Bulletin of Electrical Engineering and Informatics
Vol. 14, No. 3, June 2025, pp. 2429~2437
ISSN: 2302-9285, DOI: 10.11591/eei.v14i3.9099

Strategies, characteristics, and research gaps for improving microservices coupling design

Gintoro[1], Sunardi[2]
[1]Department of Computer Science, School of Computer Science, Bina Nusantara University, Jakarta, Indonesia
[2]Department of Information System, BINUS Online Learning, Bina Nusantara University, Jakarta, Indonesia

Article Info

Article history:

Received Aug 3, 2024
Revised Jun 21, 2025
Accepted Mar 9, 2025

Keywords:

Microservices architecture
Research gaps
Service coupling
Strategies and characteristics
Systematic literature review

ABSTRACT

The popularity of microservices architecture (MSA) has been pushed by the demand for scalable, maintainable, and efficient applications in the fast-changing digital ecosystem. The objective of this study is to determine strategies for improving service coupling in MSA, analyze the circumstances in which these strategies are successful, and recommend areas of research that need further development for future enhancements. We employed a systematic literature review (SLR) and the seven research gap methodology developed by Müller-Bloch and Kranz to pinpoint 10 essential strategies, such as API gateway and domain-driven design (DDD). The results of our study indicate that the effectiveness of each technique is contingent upon specific design criteria for the microservices, such as the presence of separate read and write operations for command query responsibility segregation (CQRS). To further enhance these techniques, it is crucial to address the research gaps that have been highlighted, particularly the lack of empirical studies on long-term repercussions. This study offers theoretical insights and practical assistance on how to improve the connection between services, thereby enabling the development of more resilient and easily maintainable applications based on MSA.

Corresponding Author:

Gintoro
Department of Computer Science, School of Computer Science, Bina Nusantara University
KH Syahdan No. 9, Kemanggisan, Palmerah, West Jakarta, Jakarta, Indonesia
Email: gintoro@binus.ac.id

1. INTRODUCTION

The rapid evolution of the digital ecosystem has increased the demand for software systems that are scalable, maintainable, and capable of adapting to dynamic requirements [1]. Microservices architecture (MSA) has emerged as a prominent solution to meet these demands by enabling the decomposition of monolithic applications into smaller, loosely coupled, and independently deployable services [2]. This modular approach enhances scalability, fault tolerance, and team productivity, making MSA a preferred choice for complex software systems [3].

One critical challenge in adopting MSA is managing the coupling between services. Service coupling refers to the extent of dependencies between microservices, which directly impacts maintainability, scalability, and system resilience [4], [5]. Low coupling enables independent development and deployment of services, while high coupling introduces complexities that hinder these benefits [6]. Although various strategies have been proposed to address service coupling, practitioners often face challenges in identifying and implementing the most suitable strategies for their specific contexts [7].

Existing studies offer valuable insights into strategies for improving service coupling in MSA [8]. For instance, Vural and Koyuncu [9] explored the use of domain-driven design (DDD) for modularity enhancement, while Bogner *et al.* [10] emphasized the role of API gateways in reducing interdependencies among services. Similarly, Ntentos *et al.* [11] provided a model-based evaluation of coupling-related practices. However, a comprehensive synthesis integrating these strategies with specific design characteristics remains absent, and significant research gaps persist in evaluating their long-term effectiveness and scalability.

The existence of numerous strategies makes possible some potential research gaps that, if filled, might be further refined. It is essential to identify these gaps so that the strategies are effective and put into place with the unique challenges that MSA applications bring. The present research will try to address this gap by looking at the following research questions:

RQ1: What strategies can be employed to improve the coupling in service design based on MSA?

RQ2: Under what specific conditions or characteristics in microservice design can the strategies identified in RQ1 be effectively employed?

RQ3: How can the strategies established in RQ1 be improved by examining the existing research gaps in the field?

This study addresses these gaps by systematically analyzing strategies for improving service coupling, identifying conditions under which these strategies are effective, and highlighting research areas requiring further exploration. Using a systematic literature review (SLR) guided by Kitchenham's methodology [12] and Müller-Bloch and Kranz's seven research gap framework [8], [13], this research provides theoretical and practical insights for software architects and researchers. The contributions of this paper are as follows:

a. Identification of ten key strategies for improving service coupling in MSA.
b. Analysis of specific microservice design characteristics that enhance the applicability of these strategies.
c. A comprehensive gap analysis to highlight areas requiring further research, offering directions for future studies.

The structure of this paper is as follows: section 2 reviews the existing literature and method used in this study. Section 3 presents the findings, including identified strategies and their associated conditions. Section 4 discusses the implications of the findings, identifies research gaps, and suggests improvements. Finally, section 5 concludes the study and outlines potential future research directions.

2. METHOD

This study employs a two-stage methodology designed to systematically analyze strategies for improving service coupling in MSA and identify areas requiring further research. The methodology integrates an SLR based on Kitchenham's framework [12] and the seven research gap methodology developed by Müller-Bloch and Kranz [8], [13]. These approaches ensure a comprehensive understanding of existing strategies, their applicability, and research gaps in the field. The methodology in this study is illustrated in the diagram blocks shown in Figure 1. The following sections will provide a detailed description of each methodology.

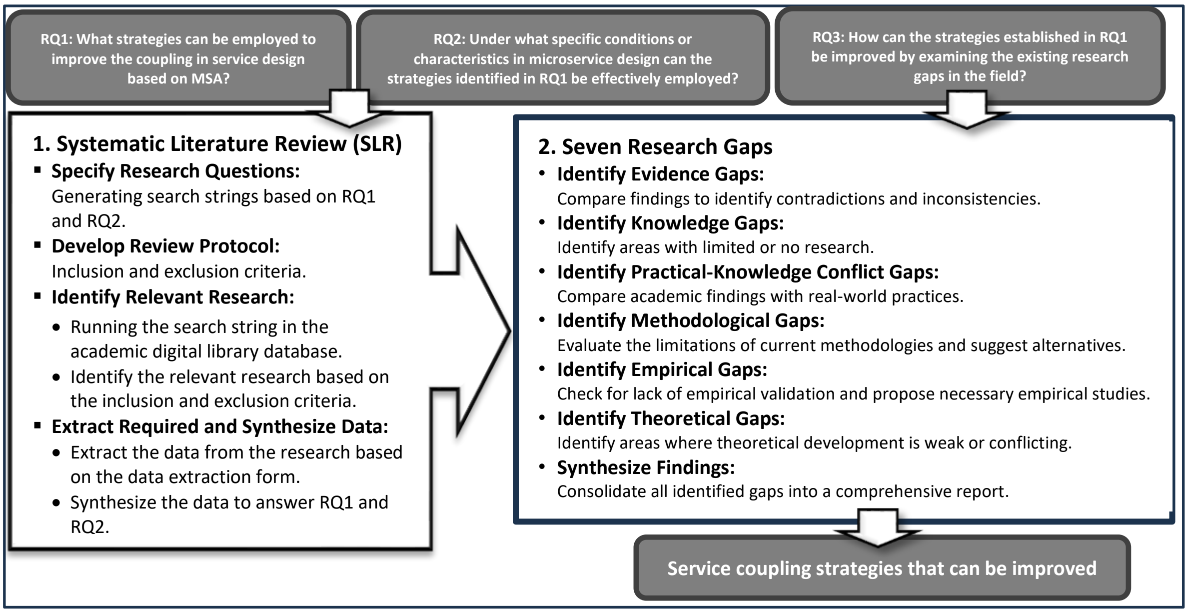

Figure 1. Research method block diagram

2.1. Systematic literature review

The SLR method was adopted to synthesize existing knowledge on strategies for improving service coupling and their associated design characteristics. This process involved the following steps [12]:

2.1.1. Specify research questions

The SLR process will begin by determining the research questions based on the issues presented above. The research questions used in this stage are aimed at obtaining answers to the issues of identifying strategies for improving service coupling and the characteristics of service design that can be enhanced with these strategies. Based on these two factors, the following research questions are formulated:
RQ1: What strategies can be employed to improve the coupling in service design based on MSA?
RQ2: Under what specific conditions or characteristics in microservice design can the strategies identified in RQ1 be effectively employed?

2.1.2. Develop review protocol

The next step of the SLR will involve compiling a review protocol that will serve as a reference for selecting articles found in the academic digital library. This protocol is designed to apply inclusion and exclusion criteria in determining which articles will be further analyzed. These criteria are developed based on the aim of addressing the two research questions mentioned above. Table 1 presents a comprehensive list of criteria for inclusion and exclusion that should be applied.

Table 1. Article selection criteria

Inclusion criteria	Exclusion criteria
I1. Studies explicitly discuss strategies to enhance service coupling in microservices.	E1. Studies are not written in English.
I2. Studies related to the characteristics of service design can be enhanced by using the strategy found above.	E2. Studies are editorials, books, articles, opinions, or technical reports.
I3. Articles published within the last 5 years to ensure relevance.	E3. Studies not directly related to microservices or service coupling improvement strategy.

2.1.3. Identify relevant research

A comprehensive search was conducted across five academic databases: IEEE Xplore, Springer Link, ScienceDirect, Wiley Online Library, and ACM Digital Library. The search string used was: ("Improving" OR "Enhancing" OR "Optimizing" OR "Strategy") AND "Coupling" AND "Microservice*". Table 2 displays the results from this third step. The search string yielded 431 articles from five academic digital libraries. After reviewing the abstracts and selecting articles based on the selection criteria in Table 1, we identified 32 articles as potential candidates for further evaluation. Upon careful analysis of the specifics of each article based on their respective results, we have selected 18 articles that discuss strategies and their service design characteristics for further evaluation. This article will go to the next step, information extraction.

Table 2. Digital library search result

No.	Academic digital library	Found	Candidate	Selected
1.	IEEE digital library (http://ieeexplore.ieee.org)	106	7	5
2.	Springer link (http://link.springer.com)	93	8	5
3.	ScienceDirect (http://www.sciencedirect.com)	83	10	4
4.	Willey online library (https://onlinelibrary.wiley.com)	44	5	3
5.	ACM digital library (http://portal.acm.org)	105	2	1
Total articles		431	32	18

2.1.4. Extract required and synthesize data

The fourth step involves extracting and synthesizing data, which includes collecting data from selected studies using standardized forms, analyzing and integrating data to identify patterns and trends that will go a long way in addressing the research questions, and giving a well-rounded understanding of the topic under review. The information extraction method will utilize the form depicted in Table 3. Each article will extract information based on the information structure outlined in Table 3.

Table 3. Data extraction form

Data field	Notes
F1. Title and author of the document.	The author's credibility.
F2. Service coupling improvement strategy.	Related with RQ1.
F3. Characteristics from the service design that can be improved with the strategy.	Related with RQ2.

Key data was extracted from the selected studies to identify recurring themes and trends, providing insights into strategies for improving service coupling. This process involves organizing the extracted information into Tables 4 to 6 to ensure clarity and structure. Table 4 presents an overview of the identified strategies for service coupling improvement, along with a summary of their characteristics and relevance. Each strategy is aligned with its respective microservices design characteristic to highlight its contextual applicability. This information is located in Tables 5 and 6, which summarize the findings on research gaps and are categorized according to the seven research gap framework. These gaps include evidence gaps, practical knowledge gaps, and methodological gaps. This categorization helps provide actionable directions for future research and highlights areas that require further empirical validation.

Table 4. List of strategies to enhance service coupling

No.	Strategy	Description	Impact on coupling	Related studies
1.	API gateway	Centralizes routing, load balancing, and cross-cutting concerns such as security and monitoring.	Provides a single entry point for client requests, reducing direct client-to-microservice coupling and centralizing cross-cutting concerns.	[4], [7], [10], [14]-[17]
2.	Command query responsibility segregation (CQRS)	Separates read and write operations into different models: commands to update data and queries to read data.	Simplifies design by decoupling read and write operations, facilitating independent evolution, and reducing service coupling.	[18]-[21]
3.	DDD	Divides the software into bounded contexts, each representing a specific part of the business domain.	Ensures each microservice has a well-defined scope and responsibility, reducing dependencies and enhancing modularity.	[9], [21]-[23]
4.	Saga pattern	Maintains consistency across distributed services by using compensating transactions.	Ensures system consistency without tightly coupling services, allowing for reversible operations and reducing dependency.	[20], [24], [25]
5.	Event sourcing	Stores all changes to the application state as a sequence of events.	Reduces the need for complex transactions and database locks, improving service independence and reliability.	[10], [19], [20]
6.	Strangler application pattern	Incrementally replaces parts of a legacy system with new microservices.	Ensures each new microservice can be developed with loose coupling, reducing risk and disruption associated with large-scale refactoring.	[22], [26]
7.	Chain pattern	Arrange components in a pipeline layout, with each service processing data and passing it to the next.	Establishes clear one-to-one relationships, minimizes dependencies, and enhances modularity.	[27]
8.	Fan (distributed) pattern	All clients establish many-to-one relationships with a central database server.	Centralizes data management, reduces inter-service dependencies, and simplifies data access.	[27]
9.	Balanced (shared data) pattern	Each service maintains its own database but allows data sharing among multiple services.	Reduces the need for direct service interactions, minimizing tight coupling and improving system robustness.	[27]
10.	Adapter microservices pattern	Creates adapter services that act as intermediaries between microservices and legacy systems or APIs.	Isolates legacy system dependencies within adapter services, allowing new microservices to be developed and deployed independently.	[22]

2.2. Seven research gaps

The identification of research gaps is a systematic process that consists of steps with the aim of ensuring all-around and proper detection of areas where further research is needed. The steps described in Table 7 present a description of the inputs, processes, and outputs derived from the identification of research gaps based on the developed framework of Müller-Bloch and Kranz [8], [13]. Each strategy identified through the SLR was analyzed using this framework to uncover gaps and opportunities for further research. The identified gaps were synthesized into actionable recommendations to guide future studies and improve the practical applicability of service coupling strategies.

2.3. Method overview

The combined application of SLR and the seven research gap methodology ensures a holistic approach to understanding service coupling in MSA. The SLR establishes a robust foundation of existing strategies, while the research gap analysis highlights deficiencies and areas for innovation. The methodology enables both theoretical insights and practical recommendations, contributing to the advancement of MSA-based system design.

Table 5. Condition or characteristics of service design that can be improved with this strategy

No.	Strategy	Conditions/characteristics	Impact
1.	API gateway	When clients need to interact with multiple microservices, managing multiple API calls and tokens.	Centralized routing reduces client-to-microservice coupling and simplifies cross-cutting concerns like security and logging.
2.	CQRS	Distinct read and write operations, high concurrency, need for independent scaling of read/write, complex business logic.	Simplifies design by decoupling read and write operations, facilitating independent evolution, and reducing service coupling.
3.	DDD	Complex business domains require clear modeling of subdomains and interactions, as well as migration from monolithic to MSA.	Ensures well-defined service boundaries, reduces dependencies, and enhances modularity by focusing on core business capabilities.
4.	Saga pattern	Transactions spanning multiple services require a sequence of operations that are consistent over time and need reversible operations.	Ensures system consistency without tightly coupling services, allowing for reversible operations and reducing dependency.
5.	Event sourcing	Frequent state changes, complex business logic, inter-service communication overhead, data recovery and consistency needs, asynchronous processing.	Reduces the need for complex transactions and database locks, improving service independence and reliability.
6.	Strangler application pattern	Incremental replacement of legacy system parts with microservices, minimizing risk and disruption during transition.	Ensures new microservices are developed with loose coupling, reducing risk and allowing for continuous improvement.
7.	Chain pattern	Strict order of operations, minimal shared state, independent processing steps, clear input/output contracts.	Establishes clear one-to-one relationships, minimizes dependencies, and enhances modularity.
8.	Fan (distributed) pattern	High data consistency requirements, single source of truth, simplified data access, reduced direct service interactions, and scalability needs.	Centralizes data management, reduces inter-service dependencies, and simplifies data access.
9.	Balanced (shared data) pattern	High read/write operations, data synchronization needs, scalability requirements, overlapping data requirements, and avoidance of single points of failure.	Reduces the need for direct service interactions, minimizing tight coupling and improving system robustness.
10.	Adapter microservices pattern	Interaction with legacy systems or external services using different protocols and incremental migration of functionalities.	Isolates legacy system dependencies within adapter services, allowing new microservices to be developed and deployed independently.

3. RESULT AND DISCUSSION

This section presents the findings of the study, including the strategies identified to improve service coupling in MSA, the conditions under which these strategies are most effective, and a critical analysis of research gaps. Each subsection aligns with the research questions outlined in the method.

3.1. Strategies to enhance service coupling (RQ1)

A comprehensive literature analysis was undertaken to investigate ways that might be employed to improve the service coupling of applications based on MSA, which addresses the first study question. The review identified several ways that help enhance service coupling in MSA. Table 4 provides a concise summary of each strategy, including its description, its effect on coupling, and the relevant papers associated with it. The list in Table 4 is arranged in descending order based on the frequency of studies mentioning each strategy. Each strategy addresses specific aspects of service coupling, providing software architects with multiple options to tailor their solutions based on application requirements. The findings indicate that combining multiple strategies results in optimal outcomes.

3.2. Conditions or characteristics of service design that can be improved by identified strategies (RQ2)

In order to answer the second research question, which focuses on identifying the specific conditions or characteristics in microservice design that are suitable for the effective implementation of the strategies mentioned in RQ1, we examine the circumstances in which each strategy can be utilized to enhance service coupling. The strategies suggested in RQ1 are aligned with specific MSA design characteristics that augment their efficacy. The list of strategies and their MSA design characteristics can be seen in Table 5. By aligning strategies with specific characteristics, this study provides actionable guidance for practitioners, ensuring optimal implementation and results.

3.3. Addressing research gaps in enhancing service coupling strategies (RQ3)

The third research question is how to enhance the strategies identified in RQ1 by analyzing the current gaps in the area. We will answer this RQ by utilizing the seven research gap framework developed by Müller-Bloch and Kranz. This framework facilitates the identification and classification of deficiencies in the existing body of literature, hence creating possibilities for future study and advancements in the methods used to enhance the interconnection of services in MSA. Müller-Bloch and Kranz have identified seven different types of research gaps [8], [13]: i) evidence gap; ii) knowledge gap; iii) practical-knowledge gap; iv) methodological gap; v) empirical gap; vi) theoretical gap; and vii) population gap.

We examine how these gaps relate to each of the ten strategies identified in RQ1, emphasizing areas that could be improved. The gap analysis for each strategy can be seen in Table 6. Addressing these gaps will significantly advance the understanding and practical application of these strategies, making MSA-based systems more resilient and maintainable.

Table 6. Gap analysis from each strategy

No.	Strategy	Identified research gaps	Improvement opportunities
1.	API gateway	Evidence gap: limited empirical studies on long-term performance and scalability impacts. Methodological gap: lack of standardized methods for evaluating security effectiveness.	Conduct longitudinal studies to assess performance and scalability over time. Develop standardized methods for evaluating security in API gateways.
2.	CQRS	Practical-knowledge gap: limited guidelines on integrating cqrs with existing monolithic systems. Theoretical gap: lack of a unified theory on cqrs's impact on system complexity.	Develop practical guidelines and best practices for integrating CQRS into legacy systems. Formulate a unified theoretical framework for CQRS.
3.	DDD	Knowledge gap: insufficient understanding of ddd's impact on team collaboration and communication. Population gap: limited studies in non-enterprise environments.	Conduct studies on DDD's impact on team dynamics and collaboration. Explore the application of DDD in small and medium-sized enterprises (SMEs).
4.	Saga pattern	Empirical gap: few real-world case studies demonstrating saga pattern's effectiveness. Methodological gap: lack of tools for automated compensating transaction management.	Collect and analyze real-world case studies to validate the Saga Pattern. Develop tools for automating compensating transaction management.
5.	Event sourcing	Evidence gap: limited data on event sourcing's impact on system latency and throughput. Theoretical gap: incomplete theories on event sourcing's role in system evolution.	Perform empirical studies to measure latency and throughput impacts. Develop comprehensive theories on event sourcing's role in system evolution.
6.	Strangler application pattern	Practical-knowledge gap: limited practical guidance on managing transition phases Population gap: Few studies have been done in industries other than finance and e-commerce.	Create detailed guidelines for managing the transition from monolithic to microservices. Investigate the pattern's application in various industries.
7.	Chain pattern	Methodological gap: lack of standardized benchmarking methods. Theoretical gap: incomplete theoretical models on the impact of chaining on fault tolerance.	Develop standardized benchmarking methods for evaluating the Chain Pattern. Formulate theoretical models on fault tolerance in chained services.
8.	Fan (distributed) pattern	Evidence gap: sparse empirical evidence on scalability limits. Practical-knowledge gap: insufficient guidelines on balancing load across distributed services.	Conduct empirical research to determine scalability limits. Provide comprehensive guidelines for effective load balancing in distributed systems.
9.	Balanced (shared data) pattern	Empirical gap: limited real-world applications demonstrating the pattern's effectiveness. Knowledge gap: inadequate understanding of data synchronization challenges.	Document and analyze real-world applications of the Balanced Pattern. Research data synchronization challenges and propose solutions.
10.	Adapter microservices pattern	Theoretical gap: lack of theories explaining the long-term impact of adapters on system evolution. Population gap: few studies on the pattern's application in emerging markets.	Develop theories on the long-term impact of adapters. Conduct studies on the pattern's application in emerging and developing markets.

Table 7. Seven research gaps from Müller-Bloch and Kranz

No.	Gap	Input	Process	Output
1.	Identify evidence gaps	Synthesized literature	Compare findings to identify contradictions and inconsistencies.	List of evidence gaps where findings are contradictory.
2.	Identify knowledge gaps	Synthesized literature	Identify areas with limited or no research.	List of knowledge gaps highlighting unexplored areas.
3.	Identify practical-knowledge conflict gaps	Practitioner interviews, industry reports	Compare academic findings with real-world practices.	List of practical-knowledge conflict gaps.
4.	Identify methodological gaps	Research methods used in existing studies	Evaluate the limitations of current methodologies and suggest alternatives.	List of methodological gaps needing innovative approaches.
5.	Identify empirical gaps	Theoretical models and hypotheses	Check for lack of empirical validation and propose necessary empirical studies.	List of empirical gaps needing validation.
6.	Identify theoretical gaps	Existing theories and models	Identify areas where theoretical development is weak or conflicting.	List of theoretical gaps needing refinement.
7.	Identify population gaps	Demographic analysis of study populations	Identify underrepresented groups in research.	The list of population gaps needs to include broader demographics.
8.	Synthesize findings	Lists of various gaps identified	Consolidate all identified gaps into a comprehensive finding.	Comprehensive findings of research gaps.

3.4. Discussion

The results of this study demonstrate that service coupling in MSA can be effectively managed through a combination of well-defined strategies. Each strategy addresses specific aspects of coupling, offering flexible solutions for varying application needs. For example:

a. API gateway centralizes client interactions, enhancing modularity but introducing challenges like potential bottlenecks.
b. CQRS simplifies concurrent operations yet increases the complexity of system design.
c. DDD ensures well-defined boundaries but requires substantial initial modeling efforts.

These findings align with and build upon existing studies. For instance:

a. Vural and Koyuncu [9] emphasized DDD's role in modularity, which is corroborated here but expanded to highlight its impact on team collaboration.
b. Ntentos *et al.* [11] stressed the importance of reducing interdependencies, which this study integrates into a broader strategic framework.

The limitations of individual strategies underscore the need for further research:

a. The long-term scalability impacts of API gateway remain underexplored.
b. Tools for automating compensating transactions in the Saga Pattern are lacking.
c. The effectiveness of strategies like event sourcing in different contexts requires empirical validation.

Addressing these gaps is crucial for advancing the field. Future research should prioritize the development of standardized evaluation methods, empirical studies on scalability and performance, and practical tools to simplify strategy implementation.

4. CONCLUSION

This study investigated strategies to improve service coupling in MSA, identified the conditions under which these strategies are most effective, and analyzed research gaps requiring further exploration. The findings offer theoretical insights and practical recommendations for software architects and researchers, providing a roadmap for advancing MSA-based system design. The study identified ten critical strategies, including API Gateway, CQRS, DDD, saga pattern, and event sourcing. These strategies enhance modularity, scalability, and maintainability by addressing service coupling challenges. Each strategy’s effectiveness depends on specific design characteristics, such as high concurrency environments for CQRS or complex transaction scenarios for the Saga Pattern. The seven research gap framework revealed areas requiring further study, such as the long-term performance impacts of API gateway, the scalability limitations of event sourcing, and the lack of automated tools for compensating transactions in the Saga Pattern.

The findings hold immediate practical relevance for software architects, who can align strategies with system design characteristics to optimize coupling in MSA-based systems, enabling more scalable and maintainable architectures. For researchers, this study provides a structured foundation for exploring unresolved issues, including conducting empirical studies to validate the scalability and performance of coupling strategies, developing standardized evaluation metrics and tools for automating complex strategies and expanding the applicability of strategies to non-enterprise and emerging market contexts. While this study provides a comprehensive analysis of coupling strategies, its findings are primarily based on existing literature. Empirical validation of the proposed strategies and their integration into diverse industries remains limited. Addressing these gaps will confirm the generalizability and practical applicability of the findings.

To further advance the field, future research should prioritize developing automated tools and frameworks for simplifying the implementation of identified strategies, conducting longitudinal studies to evaluate the long-term scalability and performance of strategies like API gateway and event sourcing, and expanding the study of strategies’ impacts on team collaboration and communication, particularly in SMEs and emerging markets. By addressing these areas, the software engineering community can enhance the resilience, scalability, and maintainability of MSA-based systems. This study provides a critical foundation for such advancements, offering both theoretical contributions and practical solutions to pressing challenges in microservices design.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to Bina Nusantara University for providing the support and resources necessary to carry out this study. We also thank the reviewers for their insightful comments and suggestions, which have greatly contributed to improving the quality of this paper. Additionally, we acknowledge previous research efforts in the domain of microservices design and maintainability, which have provided a valuable foundation for this study. Lastly, we extend our appreciation to colleagues and collaborators who offered constructive feedback and technical assistance throughout the research process.

FUNDING INFORMATION

The authors state that no funding was involved in the conduct of this research. This study was carried out independently without financial support from any funding agency, institution, or organization.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Gintoro	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	
Sunardi		✓				✓		✓	✓	✓	✓	✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors state no conflict of interest. They declare that there are no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

DATA AVAILABILITY

The dataset used and analyzed in this study is publicly available on Zenodo. It can be accessed at the following link: https://doi.org/10.5281/zenodo.14462135. Researchers and practitioners can freely use the dataset under the terms of the applicable license.

REFERENCES

[1] P. P. Tallon, M. Queiroz, and T. Coltman, "Digital-Enabled Strategic Agility: The Next Frontier," *European Journal of Information Systems*, vol. 31, no. 6, pp. 641–652, Nov. 2022, doi: 10.1080/0960085X.2022.2102713.

[2] X. Chen, M. Usman, and D. Badampudi, "Understanding and evaluating software reuse costs and benefits from industrial cases—A systematic literature review," *Information and Software Technology*, vol. 171, pp. 1-22, Jul. 2024, doi: 10.1016/j.infsof.2024.107451.

[3] A. Lercher, "Managing API Evolution in Microservice Architecture," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, Lisbon Portugal: ACM, Apr. 2024, pp. 195–197, doi: 10.1145/3639478.3639800.

[4] D. R. F. Apolinário and B. B. N. De França, "A method for monitoring the coupling evolution of microservice-based architectures," *Journal of the Brazilian Computer Society*, vol. 27, no. 1, pp. 1-35, Dec. 2021, doi: 10.1186/s13173-021-00120-y.

[5] S. Panichella, M. Rahman, and D. Taibi, "Structural Coupling for Microservices:," in *Proceedings of the 11th International Conference on Cloud Computing and Services Science*, Science and Technology Publications, 2021, pp. 280–287, doi: 10.5220/0010481902800287.

[6] T. I. Ramdhani and N. N. Faiza, "National archival platform system design using the microservice-based service computing system engineering framework," *Computer Science and Information Technologies*, vol. 4, no. 2, pp. 95–105, Jul. 2023, doi: 10.11591/csit.v4i2.pp95-105.

[7] A. Lercher, J. Glock, C. Macho, and M. Pinzger, "Microservice API Evolution in Practice: A Study on Strategies and Challenges," *Journal of Systems and Software*, vol. 215, pp. 1-19, Sep. 2024, doi: 10.1016/j.jss.2024.112110.

[8] D. A. Miles, "A taxonomy of research gaps: Identifying and defining the seven research gaps," *Journal of Research Methods and Strategies*, pp. 1–15, 2017.

[9] H. Vural and M. Koyuncu, "Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?," *IEEE Access*, vol. 9, pp. 32721–32733, 2021, doi: 10.1109/ACCESS.2021.3060895.

[10] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Industry practices and challenges for the evolvability assurance of microservices: An interview study and systematic grey literature review," *Empirical Software Engineering*, vol. 26, no. 5, pp. 1-39, Sep. 2021, doi: 10.1007/s10664-021-09999-9.

[11] E. Ntentos, U. Zdun, K. Plakidas, S. Meixner, and S. Geiger, "Assessing architecture conformance to coupling-related patterns and practices in microservices," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12292 LNCS, 2020, pp. 3–20, doi: 10.1007/978-3-030-58923-3_1.

[12] B. Kitchenham *et al.*, "Systematic literature reviews in software engineering – A tertiary study," *Information and Software Technology*, vol. 52, no. 8, pp. 792–805, Aug. 2010, doi: 10.1016/j.infsof.2010.03.006.

[13] P. Müller-Bloch and J. Kranz, "A Framework for Rigorously Identifying Research Gaps in Qualitative Literature Reviews," *International Conference on Information Systems*, Dec. 2015.

[14] T. Cerny, A. S. Abdelfattah, A. A. Maruf, A. Janes, and D. Taibi, "Catalog and detection techniques of microservice anti-patterns

and bad smells: A tertiary study," *Journal of Systems and Software*, vol. 206, pp. 1-43, Dec. 2023, doi: 10.1016/j.jss.2023.111829.
[15] H. Michael Ayas, P. Leitner, and R. Hebig, "An empirical study of the systemic and technical migration towards microservices," *Empirical Software Engineering*, vol. 28, no. 4, pp. 1-50, Jul. 2023, doi: 10.1007/s10664-023-10308-9.
[16] D. Monteiro, P. H. M. Maia, L. S. Rocha, and N. C. Mendonça, "Building orchestrated microservice systems using declarative business processes," *SOCA*, vol. 14, no. 4, pp. 243–268, Dec. 2020, doi: 10.1007/s11761-020-00300-2.
[17] J. Soldani, G. Muntoni, D. Neri, and A. Brogi, "The μ TOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures," *Software: Practice and Experience*, vol. 51, no. 7, pp. 1591–1621, Jul. 2021, doi: 10.1002/spe.2974.
[18] R. Pinciroli, A. Aleti, and C. Trubiani, "Performance Modeling and Analysis of Design Patterns for Microservice Systems," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, L'Aquila, Italy: IEEE, Mar. 2023, pp. 35–46, doi: 10.1109/ICSA56044.2023.00012.
[19] Y. Zhong, W. Li, and J. Wang, "Using Event Sourcing and CQRS to Build a High Performance Point Trading System," in *Proceedings of the 2019 5th International Conference on E-Business and Applications*, Bangkok Thailand: ACM, Feb. 2019, pp. 16–19, doi: 10.1145/3317614.3317632.
[20] S. Lima, J. Correia, F. Araujo, and J. Cardoso, "Improving observability in Event Sourcing systems," *Journal of Systems and Software*, vol. 181, Nov. 2021, doi: 10.1016/j.jss.2021.111015.
[21] C. E. Da Silva, Y. D. L. Justino, and E. Adachi, "SPReaD: service-oriented process for reengineering and DevOps: Developing microservices for a Brazilian state department of taxation," *SOCA*, vol. 16, no. 1, pp. 1–16, Mar. 2022, doi: 10.1007/s11761-021-00329-x.
[22] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," *IEEE Access*, vol. 11, pp. 88339–88358, 2023, doi: 10.1109/ACCESS.2023.3305687.
[23] F. H. Vera-Rivera, E. Puerto, H. Astudillo, and C. M. Gaona, "Microservices Backlog–A Genetic Programming Technique for Identification and Evaluation of Microservices From User Stories," *IEEE Access*, vol. 9, pp. 117178–117203, 2021, doi: 10.1109/ACCESS.2021.3106342.
[24] D. R. Matos, M. L. Pardal, A. R. Silva, and M. Correia, "μVerum: Intrusion Recovery for Microservice Applications," *IEEE Access*, vol. 11, pp. 78457–78470, 2023, doi: 10.1109/ACCESS.2023.3298113.
[25] J. M. M. Correia, "Automated Identification of Monolith Functionality Refactorings for Microservices Migrations". 2021.
[26] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1651–1681, Sep. 2020, doi: 10.1002/spe.2869.
[27] S. Chang, "Software Design Pattern Analysis for Micro-Service Architecture using Queuing Networks (S)," in *33rd International Conference on Software Engineering and Knowledge Engineering*, Jul. 2021, pp. 45–50, doi: 10.18293/SEKE2021-180.

BIOGRAPHIES OF AUTHORS

Gintoro is a lecturer from Binus University, Indonesia's School of Computer Science. He received his Computer Science degree from Bina Nusantara University in 1998. He also received a Master of Information System degree from Bina Nusantara University, Jakarta, in 2001. He currently serves as Educational Services Director at BINUS University, leading two sub-business units: Sokrates Empowering School and BINUS Center. His research interests include software engineering, software architecture, AI, educational technology, and implementing technology in teaching and learning. He can be contacted at email: gintoro@binus.ac.id.

Sunardi works as a lecturer at Binus Online Learning, which is part of Binus University in Indonesia. He received his Computer Science degree from Bina Nusantara University in 2004. He also received the Master's degree in Master of Information System from Bina Nusantara University, Jakarta, in 2012. Currently, he is the CEO of a company called Sundu.id, a startup company that provides immersive eLearning content and LMS. His research interests include user experience, usability evaluation, ARVR, immersive learning, and the metaverse area. He can be contacted at email: sunardi@binus.ac.id.