights emerging directions, such as serverless computing and AI-driven observability, as potential avenues to address residual complexities in microservices architecture [5].

2 Related Work

Service-oriented architecture (SOA) emerged as a solution to overcome limitation of monolithic architecture by promoting modularity, reusability, and standalone service communication [6,7]. Integration across diverse systems was enabled through the use of common frameworks and a centralized orchestration layer, which is commonly implemented via an Enterprise Service Bus (ESB). In practice, solutions often became tightly interconnected because of the centralized control and shared data schemas. SOA implementations began facing difficulties such as performance bottlenecks, integration complexity, and governance overhead as a architectural complexity and user demand increased [7,8].

The key factor responsible for the adoption of microservices architecture was the need to address several drawbacks of SOA by focusing on distributed management and independent service control [3,9]. Every microservice operates independently, manages its database, and is usually associated to a specific operation. This architecture solution provides better fault isolation, greater autonomy, and more scalability. Agile and DevOps practices promote quicker development and deployment cycles that are well-aligned with this approach [1,5].

Over the last few years, context-based research has gained momentum at an accelerated pace, providing fact-based insights into how the migration from SOA to microservices is being performed within organizations [2,3]. It has been seen industries such as streaming media and e-commerce have been early adopters of microservices due to their high availability, elasticity, and fast feature delivery demands [1,4]. A number of research studies in these domains have reported significant improvements in system responsiveness, deployment rate, and fault tolerance capacity following the migration from SOA to MSA [1,3,5]. While microservices provide performance benefits, theses come with several trade-offs such as service orchestration complexity, security fragmentation, observability tooling, and data management issue between microservices [6,9,11].

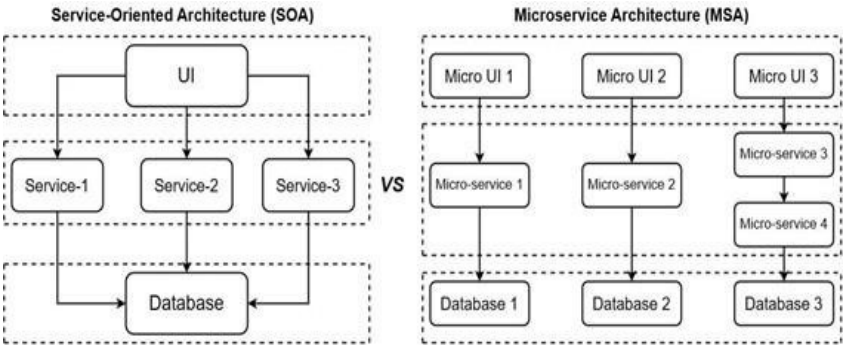

Figure 1. Conceptual architecture comparison between Service-Oriented Architecture (SOA) and Microservices Architecture (MSA)

3 Methodology

This paper explores a case-based evaluation approach to determine how the systems are affected by shifting from service-oriented architecture (SOA) to microservices architecture (MSA).

3.1 Research Design and Study Approach

A qualitative case study design is employed to examine the process of architectural migration and its implications at the operational level. Quantitative analysis complements this design by measuring key metrics before and after migration. This mixed-methods approach enables deeper insights into both behavioral and performance-based changes, which cannot be captured through simulation or theoretical models alone.

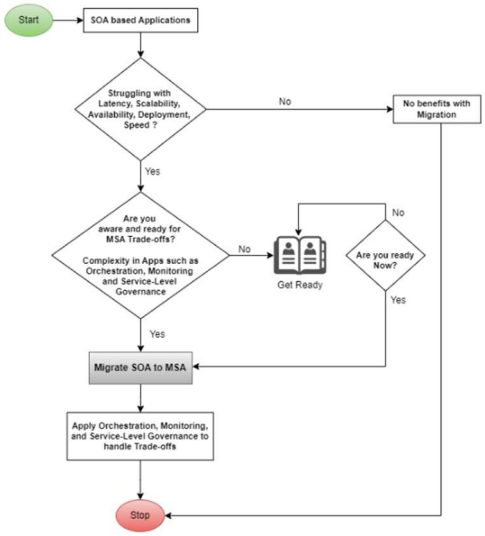

Figure 2. Migration readiness and transition flow from SOA to MSA.

3.2 Domain and Application Selection

In this paper, e-commerce and media streaming applications were selected because they are efficiency-oriented and technologically advanced. E-commerce applications require stable support for search, transaction processing, inventory refresh, and checkout operations. In contrast, media streaming applications require delivering continuous content with minimal or no buffering, adaptive quality, and rapid response times. Both domains face continuous load fluctuations, making them ideal for performance benchmarking.

3.3 Metrics and Evaluation Criteria

In this research, a set of widely accepted performance and operational metrics has been utilized. The below metrics were selected because of their applicability to large-scale systems and alignment with industry-standard architecture.

Latency: Refers to the total time required to respond to user or system requests.

Scalability: Defined as the system's ability to handle varying workloads by scaling out infrastructure.

Deployment Speed: Indicates the efficiency and frequency of releasing updates.

Availability: Represents the percentage of time a system remains operational under real-world conditions

3.4 Data Collection and Comparison Approach

Applications were first deployed using a Service-Oriented Architecture (SOA) and migrated to a Microservices Architecture (MSA). Both versions ran in identical cloud environments under controlled conditions. Apache JMeter is used for simulated the load on the application. Performance data was collected over a 3-day window using Prome-

theus, Grafana and CI/CD logs. Metrics such as latency, scalability, availability, and deployment speed were monitored. Post-migration architectural characteristics—service count, interdependencies, and database fragmentation—were also analyzed to compare structural complexity and operational impact between SOA and MSA.

3.5 Supporting Technologies

In this research, several modern technologies were used to support the shift from SOA to MSA, helping to improve performance and maintainability.

Kubernetes: Enabled container orchestration, auto-scaling, service discovery, and self-healing deployments.
Kafka: Facilitated event-driven, decoupled communication between microservices, improving message consistency.
API Gateway: Handled routing, authentication, rate limiting, and protocol translation for internal services.
Monitoring Tools: Prometheus and Grafana were used for real-time observability through metric collection and visualization.
CI/CD Pipelines: Automated testing, integration, and deployment to reduce errors and accelerate release cycles.
Additionally, **Python** and **C#** were used as the primary programming languages, with **MySQL** for transactional services, **MongoDB** for document storage, **ClickHouse** for time-series data, and a **Redis Cluster** for distributed caching to reduce latency and improve read performance.

3.6 Limitations

The findings of this research analysis are derived from limited areas and technology ecosystems, and they may be limited to the generalization to other industries or different applications. Some performance data used observational analysis, and complete access to core codebase or tooling environments was not always available. Moreover, post-migration complexities like increased or more tooling complexity, service orchestration overhead, distributed data management, and decentralized security, demand ongoing investment and organizational change. These restrictions were considered in the evaluation to ensure a fair assessment.

4 Results

The following section demonstrates the comparative results after migrating from a Service-Oriented Architecture (SOA) to a Microservices Architecture (MSA), based on two exemplary application domains: e-commerce and media streaming.

4.1 Performance Improvements Post-Migration

Migration from SOA to microservices brought notable improvements in performance across both media streaming and e-commerce spaces.
Latency was reduced by 21.8% to 25.8% in e-commerce and 24.2% to 36.0% in media streaming systems.

Scalability increased by 33.3% to 50% in e-commerce and 43.8% to 61.1% for media streaming.
Deployment speed improvements vary from 32.5% to 73.3% for e-commerce and 55.6% to 80% for media streaming.
Availability was enhanced by 58.1% to 63.9% in e-commerce and 74.5% to 77.0% in media streaming.

Table 2. Service Count Comparison Between SOA and MSA Across Applications.

Application Name	Service Count (SOA)	Service Count (MSA)
E-commerce	12	48
Streaming	10	42

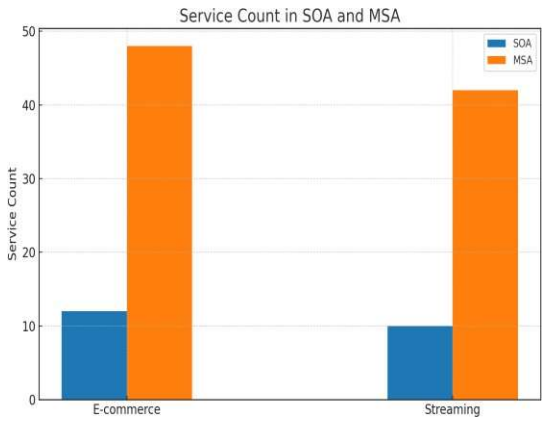

Figure 3. Comparison of Service Count After Migration from SOA to MSA.

Table 2. Comparative Performance improvement Metrics for SOA and MSA Across Applications

Application Name	Scalability (%)	Deployment Speed (%)	Latency (%)	Availability (%)
E-commerce	33.3–50.0	32.5–73.3	21.8–25.8	58.1–63.9
Streaming	43.8–61.1	55.6–80.0	24.2–36.0	74.5–77.0

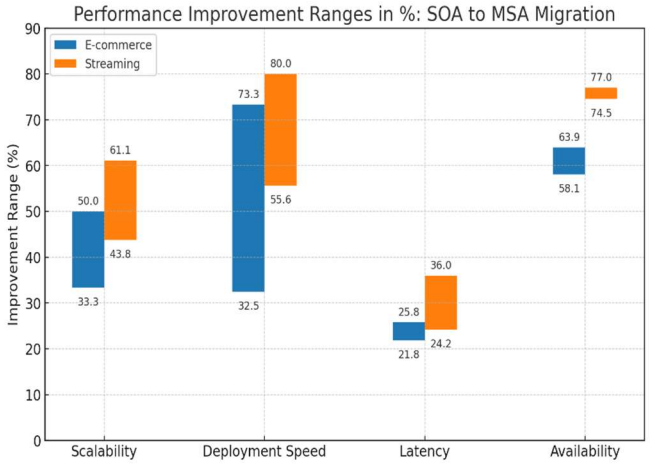

Figure 4. Comparison of Key Performance Improvements After Migration from SOA to MSA.

Table 3. Post-Migration Latency improvement(milliseconds)

Application Name	Concurrent Request	SOA Latency (millisecond)	MSA Latency (millisecond)	Improvement (%)
E-commerce	1000	780	610	21.8
E-commerce	10000	850	630	25.8
E-commerce	100000	880	660	25.0
Streaming	1000	950	720	24.2
Streaming	10000	1,100	780	29.1
Streaming	100000	1,250	800	36.0

Table 4. Post-Migration Scalability improvement (Throughput req/sec).

Application Name	Concurrent Request	SOA Through-put (req/sec)	MSA Throughput (req/sec)	Improvement (%)
E-commerce	1000	2,400	3,200	33.3
E-commerce	10000	2,500	3,600	44.0
E-commerce	100000	2,500	3,750	50.0
Streaming	1000	3,200	4,600	43.8
Streaming	10000	3,500	5,400	54.3
Streaming	100000	3,600	5,800	61.1

4.2 Architectural Trade-Offs and Limitations

There were still some complex architectural issues when the company moved from SOA to MSA, even though they have witnessed performance improvements. These shortcomings were observed in both domains. Common Limitations are:

Service Orchestration Complexity: The absence of a centralized orchestration layer made service integration highly complex, especially in high-throughput workflows.
Security Fragmentation: To handle the Identity management became challenging, as each service required its own authentication and authorization layers.
Observability Tooling: For end-to-end observability required the use of Prometheus, Grafana, Jaeger, and the various alerting systems**.**
Data Consistency: Keeping data consistency across microservice services is challenging. Patterns like Saga, Event Sourcing, and Outbox are commonly used to manage distributed transactions and support eventual consistency.

4.3 Summary of Findings

The shift from SOA to MSA resulted in notable improvements in performance across key metrics. However, these benefits were accompanied by challenges related to orchestration, security, and observability. Careful planning, tooling investment, and domain alignment are critical for success.

5 Conclusion

In this research analysis, a comparative study of Service-Oriented Architecture (SOA) and Microservices Architecture (MSA) was performed. Mainly. Two domain applications were employed, one in e-commerce and the other in media streaming. The main purpose was to evaluate the quantifiable influence of migration on the system performance using industry-driven operations.

A significant performance gain was achieved by the migration from SOA to MSA, as demonstrated by the analysis. These improvements were mainly achieved because of the utilization of Kubernetes for service deployment, CI/CD automation, Kafka for event-driven communication and observability tooling. On the other hand, architectural independence also enhanced the scalability and fault tolerance in parallel-processing scenarios. The shift from SOA to MSA also introduced new challenges such as orchestration complexity, security measures, and the requirement of the advanced monitoring infrastructure.

In future work, the approach could be further developed in other domains such as fintech or healthcare, where the trustworthiness and compliance of transactions introduce additional complications. Also, combining AI-powered observability could help address post-migration issues more effectively.

6 References

1. M. Villamizar, D. Garces, H. Castro, J. Verano. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. Future Generation Computer Systems 135, 345–359 (2023).
2. A. Balalaie, A. Heydarnoori, P. Jamshidi. Migrating to cloud-native architectures using microservices: An industrial survey. Journal of Systems and Software 192, 111384 (2022).

3.M. Khadka, A. Sapkota, J. Zdun, U. Zdun. A systematic literature review on the SOA to microservices migration. Journal of Systems and Software 197, 111545 (2023).
4.G. Preuveneers, W. Joosen. Self-adaptive microservice architectures: From goal models to runtime containers. ACM Computing Surveys 55(1), 1–39 (2022).
5.J. Soldani, D.A. Tamburri, W. van den Heuvel. The pains and gains of microservices: A systematic grey literature review. Journal of Systems and Software 146, 215–232 (2021).
6.M. Rahman, M.A. Babar, L. Zhu. Challenges and practices in adopting microservices architecture: A survey of industry experiences. ACM Trans. Softw. Eng. Methodology. 30(2), 1–47 (2021).
7.P. Jamshidi, M. Hoseini, M. Ghafari, C. Pahl. A survey of modeling approaches for microservice architectures. Journal of Systems and Software 170, 110708 (2021).
8.B. Taibi, V. Lenarduzzi. On the definition of microservice bad smells. IEEE Software 38(5), 56–64 (2021).
9.F. Ciccozzi, I. Malavolta, M. Sharaf. Model-driven engineering for microservice architecture. IEEE Software 38(5), 35–43 (2021).
10.A. Gokhale, A. Bhave, A. Dubey, G. Karsai. Challenges and opportunities in modeling and synthesizing cloud-native applications for edge computing. IEEE Software 38(1), 48–56 (2021).
11.J. Fritzsch, J. Bogner, A. Zimmermann, A. Wagner. Microservices migration in industry: Intentions, strategies, and challenges. Empirical Software Engineering 26(2), 1–39 (2021).