Universal Library of Innovative Research and Studies Research Article

ISSN: 3065-0003 | Volume 3, Issue 2

Open Access | PP: 117-125

DOI: https://doi.org/10.70315/uloap.ulirs.2026.0301016

Ensuring Data Integrity in Distributed Microservices Ecosystems: A Comparative Analysis of Architectural Patterns

Gleb Shkriabin

CTO at White Code Works LLC, New Jersey, USA.

Abstract

Microservice architecture is widely used in the development of distributed information systems; however, dividing an application into independent services significantly complicates the problem of ensuring data integrity. Unlike monolithic systems, where consistency is maintained within a single transaction and a single data store, in microservice architectures a single operation may involve multiple services and databases. This creates the need for specialized architectural mechanisms to coordinate distributed operations. The article provides a comparative analysis of the main architectural patterns used to ensure data consistency in microservice systems, including the two-phase commit protocol, the model of sequential local transactions with compensating actions, and event-driven service coordination. The study is conducted in the form of a systematic analysis of recent scientific publications on distributed system architecture. Data storage models, mechanisms of transactional coordination, and operational constraints of high-load distributed platforms are examined. The results show that the effectiveness of architectural patterns depends on system characteristics, including workload intensity, consistency requirements, the level of contention for shared state, and the architectural complexity of service interactions. Based on the analysis, an architectural model for selecting data integrity mechanisms in microservice systems is proposed. The study demonstrates that the resilience of distributed platforms is achieved through a combination of transactional coordination, compensating operations, protection against duplicate message processing, and advanced observability mechanisms. The proposed model can be applied in the design of high-load microservice platforms and in the selection of architectural solutions for distributed systems.

Keywords: *Microservices; Distributed Transactions; Data Consistency; Saga Pattern; Two-Phase Commit; Event-Driven Architecture; Distributed Systems Architecture.*

INTRODUCTION

Microservice architecture is widely applied in the development of distributed information systems. Within this architecture, an application consists of a set of small services, each fulfilling a distinct function and operating independently of the others (Singjai & Zdun, 2022). However, partitioning a system into autonomous services introduces new architectural challenges. In monolithic applications, data is typically stored in a single database and operations are executed within a single transaction, thereby simplifying the control of data correctness. Under a service-oriented architecture, each component manages its own repository, meaning that numerous operations are executed across multiple services simultaneously. Maintaining data consistency within such a framework becomes a complex undertaking.

Traditional transactional mechanisms, grounded in ACID transactions within centralized database management systems where operations are executed within a single transactional boundary and committed atomically, ensure strict data consistency but are poorly suited for distributed systems. These mechanisms require centralized coordination and can degrade performance as workloads increase (De Heus et al., 2022). Consequently, an alternative approach is frequently adopted in service architectures. Consistency is achieved gradually through a sequence of local operations and message exchanges between services (Mohammad, 2025). This introduces new risks. An operation might be executed multiple times upon repeated delivery of an event. A distributed operation could complete only partially if a single service encounters an error. Additionally, a message between services might be lost. As a result, the business rules of the system can be violated (Narváez et al., 2025).

These issues are particularly pronounced in high-load systems where services actively exchange events. Asynchronous interaction reduces component coupling and helps the system withstand heavy loads. Yet, controlling the data state

Citation: Gleb Shkriabin, "Ensuring Data Integrity in Distributed Microservices Ecosystems: A Comparative Analysis of Architectural Patterns", Universal Library of Innovative Research and Studies, 2026; 3(2): 117-125. DOI: https://doi.org/10.70315/uloap.ulirs.2026.0301016.

becomes more complicated. Events may arrive with a delay, repeat, or be processed in a different order (Mohammad, 2025). Thus, ensuring data integrity transitions into a task for the entire system architecture rather than an individual transaction.

The scientific literature has proposed several solutions to the problem of ensuring data consistency in distributed service systems. These include the two-phase commit protocol, the Saga pattern, Command Query Responsibility Segregation (CQRS), alongside event-driven data storage and distribution models (Söylemez et al., 2022). Certain of these approaches are predominantly applied to organize read streams and optimize data access; they do not, however, directly resolve the task of coordinating distributed transactions across services.

The most widely discussed mechanisms for providing data consistency in a microservice architecture remain the two-phase commit, the model of sequential local operations with compensating actions (Saga), and event-driven service coordination. These approaches enable the execution of distributed operations between services, though most studies consider them in isolation and rarely analyze their applicability under the conditions of high-load distributed systems.

Despite a substantial volume of research in the domain of microservice architecture, the academic literature more frequently analyzes isolated aspects of data consistency provision—such as distributed transactions, event-driven interaction models, or data storage architecture. Concurrently, the combined impact of the data storage architecture, transaction coordination mechanisms, and system operational characteristics on data integrity is examined far less often. The absence of such an integrated approach complicates the selection of architectural solutions when designing high-load distributed systems.

Building on this, a practical problem emerges. Currently, there is no unified architectural model that allows for the selection of data integrity mechanisms while accounting for the specific features of a given microservice system and its operating conditions.

The objective of this study is to develop an architectural model for selecting data integrity patterns in microservice systems, taking into account the constraints of high-load distributed platforms. To achieve this objective, the following tasks are addressed:

- systematize the architectural mechanisms for ensuring data consistency in microservice systems;
- identify the tradeoffs between data consistency, scalability, and system performance;
- develop a model for selecting architectural data integrity patterns depending on the characteristics of the distributed system.

The scientific novelty of the research lies in the development of an architectural model for selecting data integrity mechanisms in microservice systems while incorporating the operational characteristics of the distributed platform. Unlike existing studies, this work examines the interrelationship between the data storage architecture, distributed transaction coordination mechanisms, and system load parameters.

The hypothesis is as follows. It is posited that in high-load distributed systems, optimal results are achieved through the joint utilization of transaction coordination mechanisms, compensating operations, protection against message reprocessing, and system observability tools. This combination enhances system resilience and maintains data correctness even under heavy load and asynchronous service interaction.

MATERIALS AND METHODS

The research methods employed include theoretical synthesis of scientific publications on the architecture of service-oriented distributed systems, structural systematization of data consistency mechanisms, comparative analysis of transaction coordination patterns, and interpretation of empirical results from studies on the performance and resilience of distributed platforms. The application of these methods enabled the identification of robust architectural approaches for ensuring data integrity and allowed them to be examined as interconnected elements of a distributed system's architecture.

The study was conducted in the format of a systematic review analysis of open-access publications from 2022 to 2026, presented in international peer-reviewed journals and academic repositories. Literature searches were carried out across international scientific databases and digital libraries such as Google Scholar, IEEE Xplore, ACM Digital Library, ScienceDirect, and SpringerLink, as well as in the open academic repositories arXiv and MDPI. The search strategy relied on combinations of keywords: "microservices architecture", "distributed transactions", "data consistency microservices", "polyglot persistence", "saga pattern", "event-driven architecture", "microservices monitoring", "microservices orchestration", "serverless transactions", and "microservices design patterns", utilizing the logical operators AND/OR.

During the identification phase, 47 publications were discovered. Following the removal of duplicates and an initial screening of titles and abstracts, works unrelated to the architecture of service-oriented distributed systems or the data consistency problem were excluded. At the full-text evaluation stage, studies of a narrowly algorithmic focus or those lacking architectural and applied conclusions were further omitted. The final sample encompassed 13 studies.

Inclusion criteria comprised the analysis of service system architectures, the investigation of distributed transactions and data consistency models, the description of service coordination mechanisms and interaction patterns, and the evaluation of the operational characteristics of distributed platforms. Publications that did not address the architectural

organization of such systems or lacked reproducible methodological or practical results were excluded.

The study by Cortellessa et al. (2022) proposed a model-driven approach to managing the performance of distributed service systems. De Heus et al. (2022) examined the execution of transactions within distributed computing functions. Halili et al. (2025) analyzed the application of polyglot data persistence. Indykov et al. (2025) systematized architectural tactics for ensuring the qualitative characteristics of software systems. Mohammad (2025) presented an overview of resilience and recovery strategies for distributed platforms. Morabito et al. (2026) proposed a model for distributed service orchestration. Narváez et al. (2025) investigated the application of artificial intelligence methods in designing service architecture. Rossetto et al. (2024) explored solutions for monitoring distributed platforms. Singjai and Zdun (2022) suggested a method for evaluating architectural decisions when designing service interfaces. Söylemez et al. (2022) provided a review of service architecture practices. Suljkanović et al. (2022) developed a domain-specific language for building service applications. Waseem et al. (2022) offered a model for selecting architectural patterns. Zouani and Lachgar (2024) introduced a model-driven approach to the development of service applications.

The selected corpus of publications facilitated the identification of key directions in ensuring data integrity: data storage and distribution architecture, distributed transaction coordination, service interaction patterns, monitoring and observability mechanisms, and models for selecting architectural solutions when designing distributed platforms. The obtained findings are utilized for the subsequent interpretation of the architectural model for data integrity provision in high-load systems.

RESULTS

The transition to a microservice architecture alters the fundamental logic of ensuring data integrity. Whereas in a monolithic system consistency is typically maintained within a single database, in a distributed service environment a single operation involves multiple services, multiple repositories, and several stages of inter-service coordination. For this reason, the analysis of architectural mechanisms must commence with the data storage model before proceeding to distributed transaction patterns.

Within a microservice architecture, data ceases to be stored in a single centralized boundary and is distributed among services. Each service assumes ownership of its own repository and domain of responsibility (Halili et al., 2025). Such an arrangement enhances service autonomy and simplifies the independent development of distinct system parts. Simultaneously, it renders data consistency a more formidable challenge, as a single business operation begins to affect multiple services and datastores. Consequently, the maintenance of integrity shifts from the level of a single database to the overarching architecture of service interactions (Söylemez et al., 2022).

One implication of data decentralization is the use of diverse database types within a single distributed system. For operations related to payments and order processing, relational databases that guarantee strict transactional consistency are typically employed. At the same time, product catalogs, user profiles, search indices, and ephemeral data may be housed in document-oriented or distributed repositories. As a result, disparate data consistency regimes and varying methods of committing state changes are utilized within the same system (Halili et al., 2025).

When employing different types of databases, the distinctions between them manifest at the level of technology and system architecture. Certain repositories are oriented toward strict transactional data consistency, while others prioritize high scalability and tolerate temporary state discrepancies between services. Therefore, the task of ensuring data integrity in a microservice architecture cannot be resolved uniformly across all system components. Selecting a datastore necessitates consideration of the nature of data access, the type of operations executed, and the acceptable complexity of inter-service coordination. Table 1 outlines the consistency models and the support for transactional guarantees in databases that are most frequently applied in distributed service systems.

Table 1. Consistency Models and ACID Support in Different Databases (Compiled by the author based on source: (Halili et al., 2025))

Database	Consistency Model	ACID Support
PostgreSQL	Strong consistency	Full
MongoDB	Configurable consistency	Partial / full since version 4.0
Cassandra	Eventual + configurable	No
Redis	Basic transactional operations	No
Neo4j	Strong consistency	Full
Elasticsearch	Eventual consistency	No

The table demonstrates that relational and graph databases provide a more rigorous consistency regime within the local boundary of a service, whereas Cassandra and Elasticsearch operate on the logic of eventual state consistency. MongoDB occupies an intermediate position, permitting the configuration of the consistency level depending on the

use case. Redis, in turn, is frequently deployed as a rapid intermediate access and caching layer rather than a primary mechanism for preserving state invariants. This distinction is critical because, in a distributed system, a single business operation may traverse services that utilize fundamentally different models for committing changes.

Strict consistency implies that upon the completion of an operation, all participants in the system must observe an identical data state. This regime performs well within a confined local boundary, where state modification is governed by a single repository or a small number of tightly coupled participants (De Heus et al., 2022). However, sustaining such a regime in a distributed service environment mandates coordination across multiple services (Söylemez et al., 2022). Thus, as the number of services and inter-service steps expands, the cost of strict consistency begins to escalate rapidly.

The eventual data consistency model is constructed on a different logic. The system concedes that at various points in time, services may harbor divergent data states. This is a temporary discrepancy. Upon the conclusion of a sequence of local operations and message exchanges, the data state aligns. This regime diminishes service coupling and facilitates system scaling. Yet, it simultaneously demands supplementary control mechanisms. Protection against the repeated execution of operations is necessary. Governing the order of message processing is critical. In certain instances, compensating actions are employed in the event of failures (Mohammad, 2025). Consequently, the distinction between strict consistency and eventual data consistency relates not solely to the speed of information updating. It concerns the selection of the system's architectural logic: rigid operation coordination versus managed asynchronous service interaction (Söylemez et al., 2022). Choosing this architectural logic concurrently dictates the set of mechanisms through which change consistency is sustained in the system.

Data integrity in distributed service systems is upheld by several architectural mechanisms, each addressing a specific portion of the change reconciliation task. A subset of these relies on transactional coordination. In this scenario, consistency is achieved by aligning changes across multiple services and their respective repositories (De Heus et al., 2022). Another approach is structured around an event-driven model. The system state materializes as events are published and sequentially processed. If the chain of operations is disrupted, compensating actions are invoked (Suljkanović et al., 2022). Mechanisms preventing operation re-execution occupy a distinct role. In distributed systems, messages can be delivered repeatedly; therefore, a single operation is occasionally triggered multiple times, necessitating specialized constraints and validations (Mohammad, 2025). Finally, system observability tools emerge as a crucial component. Without request tracing, event correlation, and distributed log analysis, it is impossible to accurately discern how the system arrived at its current data state.

This classification enables the conceptualization of data integrity not as an attribute of an isolated transaction, but as the outcome of multiple interacting architectural layers (Waseem et al., 2022). Should the system employ heterogeneous repositories, transactional tools alone are insufficient. If service interaction is constructed asynchronously, an event-driven model proves inadequate without idempotency and deduplication. Alternatively, if an operation traverses a lengthy inter-service chain, correlation identifiers and execution observability acquire critical significance (Rossetto et al., 2024). Thereby, the architecture of integrity is formulated not by a solitary pattern, but through a confluence of interconnected mechanisms.

Traditional distributed transactions are geared toward scenarios where all participants can be strictly coordinated and confirm state modifications synchronously (De Heus et al., 2022). Within a microservice architecture, this paradigm encounters limitations. This is predicated on the fact that a single operation is distributed across autonomous services. These services, in turn, possess disparate repositories, fault-tolerance modes, and load profiles (Söylemez et al., 2022). Consequently, the transactional mechanism becomes contingent on business logic, the slowest participant, and the cost of inter-service coordination. This is particularly evident in high-load systems, where an escalation in concurrent operations rapidly inflates the cost of rigid synchronization.

Data consistency in distributed service systems is most commonly provisioned by three architectural approaches: the two-phase commit, the model of sequential local operations with compensating actions, and coordination via events. The two-phase commit mechanism is structured around a centralized coordinator. Initially, participants affirm their readiness to execute changes. Subsequently, the final commitment of operations is performed. This mechanism upholds strict atomicity and change isolation. However, it relies on locks to do so. Hence, elevated contention for data escalates operation execution latency. An alternative approach orchestrates a distributed operation as a chain of local actions. Each action is executed within an individual service. If an error arises at any stage, compensating operations are initiated. These revert previously executed modifications. In such instances, consistency is attained through managed system state recovery.

Event-driven transaction coordination extends this logic, further reducing participant coupling (Narváez et al., 2025). Here, local state alterations are accompanied by the publication of events, which initiate subsequent steps in other services. Such a model better accommodates workload escalation and system scaling. Nonetheless, it renders event processing order, idempotency, and the accurate commitment of intermediate states critically important (Mohammad, 2025). Consequently, each examined pattern provisions consistency disparately: via rigid coordination, compensation, or asynchronous state propagation. Table 2 presents the primary quantitative results evaluating these transactional approaches.

Table 2. Key Quantitative Results of Transactional Approaches Evaluation (Compiled by the author based on source: (De Heus et al., 2022))

Indicator	Value
Overhead of coordinator functions	about 10%
Performance advantage of Sagas over two-phase commit	15–34%
Scalability of Sagas with increasing number of workers	about 90%
Scalability of two-phase commit with 10% transfer operations	87%
Scalability of two-phase commit with 100% transfer operations	75%

The table's data indicates that discrepancies between the approaches manifest in the data consistency model and their behavior under escalating load. The model of sequential local operations with compensating actions demonstrates superior scalability with an increasing number of workers and enhanced performance relative to the two-phase commit. Concurrently, the two-phase commit affords more stringent data consistency guarantees, yet it handles an expansion in the share of transactional operations less effectively.

The impact of a distributed transaction pattern on system performance is determined predominantly by the degree of synchronization among operation participants. The more a transaction relies on centralized validation, the greater the aggregate latency, increasing the system's vulnerability to bottlenecks in individual services (Cortellessa et al., 2022). This implies that assessing a transactional mechanism should not be confined exclusively to the type of consistency guarantees it provides. It must incorporate its influence on latency, scalability, and the resilience of the distributed architecture within a genuine load environment. The corresponding quantitative distinctions between the approaches are depicted in Figure 1.

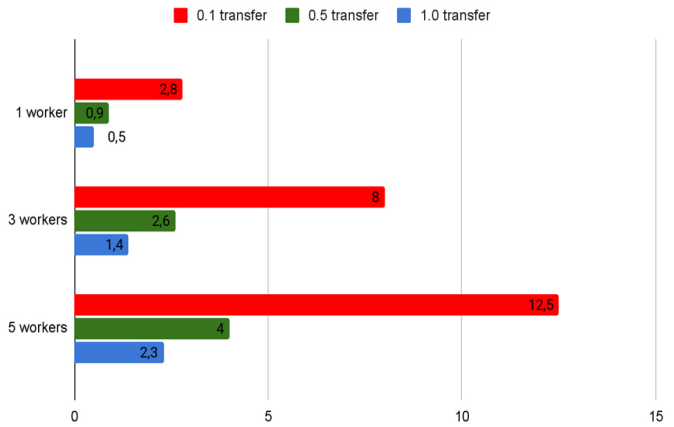

Figure 1. Maximum Transaction Throughput When Using the Model of Sequential Local Operations with Compensating Actions (Saga) (Compiled by the author based on source: (De Heus et al., 2022))

The results presented in Figure 1 reflect the variation in system throughput using sequential local operations with compensating actions as the number of workers and the proportion of data transfer operations increase. With a data transfer proportion of 0.1, the system delivers 2.8 thousand requests per second with one worker, 8.0 thousand with three, and 12.5 thousand with five workers. As the share of data transfer operations escalates to 0.5, throughput declines to 0.9, 2.6, and 4.0 thousand requests per second, respectively. In the most heavily loaded configuration, where the proportion of transactional operations reaches 1.0, the metrics drop to 0.5 thousand requests per second with one worker, 1.4 thousand with three, and 2.3 thousand with five workers.

The obtained values reveal a robust correlation between the number of workers and system performance. Across all regimes, augmenting the number of workers yields a notable surge in throughput. For instance, at a data transfer proportion of 0.1, transitioning from one to five workers amplifies performance by more than fourfold—from 2.8 to 12.5 thousand requests per second. Even under the most strenuous load configuration, scaling remains effective: the metric rises from 0.5 to 2.3 thousand requests per second. A contrasting scenario is observed when employing two-phase commit transactions (Figure 2).

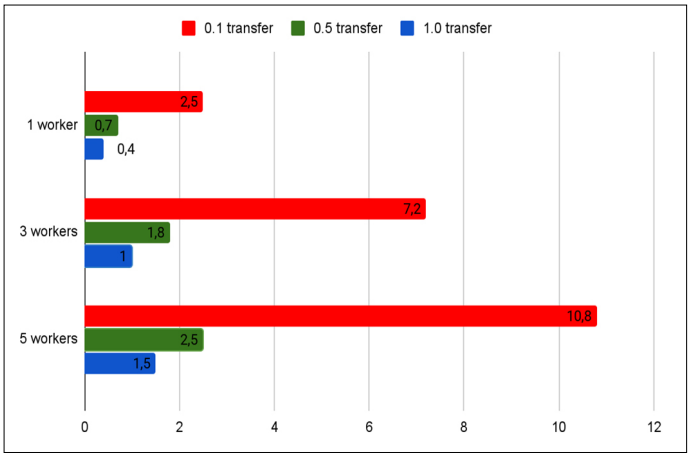

Figure 2. Maximum Transaction Throughput When Using Two-Phase Commit (Compiled by the author based on source: (De Heus et al., 2022))

At a data transfer operation proportion of 0.1, the system registers 2.5 thousand requests per second with one worker, 7.2 thousand with three, and 10.8 thousand with five workers. Upon increasing the share of transactional operations to 0.5, throughput decreases to 0.7, 1.8, and 2.5 thousand requests per second, respectively. Under a total transactional workload (a proportion of 1.0), the values decline to 0.4 thousand requests per second with one worker, 1.0 thousand with three, and 1.5 thousand with five workers.

Despite performance gains as the number of workers grows, the scaling effect here is markedly weaker. For example, at a data transfer operation proportion of 0.5, expanding the number of workers from three to five elevates throughput by a mere 0.7 thousand requests per second (from 1.8 to 2.5 thousand). This stems from the inherent characteristics of two-phase coordination, wherein every distributed operation necessitates consensus among transaction participants and confirmation from the coordinator. Consequently, as the workload intensifies, the overhead of synchronous interaction escalates, thereby curbing the overall performance augmentation of the system.

A comparison of data regarding storage models and transaction patterns reveals that in a microservice architecture, no universal mechanism is optimally suited for all scenarios. When confronted with rigid requirements for state invariants and a limited number of participants, the advantage shifts toward stricter coordination. Conversely, amidst high state contention, a multitude of concurrent operations, and intensive inter-service exchange, the advantage shifts toward Saga and event-driven coordination. Given a high risk of repeated message delivery, pattern selection can no longer be evaluated independently of idempotency, deduplication, and message routing controls (Mohammad, 2025).

Thereby, the architectural compromise in microservice systems is forged not between two abstract models, but rather among several specific system requirements. These encompass the repository type, consistency regime, load profile, acceptable locking costs, duplication risks, and operation traceability stipulations. If these parameters are assessed in isolation, the architectural choice remains incomplete (Waseem et al., 2022). If assessed holistically, the distributed transaction pattern begins to function as an integral component of a broader data integrity architecture.

Analyzing data storage architectures and distributed operation coordination mechanisms illustrates that data integrity in a service system is shaped not by a singular technical solution, but by a combination of several architectural mechanisms. The type of repository utilized establishes the baseline constraints of the data consistency model. The chosen transactional mechanism dictates the methodology for coordinating changes across services. Protection mechanisms against operation re-execution preserve event processing correctness amid repeated message deliveries. Observability tools permit the tracking of distributed change sequences and the detection of operation sequencing violations (Rossetto et al., 2024). The synergy of these elements forms the architectural foundation for ensuring data integrity in distributed service systems and defines the context for subsequent analysis of architectural solutions.

DISCUSSION

The results demonstrate that ensuring data integrity in a high-load service system cannot be reduced to the selection of a single distributed transaction pattern. The repository type, the operation coordination method, the nature of inter-service exchange, and the recovery regime following failures constitute a unified architectural framework. Therefore, the discourse should not revolve around the abstract question of which pattern is superior, but rather investigate under which operational conditions a particular mechanism preserves data correctness with the least architectural overhead.

Synchronous coordination of distributed operations retains an advantage in scenarios where a breach of state invariants is impermissible, even momentarily. In such a paradigm, state modifications are affirmed by all participants prior to operation completion, thereby maintaining strict data consistency. Yet, this precise attribute becomes a source of limitation under heavy loads. The greater the number of participants involved in a single operation, the more the system relies on the slowest node, subsequently amplifying the cost of locks and awaiting confirmations (De Heus et al., 2022). Hence, the synchronous model remains robust regarding guarantees but proves less resilient against escalating data contention and parallel request surges.

Asynchronous coordination operates on an alternative architectural logic. Here, an operation is not suspended pending global consensus from all participants; rather, it is decomposed into a chain of local state modifications followed by subsequent alignment. This approach substantially mitigates service coupling and facilitates superior scaling, which is particularly evident in high state-contention scenarios. However, the performance gains in flexibility are attained at the expense of more complex correctness oversight. While a synchronous schema strives to prevent divergence, an asynchronous schema mandates that the system be capable of detecting, localizing, and rectifying such divergence. Consequently, transitioning to an asynchronous model does not eradicate the data integrity problem; it shifts it from the domain of strict coordination to the realm of managed state recovery.

Event-driven architecture amplifies this shift. It permits services to update their states independently and link stages of a distributed operation through event publication, rather than through direct anticipation of a response from each participant (Suljkanović et al., 2022). As a result, the system more capably handles load escalation and service expansion, as the critical dependency between stages weakens. However, within such an architecture, issues of processing sequence, repeated delivery, and temporary state discrepancies manifest distinctly rapidly. An event may arrive repeatedly. It may be processed later than a preceding event. It might be delivered following a partial service failure. Therefore, an event-driven model is beneficial not autonomously, but exclusively in conjunction with mechanisms that preserve the correctness of the change sequence.

This dependency is particularly noticeable within error-handling mechanisms. Retries prove useful during transient failures, but if configured improperly, they rapidly devolve into a source of overload and secondary errors. Should the system begin indiscriminately reiterating requests to an already overwhelmed service, an availability issue swiftly transforms into a data integrity problem, as some operations are executed multiple times, others complete partially, and some stagnate in an intermediate state (Mohammad, 2025). For this reason, retry mechanisms, rate limiting, resource segregation, and failure isolation must be evaluated as elements of the data correctness architecture, not merely as components of fault tolerance.

In a high-load environment, data correctness becomes increasingly reliant on load management. As the intensity of inter-service exchange mounts, even a formally correct coordination pattern begins to underperform if the system lacks the capacity to restrict incoming traffic and balance message processing. An overloaded service acts as a latency accumulation point. Consequently, state retention durations lengthen, the probability of recurrent invocations rises, and the volume of conflicting operations expands. Therefore, the issue of data integrity within a high-load architecture must be deliberated alongside how exactly the system navigates load peaks and localized degradations.

Observability occupies a paramount position within this architecture. In a distributed system, an inconsistency rarely presents as a single glaring error. More frequently, it involves an extensive sequence of partial events: one service delayed a write, another executed a rerun, a third applied an obsolete state. Without end-to-end tracing, event correlation, and unified logging, it is impossible to reliably reconstruct the exact stage where the divergence originated and which precise step compromised the correctness of the distributed operation. This signifies that observability should not be regarded as a supplementary operational layer, but as an indispensable architectural mechanism for upholding data integrity.

The obtained results facilitate a transition from an isolated comparison of patterns to a model for selecting architectural data integrity mechanisms. This model is premised on the notion that a pattern cannot be selected absent the context of load, consistency prerequisites, data contention levels, and overall system complexity. An approach justifiable in one configuration may prove inefficient in another. Thus, selection should be guided not by universal applicability, but by the principle of architectural congruence with operating conditions.

The proposed model relies on four core parameters: load level, consistency requirements, degree of data contention, and system architectural complexity. Under low to moderate loads, a system can leverage stricter coordination mechanisms to preserve state invariants; however, as concurrent operations proliferate, the expense of synchronous confirmation escalates, rendering sequential local actions with compensating operations or event-driven coordination more effective. Architectural selection is also dictated by consistency demands: if even momentary state divergence is intolerable, rigorous change commitment mechanisms are mandatory, whereas if eventual consistency is acceptable, more flexible coordination schemes can be implemented. Data contention additionally plays a substantial role: amid high contention, the overhead of locks and synchronous waiting inflates more rapidly than the expenses associated with compensation and event alignment. Ultimately, system architectural complexity influences coordination mechanism applicability: in systems with few services and abbreviated interaction chains, strict transactional mechanisms remain manageable, whereas in distributed systems comprising numerous services and heterogeneous repositories, operation idempotency, defense against repeated message delivery, tracing, and sophisticated observability assume critical importance. The proposed architectural model for selecting data integrity mechanisms in microservice systems is illustrated in Figure 3.

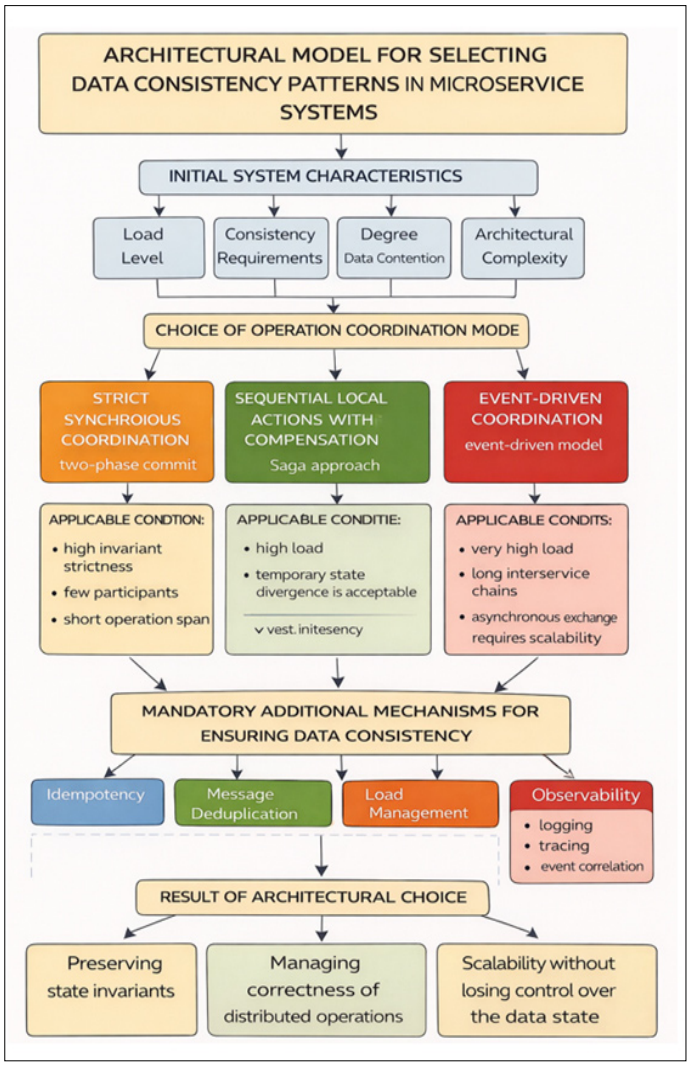

Figure 3. Architectural Model for Selecting Data Integrity Patterns in Microservice Systems (Author's Development)

Conceptually, this model can be described as follows. Under low load, a restricted number of participants, and stringent state invariant requirements, preference should be allocated to synchronous coordination featuring more rigorous change commitment. With escalating load and a surge in concurrent operations, the advantage shifts toward the model of sequential local actions paired with compensating operations. Amid high architectural complexity, extended inter-service chains, and a predominance of asynchronous exchange, event-driven coordination emerges as the most robust, provided it is strictly accompanied by idempotency, deduplication, processing order oversight, and advanced observability. Otherwise, the scalability benefits rapidly morph into an increased risk of hidden inconsistency.

The practical significance of this model lies in shifting pattern selection from the sphere of general preferences to the domain of engineering criteria. Should the system process financially sensitive operations and remain critical to invariant violations, the architecture must tolerate a higher coordination cost for the sake of strict correctness. If the system operates under an intensive event stream and requires horizontal scalability, priority is assigned not to maximum commitment rigidity, but to the capacity to maintain correctness via compensation, duplication defenses, and end-to-end tracing. Conversely, if the system integrates heterogeneous repositories, varying error costs, and differing service criticalities, pattern selection should not be a one-time, blanket decision for the entire architecture; rather, it should be enacted on a per-operation basis, contingent upon the specific business invariant requiring preservation.

This yields a practical recommendation for high-load microservice systems. Rigid synchronous coordination should be restricted to the shortest and most critical state modification boundaries. The model of sequential local actions with compensation should be applied where temporal divergence is permissible, but loss of operation control is not. Event-driven coordination should be utilized within the most scalable architectural boundaries, where inter-service exchange—rather than a local transaction—generates the primary workload. Across all three scenarios, protection against operation re-execution, precise retry configurations, load management, and comprehensive execution observability remain mandatory. This exact synergy constitutes the practical architecture for ensuring data integrity within a high-load distributed system.

CONCLUSION

The conducted research demonstrates that the issue of data integrity in microservice systems is fundamentally distinct in nature compared to monolithic architectures. Within a distributed service environment, data consistency ceases to be a property of a single database or a single transaction. It is forged as the outcome of interactions among multiple services, repositories, and inter-service coordination mechanisms. This implies that architectural solutions aimed at maintaining data correctness must be evaluated at the aggregate system level, rather than at the level of individual components.

The obtained results confirm that ensuring data integrity in a microservice architecture cannot be distilled into the application of one universal mechanism. System state correctness is achieved through the concurrent utilization of several architectural tools. The data storage model establishes the baseline constraints of the consistency regime, the transactional mechanism determines the approach to coordinating changes across services, while mechanisms defending against operation reprocessing and providing distributed execution observability preserve operation correctness amidst asynchronous interaction and high loads. Only the convergence of these mechanisms constructs a resilient data integrity architecture.

The study further revealed that the efficacy of architectural patterns is directly contingent upon the characteristics of the distributed system. Architectural solutions delivering high consistency rigor remain justifiable in systems with a limited number of participants and rigid state invariant requirements. As the load escalates, the volume of concurrent operations expands, and inter-service chains grow more intricate, models of sequential local actions with compensation and event-driven operation coordination prove more effective. Consequently, pattern selection must be guided not by universal recommendations, but by alignment with the specific operational conditions of the system.

The practical significance of the research lies in the formulation of an architectural model for selecting data integrity mechanisms for microservice platforms. The proposed architectural model demonstrates that selecting a data integrity mechanism in microservice systems must be rooted in a comprehensive evaluation of several parameters: load levels, data consistency prerequisites, the degree of contention for shared states, and the architectural complexity of service interactions. Holistically accommodating these factors enables the identification of the most efficient distributed operation coordination mechanism for a specific system configuration.

Future research may be directed toward refining the proposed model and validating it empirically. A promising avenue involves analyzing architectural solutions within genuine high-load systems where heterogeneous data repositories, disparate consistency regimes, and varying tiers of business-invariant criticality interact. Furthermore, quantifying the impact of observability mechanisms, load management, and operation retries on distributed transaction resilience holds considerable interest. Advancing these trajectories will deepen the understanding of architectural data consistency mechanisms and elevate the reliability of contemporary microservice platforms.

REFERENCES

1. Cortellessa, V., Di Pompeo, D., Eramo, R., & Tucci, M. (2022). A model-driven approach for continuous performance engineering in microservice-based systems. Journal of Systems and Software, 183, 111084. https://doi.org/10.1016/j.jss.2021.111084
2. De Heus, M., Psarakis, K., Fragkoulis, M., & Katsifodimos, A. (2022). Transactions across serverless functions leveraging stateful dataflows. Information Systems, 108, 102015. https://doi.org/10.1016/j.is.2022.102015
3. Halili, F., Nuhiji, A., & Mustafai Veliu, D. (2025). Polyglot persistence in microservices: Managing data diversity in distributed systems (arXiv:2509.08014). arXiv. https://doi.org/10.48550/arXiv.2509.08014
4. Indykov, V., Strüber, D., & Wohlrab, R. (2025). Architectural tactics to achieve quality attributes of machine-learning-enabled systems: A systematic literature review. Journal of Systems and Software, 223, 112373. https://doi.org/10.1016/j.jss.2025.112373
5. Mohammad, M. (2025). Resilient microservices: A systematic review of recovery patterns, strategies, and evaluation frameworks (arXiv:2512.16959). arXiv. https://doi.org/10.48550/arXiv.2512.16959
6. Morabito, G., Ficara, A., Celesti, A., Villari, M., & Fazio, M. (2026). Consensus-based distributed orchestration framework for microservices in edge computing clusters. Future Generation Computer Systems, 176, 108221. https://doi.org/10.1016/j.future.2025.108221
7. Narváez, D., Battaglia, N., Fernández, A., & Rossi, G. (2025). Designing microservices using AI: A systematic literature review. Software, 4(1), 6. https://doi.org/10.3390/software4010006
8. Rossetto, A. G. d. M., Noetzold, D., Silva, L. A., & Leithardt, V. R. Q. (2024). Enhancing monitoring performance: A microservices approach to monitoring with spyware techniques and prediction models. Sensors, 24(13), 4212. https://doi.org/10.3390/s24134212
9. Singjai, A., & Zdun, U. (2022). Conformance assessment of architectural design decisions on API endpoint designs derived from domain models. Journal of Systems and Software, 193, 111433. https://doi.org/10.1016/j.jss.2022.111433
10. Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Feature-driven characterization of microservice architectures: A survey of the state of the practice. Applied Sciences, 12(9), 4424. https://doi.org/10.3390/app12094424
11. Suljkanović, A., Milosavljević, B., Inđić, V., & Dejanović, I. (2022). Developing microservice-based applications using the Silvera domain-specific language. Applied Sciences, 12(13), 6679. https://doi.org/10.3390/app12136679
12. Waseem, M., Liang, P., Ahmad, A., Shahin, M., Ali Khan, A., & Márquez, G. (2022). Decision models for selecting patterns and strategies in microservices systems and their evaluation by practitioners. Proceedings of the 44th International Conference on Software Engineering (ICSE): Software Engineering in Practice (SEIP). https://doi.org/10.48550/arXiv.2201.05825
13. Zouani, Y., & Lachgar, M. (2024). Zynerator: Bridging model-driven architecture and microservices for enhanced software development. Electronics, 13(12), 2237. https://doi.org/10.3390/electronics13122237