International Journal of Scientific Research & Engineering Trends
Volume 14, Issue 2, Mar-Apr-2026, ISSN (Online): 2395-566X

Challenges in Adopting Microservices Architecture: A Systematic Review of Data Consistency and Fault Tolerance

[1]Devang Sethi, [2]Dr. Rajat Takkar
Department of Computer Science and Engineering Chitkara University Patiala, India

Abstract- Microservices architecture has gained significant attention as a dominant paradigm for building scalable and cloud-native applications by decomposing monolithic systems into independently deployable services with decentralized data ownership. However, this architectural approach introduces challenges related to distributed data management and system reliability. This paper presents a systematic literature review examining data consistency and fault tolerance mechanisms in microservices environments. The study analyzes research published between 2016 and 2026 collected from major academic databases including IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, Google Scholar, and arXiv. The findings indicate that strict consistency models often limit system scalability and availability, leading many architectures to adopt eventual consistency and BASE principles. Saga-based transaction management patterns are increasingly preferred over traditional Two-Phase Commit protocols due to improved resilience, although they introduce additional implementation complexity. The review also highlights the lack of standardized evaluation frameworks for benchmarking distributed resilience strategies. Overall, the study emphasizes the importance of balancing consistency, scalability, and fault tolerance when designing reliable microservices-based systems.

Keywords – Microservices Architecture, Data Consistency, Fault Tolerance, Distributed Systems, Systematic.

I. INTRODUCTION

Microservices architecture has emerged as a widely adopted approach for developing scalable and flexible software systems, particularly in cloud-native and large-scale distributed environments. While microservices offer significant benefits in terms of modularity, independent deployment, and scalability, their distributed nature introduces complex architectural challenges that are not present in traditional monolithic systems.

Among the most critical challenges in microservices- based systems are data consistency and fault tolerance. The absence of centralized data management and the reliance on network-based communication make it difficult to maintain consistent system states across services. At the same time, failures such as service crashes, network partitions, and infrastructure outages are inevitable in distributed environments and must be handled effectively to prevent cascading failures.

Despite extensive research and industrial adoption, there is no universally optimal solution for achieving both strong data consistency and high fault tolerance in microservices architectures. Existing approaches involve trade-offs between consistency, availability, and system resilience, and their effectiveness varies depending on system requirements and operational constraints. This review paper aims to systematically examine existing literature on data consistency and fault tolerance in microservices architecture, analyze current challenges and solutions, and identify open research gaps and future research directions.

Problem Statement

Despite extensive adoption of microservices architecture, maintaining data consistency across distributed services remains a significant challenge. Operations that span multiple services often require coordination among independent databases, making traditional transaction management mechanisms difficult or impractical to apply. Approaches such as eventual consistency and compensating transactions introduce additional complexity and require careful system design.

At the same time, fault tolerance is a critical concern in microservices-based systems. Failures are inevitable in distributed environments, including service crashes, network partitions, and infrastructure outages. Without proper fault tolerance mechanisms, such failures can propagate across

services, leading to cascading failures and system-wide disruptions.

The inherent trade-offs between consistency, availability, and fault tolerance further complicate architectural decisions. There is no universally optimal solution, and existing approaches vary significantly in their effectiveness depending on system requirements and operational constraints.

II. BACKGROUND

Background and Motivation

Modern software systems are increasingly required to support scalability, high availability, and rapid evolution while serving large and geographically distributed user bases. To meet these demands, organizations have progressively shifted from traditional monolithic architectures to microservice architectures, where applications are decomposed into small, independently deployable services that communicate over lightweight protocols.

Microservices architecture offers several advantages, including improved scalability, flexibility, and faster development cycles. Independent service deployment enables teams to adopt diverse technologies and evolve services without impacting the entire system. As a result, microservices have been widely adopted in cloud-native systems, large-scale web applications, and enterprise platforms.

However, these benefits come at the cost of increased architectural complexity. The decentralized and distributed nature of microservices introduces new challenges that are not present in monolithic systems. Among these challenges, data consistency and fault tolerance have emerged as two of the most critical and difficult problems to address in real- world deployments.

Micro services Architecture Overview

In micro services architecture, each service is designed around a specific business capability and typically maintains its own data store. This principle of decentralized data management promotes service autonomy and scalability but eliminates the possibility of enforcing global consistency through a centralized database.

Services interact through synchronous communication (such as REST APIs) or asynchronous messaging mechanisms (such as message queues or event streams). While this loose coupling improves flexibility, it also increases susceptibility to network latency, partial failures, and inconsistent system states. Consequently, system designers must explicitly address failure handling and data coordination across services.

The absence of global transactions and the reliance on network-based communication distinguish microservices systems from traditional architectures and fundamentally reshape how consistency and reliability are achieved.

Data Consistency and Fault Tolerance in Distributed Systems

In distributed systems, maintaining data consistency and ensuring fault tolerance are fundamental challenges. In microservices architectures, these challenges are intensified due to decentralized data ownership, network communication, and independent service failures, requiring explicit architectural strategies to manage reliability and correctness.

III. RESEARCH METHODOLOGY

This study follows a systematic literature review (SLR) approach to identify, analyze, and synthesize existing research related to data consistency and fault tolerance in microservices architecture. A structured and transparent review protocol was adopted to ensure rigor, reproducibility, and comprehensive coverage of relevant academic literature.

Review Scope

The scope of this literature review includes:

- To identify key challenges related to data consistency in microservices-based systems
- To analyze fault tolerance issues and mechanismsreported in existing studies
- To compare architectural patterns and strategies addressing these challenges
- To identify research gaps and open challenges in the current literature

Data Sources

Relevant literature was collected from several academic databases including Google Scholar, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and arXiv. These sources were selected because they host peer-reviewed journal articles, conference papers, and high-quality systematic reviews related to software architecture and distributed systems.

Search Strategy

A keyword-based search strategy was employed to retrieve relevant studies. The search terms were designed to capture variations of microservices, data consistency, and fault tolerance concepts. The primary search queries included combinations of the following keywords:

- microservices architecture
- data consistency in microservices
- fault tolerance in microservices
- distributed transactions
- Saga pattern

- resilience in microservices
- distributed systems failures

Boolean operators (AND, OR) were used to refine the search results and improve relevance.

Inclusion and Exclusion Criteria

To ensure relevance and quality, studies were selected based on the following criteria:

Inclusion Criteria

- Studies published between 2016 and 2026
- Peer-reviewed journal articles, conference papers, and systematic reviews
- Research focusing on microservices architecture
- Studies addressing data consistency, distributed transactions, or fault tolerance
- Empirical studies, surveys, and architectural analyses

Exclusion Criteria

- Studies focusing solely on monolithic architectures
- Papers unrelated to consistency or fault tolerance
- Non-academic blog posts, tutorials, or opinion pieces
- Short papers with insufficient technical depth
- Duplicate or superseded studies

Study Selection Process

The initial search across all databases returned a large number of publications spanning microservices architecture and distributed systems. Titles and abstracts were first screened to remove irrelevant studies. The remaining papers were then reviewed in full text to assess their relevance based on the inclusion and exclusion criteria.

After this filtering process, a focused set of high-quality and relevant studies was selected for detailed analysis. The final selection included systematic reviews, empirical studies, and architectural evaluations that directly addressed data consistency and fault tolerance challenges in microservices. The study selection process followed a structured and iterative screening approach inspired by PRISMA guidelines to ensure transparency and minimize selection bias.

Data Extraction and Analysis

From each selected study, key information was extracted, including:

- Research focus and objectives
- Identified data consistency challenges
- Fault tolerance mechanisms and patterns
- Architectural approaches and trade-offs
- Limitations and future research directions

The extracted data were then synthesized qualitatively. Rather than summarizing individual papers in isolation, the review focused on identifying recurring themes, common challenges, and comparative insights across multiple studies. This synthesis approach enables a holistic understanding of the state of research and practice.

Threats to Validity

This literature review may be subject to certain limitations. First, despite using multiple databases, some relevant studies may have been missed due to keyword selection or publication accessibility. Second, the review relies on the accuracy and completeness of the original studies. To mitigate these risks, multiple sources and diverse study types were included, and emphasis was placed on peer- reviewed and widely cited research.

IV. LITERATURE REVIEW

This section synthesizes existing literature on data consistency and fault tolerance in microservices architecture, focusing on recurring challenges, architectural patterns, and solution strategies reported across prior studies.

Data Consistency in Microservices

Microservices architecture promotes decentralized data management by allowing each service to own and manage its own database. While this design improves scalability and service autonomy, it introduces significant challenges in maintaining data consistency across distributed services [2], [4].

One of the primary challenges arises from distributed data ownership. Business operations often span multiple services, requiring updates across several independent databases, which complicates coordination and recovery in the presence of failures [2], [3].

A major consistency dilemma in microservices stems from the trade-off between strong consistency and availability, as described by the CAP theorem [3], [5]. Many microservices-based systems therefore, adopt BASE consistency models rather than strict ACID transactions to improve scalability and resilience [2].

Eventual consistency is widely adopted in practice, but it introduces issues such as stale reads, write conflicts, and increased application-level complexity [3], [6]. Developers must design workflows that tolerate temporary inconsistencies while ensuring eventual correctness.

Distributed transaction management remains a significant challenge. Classical mechanisms such as Two-Phase Commit (2PC) are often avoided due to their blocking nature and impact on system availability [3], [5]. The Saga pattern has emerged as an alternative, enabling long-running transactions through a sequence of local operations with compensating actions, though at the cost of higher implementation complexity [5], [7].

Fault Tolerance Challenges in Microservices Architecture

Fault tolerance is a critical requirement in microservices architectures due to their distributed and loosely coupled nature [1], [6]. Failures such as service crashes, network delays, and resource exhaustion are inevitable and must be handled gracefully.

A key challenge is the occurrence of partial failures, where some services fail while others remain operational. Such failures can lead to cascading effects if dependent services repeatedly attempt communication with unavailable components [6], [7].

Inter-service communication over networks increases susceptibility to latency and transient errors. Without proper safeguards, retry mechanisms can amplify failures rather than resolve them [6].

Service dependency complexity further complicates fault tolerance. As systems grow, understanding and managing dependencies becomes increasingly difficult, making fault isolation and recovery more challenging [4], [8].

Fault Tolerance Mechanisms and Solutions

The literature highlights that fault tolerance in microservices is best achieved through architectural and design patterns rather than centralized recovery mechanisms [1], [6].

The Circuit Breaker pattern is widely recognized as an effective mechanism for preventing cascading failures by temporarily blocking calls to unhealthy services [6], [7]. Empirical studies report improved availability and reduced failure propagation when circuit breakers are combined with monitoring and health checks [7].

Retry and timeout mechanisms are commonly used to handle transient faults, but improper configuration can overload systems and worsen outages [6]. Adaptive retry strategies with exponential backoff are therefore recommended [7].

Service replication and load balancing increase availability by ensuring that multiple service instances can handle requests during failures [1]. Asynchronous, message- driven communication further improves resilience by decoupling services and enabling continued operation during partial outages [2], [6].

V. CHALLENGES AND RESEARCH GAPS

Despite significant progress in research on data consistency and fault tolerance in microservices architecture, several challenges remain insufficiently addressed. The literature reveals that many proposed solutions focus on specific architectural patterns or failure scenarios, while broader system-level concerns and practical deployment issues continue to persist.

One major challenge is the lack of standardized evaluation frameworks for assessing data consistency and fault tolerance mechanisms. Existing studies often evaluate proposed approaches using custom benchmarks, simulation environments, or limited case studies, making it difficult to compare results across different systems. This inconsistency in evaluation limits the ability to objectively assess trade-offs between consistency, availability, and fault tolerance in real-world deployments.

Another unresolved issue is the operational complexity introduced by consistency and resilience mechanisms. Techniques such as Saga-based transactions, circuit breakers, and compensating actions improve fault tolerance but significantly increase system design, testing, and maintenance complexity. The literature highlights that implementing these mechanisms correctly requires substantial expertise, and misconfigurations can lead to data inconsistencies or cascading failures rather than preventing them.

Scalability under failure conditions also remains an open challenge. While many approaches perform well under normal workloads, fewer studies analyze system behavior during large-scale failures, high traffic spikes, or correlated service outages. The interaction between fault tolerance mechanisms and system scalability is often insufficiently explored, particularly in highly dynamic cloud-native environments.

Additionally, there is a noticeable gap between academic research and industrial practice. Many studies propose theoretical models or controlled experimental setups, but limited empirical evidence exists regarding their adoption and effectiveness in production systems. Real-world constraints such as cost, legacy integration, and organizational factors are rarely considered in depth.

Finally, emerging techniques such as AI-driven fault prediction, automated recovery, and self-healing microservices are still in early stages of research. While these approaches show promise in reducing manual intervention and improving system resilience, they lack comprehensive validation and standardized methodologies.

Overall, the literature indicates that although numerous strategies exist to address data consistency and fault tolerance in microservices architecture, significant research gaps remain. Addressing these gaps is essential to developing reliable, scalable, and maintainable microservices-based systems [1], [25], [27].

VI. COMPARATIVE ANALYSIS: CONSISTENCY AND FAULT TOLERANCE TRADE-OFFS

The reviewed literature highlights that achieving data consistency and fault tolerance in microservices architecture involves inherent trade-offs between consistency guarantees, system availability, scalability, and resilience [2], [3], [5].

Aspect	Strong Consistency	Eventual Consistency
Data correctness	Immediate	Temporary inconsistencies
Availability	Lower	Higher
Scalability	Limited	High
Fault tolerance	Weaker under failures	Better resilience
Implementation complexity	High (coordination required)	High (application-level handling)

Transaction Model	Advantages	Limitations
Two-Phase Commit (2PC)	Atomicity guarantees	Blocking, failure-prone
Saga Pattern	Fault-tolerant, scalable	Complex compensation logic

Fault Tolerance Technique	Strengths	Weaknesses
Retries & Timeouts	Simple	Can amplify failures
Circuit Breakers	Prevent cascading failures	Configuration overhead
Replication	High availability	Resource cost

Table I. Comparison of Data Consistency and Fault Tolerance Approaches in Microservices

A key comparison discussed in the literature is between strong consistency and eventual consistency models. Strong consistency ensures immediate correctness of data across services but often relies on coordination mechanisms that reduce system availability and scalability [3], [5].

In contrast, eventual consistency improves system resilience and performance by allowing temporary inconsistencies, but it shifts complexity to the application layer, requiring developers to handle stale data and conflict resolution explicitly [2], [6]. Several studies report that large-scale microservices systems favor eventual consistency due to its alignment with fault-tolerant and highly available designs [2], [25].

Another important comparison concerns distributed transaction management approaches. Traditional Two-Phase Commit (2PC) protocols provide strong atomicity guarantees but introduce blocking behavior and are highly sensitive to failures, making them unsuitable for highly distributed microservices environments [3], [11], [15].

In contrast, Saga based transaction patterns decompose transactions into a sequence of local operations with compensating actions, improving fault tolerance and system availability [5], [9], [13]. However, Saga implementations increase design complexity and require careful handling of failure scenarios, as compensation logic must be explicitly defined [9], [10].

Fault tolerance mechanisms also exhibit trade-offs in their effectiveness. Reactive approaches such as retries and timeouts are simple to implement but may amplify failures if misconfigured [6]. Proactive mechanisms such as circuit breakers, bulkheads, and service isolation are more effective in preventing cascading failures but introduce additional operational and configuration overhead [1], [7], [18]. Empirical studies indicate that systems combining multiple resilience patterns achieve better fault isolation and recovery than those relying on a single mechanism [1], [8].

Overall, the literature suggests that no single approach optimally addresses both data consistency and fault tolerance. Instead, effective microservices architectures adopt a combination of consistency models, transaction patterns, and fault tolerance mechanisms tailored to specific system requirements and operational constraints [2], [25].

VII. CONCLUSION

Microservices architecture has become a widely adopted approach for building scalable, flexible, and cloud-native software systems. However, the distributed and decentralized nature of microservices introduces significant challenges related to data consistency and fault tolerance, which are critical to ensuring system reliability and correctness. This review paper systematically examined existing research addressing these challenges, with a focus on architectural patterns, transaction management strategies, and resilience mechanisms used in microservices-based systems.

The literature indicates that maintaining data consistency in microservices often requires trade-offs between strong consistency guarantees and system availability. Approaches such as eventual consistency and Saga-based transaction patterns are commonly favored in practice due to their alignment with scalable and fault-tolerant system designs, despite the additional complexity they introduce. Similarly, fault tolerance in microservices is primarily achieved through

design patterns such as circuit breakers, retries, service replication, and asynchronous communication, which aim to prevent cascading failures and improve system resilience.

Through comparative analysis, this review highlights that no single solution effectively addresses all consistency and fault tolerance requirements. Instead, effective microservices architectures rely on a combination of strategies tailored to specific system requirements, operational constraints, and failure scenarios. The analysis also reveals several open challenges, including the lack of standardized evaluation frameworks, high operational complexity, limited large-scale empirical validation, and the gap between academic research and industrial practice.

Overall, this review provides a structured understanding of current approaches to data consistency and fault tolerance in microservices architecture and identifies key research gaps that require further investigation. Addressing these challenges is essential for the development of reliable, scalable, and maintainable microservices-based systems in increasingly complex distributed environments.

REFERENCES

1. M. Mohammad, "Resilient Microservices: A Systematic Review of Recovery Patterns, Strategies, and Evaluation Frameworks," arXiv preprint arXiv:2512.16959, 2025.
2. R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, and M. Kalinowski, "Data Management in Microservices: State of the Practice, Challenges, and Research Directions," Proceedings of the VLDB Endowment (PVLDB), vol. 14, no. 13, pp. 3348–3361, 2021, doi:10.14778/3484224.3484232.
3. "Current Trends in the Management of Distributed Transactions in Micro-Services Architectures: A Systematic Literature Review," SCIRP, 2024.
4. "Microservices Architecture: Evolution, Realizing Benefits, and Addressing Challenges," Arab Journals Platform, 2023.
5. "Microservices Security: A Systematic Literature Review," PMC / NCBI, 2022.
6. H. Gopal et al., "Security, Privacy and Challenges in Microservices Architecture and Cloud Computing," arXiv preprint arXiv:2212.14422, 2022.
7. F. Ponce et al., "Microservices Testing: A Systematic Literature Review," ScienceDirect, 2021.
8. M. Waseem et al., "Understanding the Issues, Their Causes and Solutions in Microservices Systems: An Empirical Study," arXiv preprint arXiv:2302.01894, 2023.
9. "An Evaluation of Saga Pattern Implementation Technologies," CEUR Workshop Proceedings, vol. 2839, 2021.
10. "Analysis of Distributed Transaction Patterns in Microservices," TalTech Digital Repository, 2020.
11. "Distributed Transactions in Microservice Architecture," Lviv Polytechnic National University Scientific Journal, 2019.
12. G. Zhang et al., "GRIT: Consistent Distributed Transactions Across Polyglot Microservices and Databases," IEEE, 2020.
13. X. Limón et al., "SagaMAS: A Software Framework for Distributed Transactions in Microservices," IEEE, 2019.
14. "Supporting Transparent Distributed Transactions Among Containerized RESTful Services," IEEE, 2022.
15. T. Janssen, "Distributed Transactions: Don't Use Them for Microservices," Industry Perspective, 2018.
16. B. Barua and M. S. Kaiser, "A Microservices Approach to Fault Tolerance, Load Balancing, and Service Discovery," arXiv preprint arXiv:2410.19701, 2024.
17. H. H. Addeen, "A Dynamic Fault Tolerance Model for Microservices Architecture," ProQuest Dissertation, South Dakota State University, 2021.
18. "Architectural Patterns for Designing Fault-Tolerant Microservice Systems," UL Open Access Journal, vol. 3, no. 1, 2023.
19. "Fault Tolerance and Resilience in Microservices-Based Systems," International Journal of Novel Research and Development (IJNRD), 2023.
20. "Model-Based Resilience Pattern Analysis for Microservices," Journal of Advanced Trends in Information Technology (JATIT), 2022.
21. G. Shekhar, "Microservices Design Patterns for Cloud Architecture," IEEE, 2019.
22. "Insights on Microservice Architecture Through the Eyes of Practitioners," arXiv preprint arXiv:2408.10434, 2024.
23. "Adopting and Sustaining Microservice-Based Software Development," ACM Communications, 2020.
24. "Challenges in Adopting and Sustaining Microservice-Based Software Development," ACM Queue, 2019.
25. M. Viggiato et al., "Microservices in Practice: A Survey Study," Empirical Software Engineering, 2020.
26. J. Bogner et al., "Assuring the Evolvability of Microservices," arXiv preprint arXiv:1906.05013, 2019.
27. "A Benchmark for Data Management in Microservices,"ACM, 2024.
28. "Online Marketplace: A Microservice Benchmark,"ACM, 2024, doi:10.1145/3709653.
29. G. Vale et al., "Designing Microservice Systems Using Patterns: An Empirical Study," arXiv preprint arXiv:2201.03598, 2022.
30. "A Systematic Literature Review on Fault Injection Testing of Microservice Systems," IEEE,2023.