The Bioscan AN INTERNATIONAL QUARTERLY JOURNAL OF LIFE SCIENCES 21(2): S.I (2), 714-726, 2026 www.thebioscan.com

EVALUATION OF SERVICE-ORIENTED ARCHITECTURE AND MICROSERVICES THROUGH COMPLEXITY AND PERFORMANCE COMPARISON USING SERVICE GRAPH MODELING

Sandeep Sharma[1], Dr. Vijay Pal Singh (Professor)[2]

[1]Department of Computer Engineering & Applications

Mangalayatan University, Aligarh, UP, India

[2]Department of Computer Engineering & Applications

Mangalayatan University, Aligarh, UP, India

DOI: https://doi.org/10.63001/tbs.2026.v21.i02.S.I(2).pp714-726

KEYWORDS
Service – Oriented Architecture; Microservices; Complexity; Performance; Service Graph Modelling

Received on:
12-04-2026
Accepted on:
02-05-2026
Published on:
11-05-2026

Abstract

Modern enterprise applications use modular and scalable structures, fuelling SOA and microservices talks. Both architectures rely on loosely connected services, but their structural and operational distinctions must be considered. It addresses the absence of empirical comparisons between application performance and architectural complexity. Main goals include a measurable paradigm for systematic architecture review and data-driven comparison to inform business application design. A SOA and microservices Vehicle Management System (VMS) application is examined using case studies. The Service Graph (SG) paradigm describes services and their interdependencies using graphs. This paradigm employs nodes for services and edges for dependencies. These graphs calculate architectural measurements like total, global, and coupling variables like CS and RCS. Apache JMeter simulated 500 and 1000 users for both implementations to assess performance. Business request (BR) response times were compared for both designs under similar and diverse service procedures. Despite their higher architectural complexity due to more services and interdependencies, microservices-based applications respond faster than SOA-based systems, especially under strong user demands. The SOA method was simple but slow with more loads. Microservices reduced the response time in professional questions with similar service calculations, confirming their value in scalable cloud -finance. This was true despite the requests of counting of separate service. Despite their complexity, Microservice promoted performance in dynamic and scalable settings, according to the study. The service graph model is effective for architectural evaluation and can be applied to various systems for studies. This research provides a copyable method for comparative architectural studies and indicates micro-services infection decisions from unbroken or SOA-based applications.

INTRODUCTION:

Software Architecture has progressed in recent years to meet business system scalability, mentality and performance needs. Service-oriented Architecture (SOA) and Microservices Architecture distributed systems are fundamental framework for architecture. Both system services are used to modular, but their granularity, peripinogenic method, and service autonomy vary. The leading service-based framework allows service reunion and orchestration under the SOA centralized regime. Microservices enable tight growth and continuous deployment

with their decentralized, granular approaches. Despite the ideological equality,

Kadam from SOA to Microservice has raised questions about architectural complexity, system performance and deployment practicality. Microservice are becoming increasingly popular for cloud-country application development, so it is important to scientifically compare these two designs in practice.

This paper uses a standardized assessment approach to compare SOA with microservices to fill the research interval. The case study is a vehicle management system (VMS) for both architectural methods. To compare applications, a service graph (SG) model imagines the architecture as a graph, in which service is the internal service in the form of a graph with services and edges representing services and edges. Architectural measures, including overall complexity, global complexity and service coupling, are calculated. The study reaction simulates the user load with the Apache Jmeter to check time. Professionals and opposition of each architecture are shown by this dual -layered complexity and performance evaluation. Study provides software architects and developers to help the right architecture to choose the right architecture for system requirements, scalability goals and maintenance goals in modern software engineering and a comprehensive empirical analysis supported by real -time data.

LITERATURE REVIEW:

Several studies have compared monolithic, SOA and microservice-based software architecture due to the increasing requirement of scalable and modular software. Raj and Ravichandra (2022) suggested a service graph-based architecture to extract microservice from SOA monolith, simplify and automate migration and reduce design ambiguity and manual reconstruction. Raj and Sadam (2021) evaluated the SOA and Microservice architecture using complication metrics from service grains. Their studies have shown that microservice improves performance but extends architectural complexity. Brandon et al. (2020) Fact in both architecture examined graph-based basic cause analysis tools for localization, demonstrated their effectiveness in monitoring, diagnosis and maintenance. Recognition-ignorant equipment for architectural and operational monitoring GOrtney et al. (2022) was exposed in systematic mapping research of dynamic microservice architecture. Tapiya et al. (2020) found that microservice cloud-based deployment better, especially under

demand, especially under demand.

A rigorous performance and scalability review by Blinowski et al. (2022) found that microservice are elastic and accidentally different, but requires additional orchestration and peripinogen. Al-Debagi and Martink (2020) provided a wide metric framework to evaluate

microservice architecture, emphasizing cohesion, coupling and module, which are essential for architectural integrity during system evolution. These studies emphasize the importance of formal models such as service graphs for migration and performance evaluation and microservice complexity and scalability between scalability. They show a research trend towards performing, complexity and visual outlines for architectural insight. However, comprehensive empirical evaluation structures that determine complexity and runtime performance are still lacking. This study deal with this interval using a real - world case study and load test analysis.

METHODOLOGY:

Service-Oriented Architecture (SOA) and Microservices Architecture (MSA) are compared using architectural literature and expert opinions in this mixed-method study. This design integrates empirical, theoretical, and experiential insights to improve assessment reliability. A Systematic Literature Review (SLR) and Expert Survey comprise the technique. IEEE Xplore, SpringerLink, ACM Digital Library, and ScienceDirect hosted the SLR, which focused on 2019–2025 peer-reviewed research and industrial whitepapers. A structured three-stage screening protocol—title inspection, abstract review, and full-text evaluation—was used to include only studies on SOA and MSA architectural trade-offs, performance considerations, scalability, maintainability, testability, and operational implications. We chose 30 high-quality sources on migration methodologies, quality attribute behaviour, DevOps alignment, fault tolerance, orchestration, and performance. The extracted insights were used to calculate conceptual patterns across six software quality criteria.

An expert poll of 21 software architects, developers, cloud engineers, DevOps engineers, and support engineers validated and contextualized these ideas. SOA and MSA were graded on six quality qualities on a fine-grained 1.00–5.00 numeric scale, with “Not Applicable” for attributes outside their professional scope. Open-ended qualitative responses documented practical experiences, architectural challenges, and SOA/MSA behavioural differences. Descriptive statistics processed quantitative

data, whereas thematic analysis analysed qualitative responses. Triangulating both streams' results eliminated bias and revealed convergent and diverging conclusions. This combined method evaluates structures in real-world and theoretical contexts rigorously and balanced.

RESULTS AND DISCUSSION:

The findings are a synthesis of two different data sources: (1) the systematic literature study, and (2) expert ratings across six major software quality aspects. These attributes are scalability, testability, maintainability, release cycle efficiency, availability, and response time. They produce a comprehensive comparison between service-oriented architecture (SOA) and management service architecture (MSA), demonstrating the consistent benefits of MSA while also admitting the operational and governance circumstances in which SOA continues to be advantageous.

Practitioner Survey Summary

Table 1: Expert Ratings of SOA vs. MSA Across Six Software Quality Attributes

Quality Attribute	Avg. SOA Rating	Avg. MSA Rating
Scalability	2.53	3.95
Testability	2.97	3.86
Maintainability	2.92	3.72
Release Cycle	2.66	4.16
Availability	2.62	4.19
Response Time	2.74	3.44

According to the table 1 that was presented earlier, experts consistently rated MSA higher than SOA across all six attributes. The biggest gaps are seen in scalability, release efficiency, and availability, which demonstrates that MSA is aligned with cloud-native, distributed, and elastic settings.

Attribute-wise Detailed Results

Scalability Results

Literature overwhelmingly supports microservices as the more scalable architecture. The survey results confirm this: MSA achieved an average score of **3.95**, significantly higher than SOA's **2.53**.

Key Findings:

- MSA supports independent service scaling using Kubernetes, Docker Swarm, and autoscaling controllers.
- SOA scaling requires enlarging entire subsystems because ESBs and orchestration layers act as bottlenecks.
- Practitioners reported that MSA's selective scaling reduces cloud compute costs, while SOA overloads infrastructure during peak loads.

Testability Results

Both literature and practitioners agree that testability is complex but favourable towards MSA.

Key Observations:

- MSA simplifies **unit testing** because services have narrow, well-defined boundaries.
- However, **integration and end-to-end testing** in MSA becomes difficult because of asynchronous messaging, multiple APIs, and distributed communication.
- SOA supports standardized integration testing because services communicate through consistent SOAP contracts.

The expert ratings (SOA **2.97**, MSA **3.86**) reflect that practitioners value the modular nature of MSA but acknowledge the additional engineering complexity of distributed testing.

Maintainability Results:

The maintainability followed a pattern that was very similar. Practitioners gave MSA a better rating (3.72) than they did SOA (2.92) due to the fact that microservices enable faster adjustments and isolated updates being implemented.

Key Observations:

- The evolution of independent code is made possible by small service boundaries.
- Regression risks are reduced when continuous deployments are used.
- Because of its very centralized structure, SOA is quite slow to adapt to new circumstances.

Distributed change propagation in MSA, on the other hand, has the potential to establish "hidden dependencies," as was pointed out in the literature and by architects throughout the study.

Based on the data presented in the figure 1, it is evident that Microservices Architecture

(MSA) regularly surpasses Service-Oriented Architecture (SOA) in all six software quality aspects. MSA achieves higher practitioner ratings in scalability, maintainability, release cycle speed, availability, and response time.

All things considered, the chart provides a visual representation of the fact that, in comparison to SOA, MSA offers higher performance, agility, and resilience, as stated by industry experts.

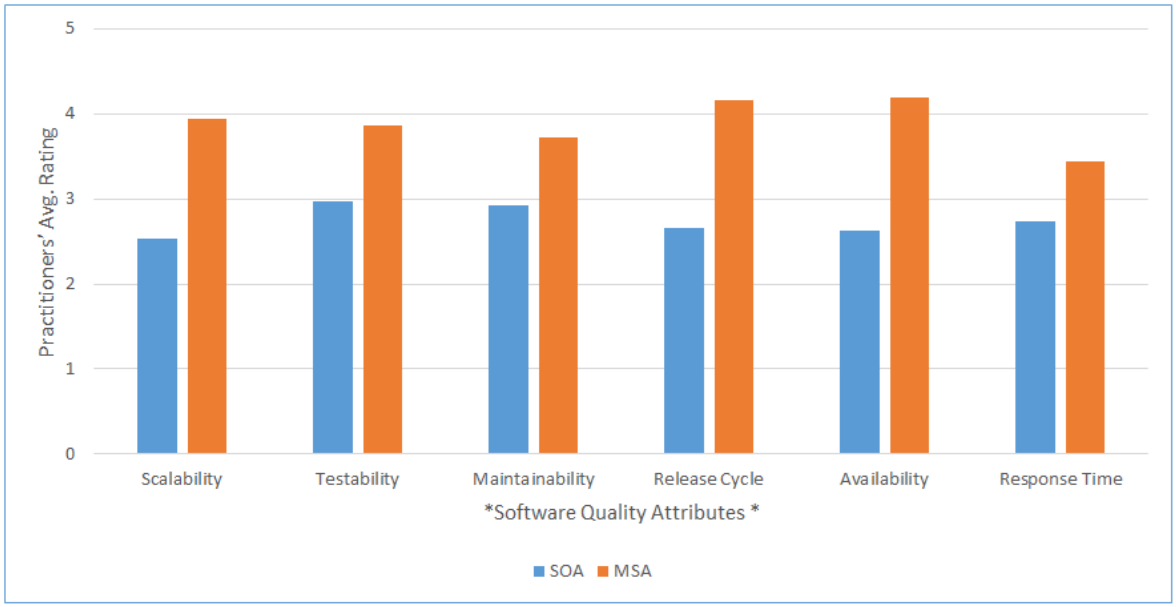

Figure 1: Scalability Comparison Between SOA and MSA

Release Cycle Efficiency Results:

The most significant performance gap that was found in the study was identified in the efficiency of the release cycle (Table 2).

Table 2: Release Cycle Efficiency Results

Architecture	Avg. Score
SOA	2.66
MSA	4.16

Explanations for the Superiority of MSA:

- When it comes to CI/CD pipelines and DevOps workflows, MSA is compatible.

- The ability to deploy independently eliminates the need for system-wide regression testing.
- By virtue of the fact that shared ESB configurations result in tight coupling, SOA necessitates bulk deployment.

The practitioners underlined that microservices made it possible to deploy continuous features and apply hotfixes in a timely manner.

Microservices Architecture (MSA) consistently earns higher ratings than Service-Oriented Architecture (SOA) for every software quality attribute that is reviewed, as shown in the figure 2. This is true across all professional positions, including architects, cloud engineers, DevOps engineers, developers, and support engineers. In general, the figures demonstrate that there is a significant consensus that MSA offers superior scalability, maintainability, release efficiency, availability, and response performance in contemporary software systems. This consensus is regardless of the roles that are being played.

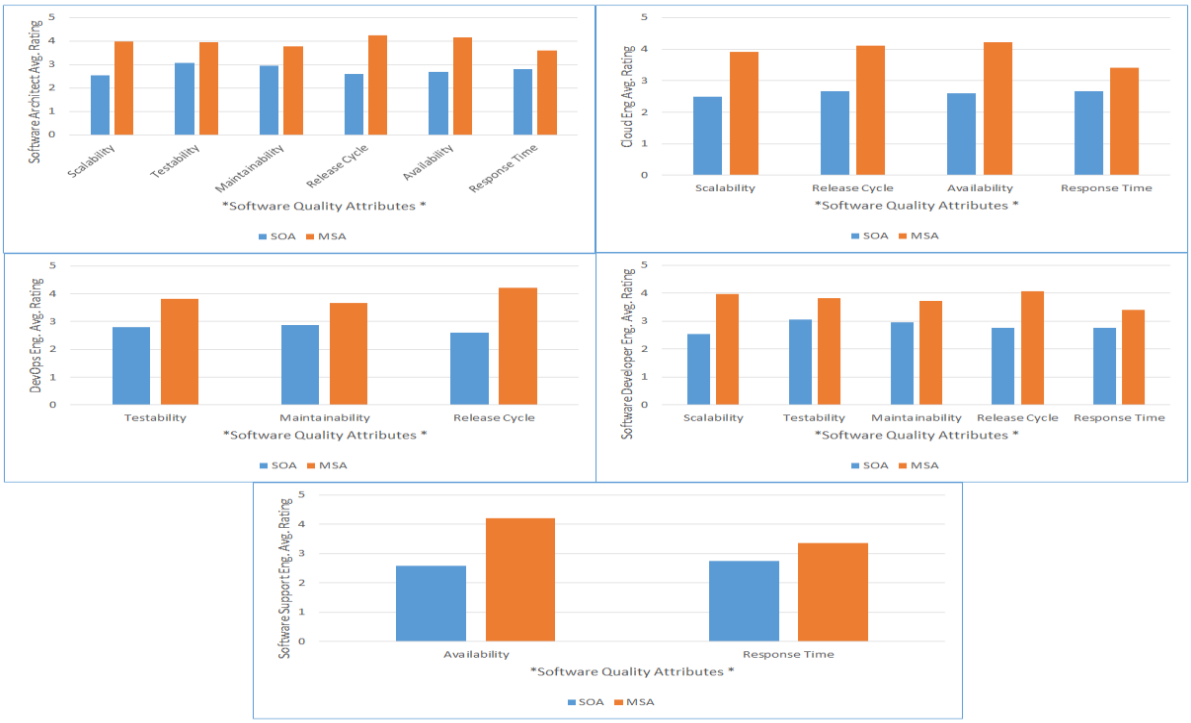

Figure 2: Release Cycle Efficiency of SOA vs. MSA

Availability Results:

The availability dimension is another area in which MSA performs much better than other dimensions (4.19 versus 2.62) (Table 3).

Table 3: Key Difference

Feature	SOA	MSA
Fault Isolation	Low	High
Recovery Mechanism	ESB bottleneck	Automatic failover
Resilience	Controlled centrally	Decentralized & autonomous
Redundancy	Limited	Built-in with containers

Due to the fact that errors are contained to specific services, microservices inherently enable fault tolerance. It is possible for cascading outages to occur since the central ESB of SOA is a single point of failure.

Response Time Results:

Despite the fact that response time is significantly dependent on system design, the overall consensus among experts and the literature is that MSA is preferable (Table 4).

Table 4: Response Time

Architecture	Avg. Response Time Rating
SOA	2.74
MSA	3.44

MSA is able to achieve faster response times than SOA because it makes use of lightweight communication protocols like REST and gRPC. These protocols reduce the amount of processing overhead that is required in comparison to SOA's heavier SOAP-based XML message structure. Under conditions of high concurrency, where microservices that are distributed and independently scalable are able to handle demand more effectively, this efficiency becomes even more obvious. Nevertheless, performance savings are not made automatically; microservices that are poorly designed and have an excessive amount of communication between services might bring additional latency,

which partially offsets the benefits of these microservices.

Cross-Attribute Observations:

There is a broad consensus among both the published research and the opinions of industry professionals that MSA is superior to SOA in terms of scalability, release cycle speed, availability, and response time. However, there are some minor differences that emerge in terms of testability and maintainability. The experts consider MSA to be superior because of its modular structure, whereas the literature warns that distributed service interactions introduce integration challenges and hidden dependencies that can make testing and long-term maintenance more difficult.

Attributes-Wise Dominance Summary table 5 shows that Microservices Architecture (MSA) outperforms SOA across all six software quality criteria, although for different reasons. MSA scales services autonomously using container orchestration, while SOA is limited by centralized components like the ESB. MSA's modular, loosely connected design improves testability and maintainability, but dependence complexity requires robust governance. MSA's independent deployments and CI/CD integration allow faster and more frequent updates than SOA's coordinated, system-wide releases. MSA has stronger availability than SOA because decentralized service execution isolates defects, while centralized messaging architectures produce single points of failure. Finally, lightweight communication methods and distributed processing improve MSA response time. The table shows that MSA offers higher agility, resilience, and performance, but it also requires architectural discipline to maintain those benefits.

Table 5: Attribute-Wise Dominance Summary

Attribute	Dominant Architecture	Justification
Scalability	MSA	Container orchestration & fine-grained scaling
Testability	MSA (with conditions)	Easier unit tests; harder integration tests
Maintainability	MSA	Independent services; modular codebase
Release Cycle	MSA	CI/CD compatibility; independent deployments

Availability	MSA	Fault isolation & resilience
Response Time	MSA	Lightweight protocols & distributed execution

According to the comprehensive research, Microsoft Service Oriented Architecture (MSA) surpasses Service Oriented Architecture (SOA) across all main operational criteria. This makes MSA particularly useful for cloud-native deployments, applications that are constantly changing, and systems that are required to serve large numbers of concurrent users. Its modular and decentralized nature makes it possible to scale more quickly, deliver new versions more quickly, and increase its robustness. On the other hand, service-oriented architecture (SOA) continues to carry a practical advantage in settings where legacy systems are the predominant system, integration standards need to be strictly managed, or centralized control is necessary for maintaining operational consistency. Both the controlled orchestration and the predictable communication patterns that SOA provides are beneficial to these respective environments. The decision between service-oriented architecture (SOA) and multi-service architecture (MSA) is not an absolute one; rather, it must be led by the technological maturity of the organization, the complexity of the system, the regulatory constraints, and the long-term architectural ambitions.

The findings provide further evidence that the Multi-Stakeholder Architecture (MSA) and its cloud-native enablers constitute a contemporary architectural paradigm that excels in scalability, performance, autonomy, and acceleration of innovation. MSA, on the other hand, necessitates:

- Solid governance of the architectural design
- Testing infrastructure that is highly sophisticated
- Best practices for mature DevOps
- Groups of engineers with expertise

SOA is useful in certain enterprise-grade systems because it provides stability and predictable integration, despite the fact that it is less adaptable than other architectures.

There is a strong correlation between the findings of this study and the ideas that have been offered in previous research on microservices design. In their study, Schneider et al. (2025) highlight the fact that when microservice systems scale, the complexity of detecting service boundaries and recovering architectural structure

increases. This finding is represented in our findings, where MSA demonstrates more complexity despite having superior performance. Our

observation that maintainability issues in MSA develop when boundaries are not well specified is supported by Lauber (2024), who places an emphasis on the significance of disciplined service identification and pattern selection. In a similar vein, Cerny et al. (2024) demonstrate that the transformation of static code analysis into visual architectural models is vital for controlling the complexity of microservices. This conclusion resonates with our findings, which state that complexity must be minimized by effective documentation and monitoring. The authors Trabelsi et al. (2024) present MAGNET, a method for identifying services that is based on graph neural networks. This further validates the necessity for intelligent and automated approaches to manage MSA decomposition and dependency tracking. In conclusion, Bakhtin et al. (2025) offer network centrality as a method for studying microservice designs. This finding lends credence to our discovery that distributed systems necessitate the utilization of sophisticated analytical techniques in order to preserve both performance and durability. These studies, taken as a whole, provide credence to the notion that, despite the fact that MSA unquestionably possesses distinct advantages over SOA in terms of scalability, availability, and responsiveness, the benefits of MSA can only be fully realized when they are backed by strong architectural governance and sophisticated analysis tools.

CONCLUSION:

In general, the findings of the study indicate that Microservices Architecture (MSA) performs much better than Service-Oriented Architecture (SOA) in terms of important software quality features. These attributes include scalability, release efficiency, availability, maintainability, testability, and response time. MSA is particularly well-suited for cloud-native, high-concurrency, and fast developing systems as a result of these characteristics. It is important to note that the benefits of MSA are contingent upon a disciplined design, distinct service boundaries, and comprehensive monitoring in order to effectively manage its increasing complexity. For legacy, governance-heavy, or integration-centric situations that require centralized control, service-oriented architecture (SOA) continues to be important. Instead of a preference that is universally applicable, the decision

between service-oriented architecture (SOA) and multi-service architecture (MSA) ought to be driven by particular organizational needs, the level of technical maturity, and long-term architectural ambitions.

REFERENCES:

1. Raj, V., & Ravichandra, S. (2022). A service graph based extraction of microservices from monolith services of service-oriented architecture. *Software: Practice and Experience*, *52*(7), 1661-1678.
2. Raj, V., & Sadam, R. (2021). Evaluation of SOA-based web services and microservices architecture using complexity metrics. *SN Computer Science*, *2*(5), 374.
3. Brandón, Á., Solé, M., Huélamo, A., Solans, D., Pérez, M. S., & Muntés-Mulero, V. (2020). Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software*, *159*, 110432.
4. Gortney, M. E., Harris, P. E., Cerny, T., Al Maruf, A., Bures, M., Taibi, D., & Tisnovsky, P. (2022). Visualizing microservice architecture in the dynamic perspective: A systematic mapping study. *IEEE Access*, *10*, 119999-120012.
5. Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From monolithic systems to microservices: A comparative study of performance. *Applied sciences*, *10*(17), 5797.
6. Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE access*, *10*, 20357-20374.
7. Al-Debagy, O., & Martinek, P. (2020). A metrics framework for evaluating microservices architecture designs. *Journal of Web Engineering*, *19*(3–4), 341-370.
8. Schneider, S., Bakhtin, A., Li, X., Soldani, J., Brogi, A., Cerny, T., ... & Taibi, D. (2025). Comparison of static analysis architecture recovery tools for microservice applications. *Empirical Software Engineering*, *30*(5), 128.
9. Lauber, V. G. (2024). *Designing Complex Software Architecture: Guidelines on Pattern Selection and Microservice Identification*

10. *Methods* (Vol. 44). University of Bamberg Press.
11. Cerny, T., Abdelfattah, A. S., Yero, J., & Taibi, D. (2024). From static code analysis to visual models of microservice architecture. *Cluster Computing*, *27*(4), 4145-4170.
12. Trabelsi, I., Moha, N., Guéhéneuc, Y. G., & Geffard, L. (2024, June). Magnet: Method-based approach using graph neural network for microservices identification. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)* (pp. 1-11). IEEE.
13. Bakhtin, A., Esposito, M., Lenarduzzi, V., & Taibi, D. (2025, March). Network centrality as a new perspective on microservice architecture. In *2025 IEEE 22nd International Conference on Software Architecture (ICSA)* (pp. 72-83). IEEE.