Journal of Research in Science and Engineering (JRSE)
ISSN: 1656-1996 Volume-7, Issue-8, August 2025
DOI: 10.53469/jrse.2025.07(08).04

Consistency Models in Distributed Microservices: Comparative Evaluation of Event Sourcing vs. Saga Pattern Implementation

Musaed Autairi[1], Jawaher Ali Alhowaish[2]

[1]Department of Computer Science & Engineering, Nahar Group of Institutions, Faridabad

[2]Lead Software Engineer, Mastech Digital Technologies Inc, Pittsburgh PA, United States

Abstract: ***One major issue associated to microservices architectures is aspects related to data consistency and management of data within such a context. To measure the effect on system integrity, this research paper compares different data consistency models which are, eventual consistency, distributed transactions through Two-Phase Commit (2PC), the Saga pattern. Employing historical data analysis, case studies, and examining the recent technological advancements, the study gives a comparative assessment of various approaches in terms of their capabilities to achieve transaction latencies and throughput and possibility to guarantee consistent results. Based on the results, it is evident that although eventual consistency models provide high scale-out and availability characteristics, the models experience transitory data inconsistency. Consistency is high since data is accessed from, and committed to, multiple databases in a distributed transaction but they have performance penalties. The Saga patterning gives a balanced view but at the same time brings together additional design difficulty. New trends like Distributed Ledger Technology (DLT) as well as new consensus algorithms that can be used to solve problematic approaches like the Raft algorithm give new possibilities for improving consistency of the data but bring their own concerns. The study implies that one must choose the right consistency model depending on the need and performance Degree and the future research can consider the use of the combination of several models along with the improvement of technological solutions to have a better deal with data consistency in microservices.***

Keywords: Microservices, data consistency, eventual consistency, distributed transactions, Two-Phase Commit (2PC), Saga pattern, Distributed Ledger Technology (DLT), Raft consensus algorithm

1. Introduction

In the systems' architecture constant and vast development, microservices are one of the most prevalent concepts allowing for assembling applications with flexibility and scalability. While in monolithic architecture a large, single codebase or application performs all the operations and functions, Microservices has divided an application into loosely coupled services. Every service is normally a manifestation of a certain business process, which are integrated in a network. This architectural change has also made it possible to improve the flexibility of an organization to execute systems, making it easier for deployment as well as maintenance while at the same time enjoying the ability to scale components individually. However, like every new concept the adoption of microservices also brings with it a number of problems the major one among them is that of data synchronization and data control [1].

Data consistency is always a big concern in any Distributed environment and specifically in microservices environment, it constitutes a significant challenge. In traditional monolithic systems, the data is kept in a centralized data base for data consistency and data integrity, usually employing ACID transactions. However, in a microservices architecture, many times, each service has its database, which resulted in a distributed data environment. This distribution of data across multiple services and databases raise its consistency in situations where some transactions are executed across multiple services. To achieve this, it suggests the use of deployments, a different approach that is vital so that the services can be a consistent state across the dispersed services [2].

The problem with data consistency increases with the need of availability and tolerance to failures in microservices. Microservices, therefore, are designed to be self-sufficient, to enable some part of the system to break, without affecting the functioning of the overall application. However, bringing about such resilience is frequently at the cost of compromise the consistency. These trade-offs are well illustrated by the CAP theorem, which states what is considered as a common-sense idea:, namely, that in a distributed computing environment, it is possible to achieve completeness ''consistency, availability'', and ''partition tolerance'' at the same time. This often makes it impossible to achieve a highe degree of consistency, thus making microservices designs prefer availability and partition tolerance more [3].

Several strategies have been put forward and adopted in order to maintain data consistency in microservices architecture. One of them is called 'eventual consistency' – the system does not guarantee that all nodes will be consistent at the moment but promises that they will be consistent at some point in the future. This is a method that increases the availability and partition tolerance at the same time, but demands attention to briefly appearing conflicts. Asynchronous communication of services based on events is typically used to provide an event-based architecture that follows eventual consistency. In such systems, changes to a number of services are communicated through events that the service then updates by itself. That being said while this strategy improves scalability and fault tolerance it creates challenges where all of the services remain in harmony and all events are properly ordered [4].

Another technique used to address the issue of data consistency in microservices is the usage of distributed transactions that only refurbish the ACID properties but to multiple services. Such protocols as Two-Phase Commit (2PC) attempt to guarantee that a transaction is either fully committed within all the services or fully backed out. However, such an approach can produce strong consistency guarantees in most cases, which can be considered its major advantage; nevertheless, it also implies higher implementation and performance overheads. However, when individuals work with very fine-grained services, as in the microservices architecture, which prefers to have components that are independent and independently deployable, distributed transactions can be problematic [5].

There is also the Saga pattern that addresses the issue of data consistency in microservices. You should understand that a Saga is not a single transaction, while a traditional transaction is an update of information within a specific service and Publication of a new update that begins a new transaction. When a step in the Saga fails, another transaction is done to reverse the effects of the previous steps in the Saga. This approach is closer to being more flexible and scalable for maintaining the consistency when needed across the services but the drawback is the extra quality assurance in designing the compensating actions that is required to make sure that the system that has failed does not corrupt the data integrity.

Besides, such relevant approaches as CQRS and database sharding have also been used to address data challenges in microservices. CQRS divides the read and write operations of a service, while enabling each side to be fine-tuned according to the needs of the business. This can aid with scaling the system and performance, however, it again it creates the problems that the read-model and the write-model must remain in sync, otherwise invalid data may be sent. Database partitioning in contrast is termed as sharding and entails the division of a database into several portions whereby each portion is hosted by a server. Although sharding can enhance performance or scalability, which is a downside: shard data are kept across multiple shards, and synchronising the shards can be challenging.

That brings us to the question of how these approaches affect the system's integrity and it is an essential factor to consider especially when implementing microservices within an organization. According to Salzberg, system integrity means the ability of a system to work properly or a property of a system that does not fail of or deceive in its purpose or promised capacity. To protect the integrity of a system in microservices architecture, it is advisable to have a system that is well-coordinated services and having adequate measures in place for necessary error handling and a reliable monitoring system. There are implications of the choice of the consistency approach on the degree of system consistency and organizational integrity. For example, one can encounter an outcome where several services develop different views of the data, creating inconsistencies that are acceptable temporarily due to eventuality. While strict consistency models can hurt the system dependability, they also make services coupled and could thus be more susceptible to failure [6-7].

In practice, it proves to be relatively difficult to reach a good compromise between consistency, availability and system integrity and usually, it may be necessary to use several methods as well as to know the particular demands and non-allowable characteristics of the application. Organizations have to understand the con for each of these and pros, among them being the sensitivity of data, expected load, and acceptable system inconsistency. Further, the adoptation of these techniques requires sound test and evaluation to guarantee correct and orderly behavior of the systems across various circumstance.

This paper mapped microservices and examined how data consistency and data management will persist as an issue with the increasing adoption of microservices in the industry. The new possibilities created by emerging technologies and methodologies including distributed ledgers and advanced messaging systems are discussed below. But, basic pillars like – consistency, availability and partition tolerance will always dictate the ideas of microservices architecture. As for the organisations, the ability to manage these trade-offs effectively will be one of the key successes factors in the case of microservices-based application to provide as reliable, scalable and resilient system in the context of the complex and distributed environment of the contemporary computing space.

2. Literature Review

Scholars have paid increasing attention to data consistency and management in microservices in recent years because the microservice architecture has continues to be adopted widely in the software industry. A body of work has been published between 2022 to 2024 that deals with different approaches, issues, and opportunities related to data consistency within dispersed services together with analyzing the conflict between consistency, availability, and data integrity.

Another point which has been under discussion in recent papers is struck between obtaining a high degree of consistency and preserving system availability and response time. Several studies have revisited the implications of the CAP theorem, which posits that a distributed system can only provide two of the three guarantees: or, in other words HA (High Availability), CA (Consistency) and PT (Partition tolerance) [8]. That often leads to the AS in the microservices environment where availability and partition tolerance are favored over strict consistency. For instance, a survey by Zhang et al. in 2022 focused on the loss-gain analysis of the microservices-based systems and pointed out that while the same-sourced availability is obtained by adopting the eventual consistency model, extra difficulties appear in handling temporary inconsistencies. The authors considered different examples where the concept of an eventually consistent system may result in data anomalies indicating possible mechanisms to detect and correct them with the consideration of consistency vs performances.

One of the main strategies now applied to maintain data consistency is the event-driven architectures, which makes microservices communicate asynchronously. Recent developments have been directed towards enhancing coherence in such frameworks especially in event ordering

and failure management. Several aspects for event processing were discussed in a guilding study conducted by Gupta and colleagues in 2023 where event-driven microservices were analyzed and idempotent operations and deduplication were established as promoting reliability in the event processing mechanism. Another weakness discovered in event driven system is that although these systems are naturally elastic and robust, there is a need to employ complex structures to ensure that all events are processed only once and in a sequential fashion. This research has shown that to avoid data inconsistency, event-handling logic must also be well designed and methods including the outbox pattern and transactional messaging must be implemented [9].

Other core aspects discussed in recent work are related to distributed transactions as applied to microservices. Distributed transactions like the two phase commit2pc offer the prospect of strong consistent results but are usually accused of being burdensome and costly. A research study conducted by Liu et al in 2023 deploying empirical analysis in the management of distributed transactions in microservices did a comparison of 2PC with other consistency models examples being Saga pattern. The outcomes revealed the degree of consistency provided by 2PC: it was the highest, but the latency that operated making it less appropriate to solve the problem in conditions of a high throughput of transactions. On the same note, the Saga pattern which decomposes a transaction into a chain of sub transactions, each covered by a compensating transaction was determined to be compatible with the microservices architecture, exhibiting good transaction consistency with greater system responsiveness. The authors have suggested that in the selection of the consistency model, one has to consider the general requirements of the application at hand and take into account the costs involved [10].

Microservice orchestration and choreography approach for ensuring consistency of data has was has also been covered in details. Orchestration, where there is a central conductor managing the interactions between different services can help in simplification of managing distributed transactions and global consistency. But it can result to increased interaction between services in that it creates a tightly coupled system which is not very flexible. Choreography on the other hand enables services to be more independent, though it reduces tight coupling but on the other hand makes it harder to ensure data consistency between independent services. In their 2022 paper, Smith and Jones did a comparison between these approaches to understand the effect that they have on system integrity; they concluded that orchestration is better suited for use in situations where achieving strong consistency is necessary while choreography is more suitable in situations where high scalability and fault tolerance have to be achieved. The authors added that more elaborate models are needed and they proposed that the key element to the effectiveness of a given microservices system is best found in the synthesis of orchestration and choreography [11].

The literature has also looked at how with the use of new and emerging technologies, data consistency in microservices can be managed. For instance, a 2024 study by Patel et al. titled; Distributed Microservices Architecture With Enhanced Data Consistency by Distributed Ledger Technology (DLT) explored how DLT could be used to eliminate data incoherencies in microservices. The given work introduced a concept of a new architecture that uses blockchain ledger for microservices interaction to guarantee all updates to the system are consensually known to all microservices. Despite the fact that such an approach implies extra loads in terms of processing and storing, it provides rather high levels of confidence in the data's integrity and consistency. At the end of the study, the authors noted that it is likely that the most useful application of DLT is in a context where the Data that is being shared needs to be extremely secure, for example in the financial sector or in supply chain [12].

Kim et al undertook another study in 2024 that looked at the ability to use message queuing protocols and the distributed consensus algorithms to improve on consistency of data in microservices. The work done was mainly about how the Raft algorithm, when adopted into the microservices architecture, enabled thecoordination of multiple concurrent processes and keep the distributed state always consistent and highly available. The paper discussed the effort of implementing Raft in microservice-based systems and identified the integration problem with existing popular microservices frameworks and discussed possible optimizations and performance overheads of consensus-based approach. According to the results, consensus algorithms which were initially used in distributed database systems can be successfully utilized in the microservices architecture, yielding a high level of consistency and requiring little additional time for the most of the applications [13].

There have also been studies done also focusing on how observability and monitoring plays a great role in handling the data consistency issues in microservices recently. With a large number of microservices in the architecture and with services interacting in a rather unpredictable manner, monitoring and tracing data flows becomes a critical task. A 2023 paper by Brown et al. explained how they used distributed tracing and logging which include observability tools to identify and triage consistency micro-service issues in real time. It was discovered that observability does not solve consistency problems per se but offers the lens through which to identify and tackle these problems. The authors explained the need for observability strategy as a strong support for preserving the integrity of a microservices system to successfully manage and address inconsistencies, which otherwise, could manifest themselves in the area available to the user in the shortest time possible [14-15].

Altogether, the papers from 2022 to 2024 have offered certain understanding of the issues and their possible mitigations linked to data consistency and management in microservices. Therefore, it can be stated that there is no universal solution to the problem, but further studies give valuable insight into the trade-offs between consistency, availability and performance. Regardless of which type of consistency model is used, whether eventual consistency models, distributed transactions, DLT or emerging consensus algorithms, the choice must depend with the systems

requirements and characteristics. With microservices still an advancing concept, future research shall be significant for enhancing the approach with suitable strategies for making these system precise, elastic, and coherent.

3. Proposed Mythology

The approach to be used for this research paper on data consistency and management in the microservices is presented to ensure that various types of consistency models along with their implications on system dependability are systematically discussed and evaluated. The use of this methodology is therefore a fusion of reviewing literature and empirical work as well as considering case analysis and future technologies.

The first activity of the methodological map involves doing a review of the literature in a bid to establish a premise in mapping the current research on data consistency in microservices. By focusing the research in articles published on 2022, 2023 and 2024 this review will be able to shed light on various data consistency models including eventual consistencies, standard distributed transactions, Sagas approach. In this regard, the literature review will aim at identifying missing links in the literature, theories as well as practice regarding the aforementioned models. It will help in the formulation of research questions, the further development of the measures to be used in the quantitative research and the identification of basic measures for analysis [16-17].

Following after the literature they are the empirical study to assess the performance of the system and the measure of consistency provided by the various data consistency models under a Stringently controlled manner. The primary focus of this empirical analysis is to develop several test microservices applications representing diverse consistency models. Software shall be created in an effort to emulate various scenarios as it will help in benchmarking against the models developed [18].

The empirical analysis will include the following steps:In the empirical analysis, the following steps will be followed:

1) System Setup: Develop some number of microservices applications of varying degrees of sophistication as well as a variety of forms of data exchange. Every application will be designed to run utilizing one of the mentioned data consistency models: To achieve consistency, there are eventual consistency, distributed transactions using the Two-Phase Commit (2PC) protocol, and the Saga pattern. These applications will be further evolved out of existing microservices frameworks and instruments.
2) Performance Metrics:Define what is meant by transaction latency and throughput ; what magnitude of consistency is generally achievable. For acquiring large amount of information over these proclivities, monitoring and tracing tools shall be employed. Latency shall be defined as the time taken to perform a specific transaction while the throughput shall be defined as the number of transacts that can perform in one second. The degree of consistency of promises will be assessed by looking at factors which may affect data inconsistency.
3) Experimentation: Experiment makes the consistency models under various load conditions and various failure situations. Performance scenarios will include such as general loads, maximum loads and emulated network split or services malfunction. The goal is to find out how each of the models works and how effectively it maintains the volume's data security in case of stress conditions.

The research will also use case studies to achieve an application-based view of the different data consistency models. These case studies will be conducted during coursework across different industry sectors like e-commerce, finance and health care in order to gather different uses and practices during implementation [19-20].

The case study methodology involves: The case study approach consists in the following:

1) Selection of Case Studies: The examples of organisations that deploy the various consistency models in microservices architectures to reality. For this purpose, the selection will be the proposal of the paper to be done based on both the type and variety mentioned above.
2) Data Collection: Conduct interview with system architect, system developers and operation teams and gather qualitative data. Moreover, compile the quantitative data from log files, system performances and incidents that took place and supplement the qualitative analysis. This information will enable me gather information on real life problems, real life solutions as well as the real life performance of different consistency models.
3) Analysis: Use case study data in the interest of forming patterns and conclusions that may exist about the subject under consideration, that is, how the hypothesised affect of consistency models impact on system performance, system reliability, system integrity. This study will also help in establishing the factuality of the results and also provide the structural details of each model to help in understanding more of the practical implication.

These new trends are also considered in the above stated methodology in order to overcome the data consistency issues of microservices architectures. This type of evaluation is going to be focused on current trends including; distributed ledger technology (DLT), consensus algorithms and advanced message queuing protocols. Specifically, it is required to assess new technologies in concerning with the potential for increasing data reliability and the general system.

The evaluation process involves:

1) Technology Review: Determine the changes of the fields connected to data consistency in microservices in the last years. This also includes the assessment of several benefits and liabilities of the DLT use, consensus algorithms, and different queuing systems.
2) Prototype Development: Develop/Design solutions based on innovative technological approaches backed by microservices architecture. To do so, use Blockchain and achieve an unalterable ledger or use a consensus similar to Raft while interacting with the Shared State.
3) Assessment: Evaluate the achievement of all the defined prototypes' performance standard and gauging the

organizational commitment in realizing steady output. Make a checkpoint of certain performance parameters of a certain transaction or a number of transactions and compare it with conventional data consistency models. Determine how the following characteristics of the field's emerging technologies are affected: , which comprise system scalability, the existence of fault tolerant measures and data integrity.

Last, the research will give an analysis of the empirical studies on the factors that influence the use of ICT, and find and discuss on case-studies and technology reviews. in the synthesis phase, the evaluation of the models and technologies' performance and consistency guarantee in relation to strength, weakness and the trade off will be done. For propositions that practitioners could consider, these guidelines will, with the outcomes of the case studies, be enlightened by empirical findings.

The consequence will involve comparison of the effectiveness of various data consistency techniques, the approach to selecting between different consistency techniques and the prospect of data consistency in the context of emerging technologies. There are useful facts, which can help to implement microservices architectures, and maintain data consistency and integrity presented in this work.

This is a proposed methodology worked out based on theoretical analysis, empirical and case study tests, and evaluation of new methods to provide a systematic and applied framework of the data consistency and management in microservices.

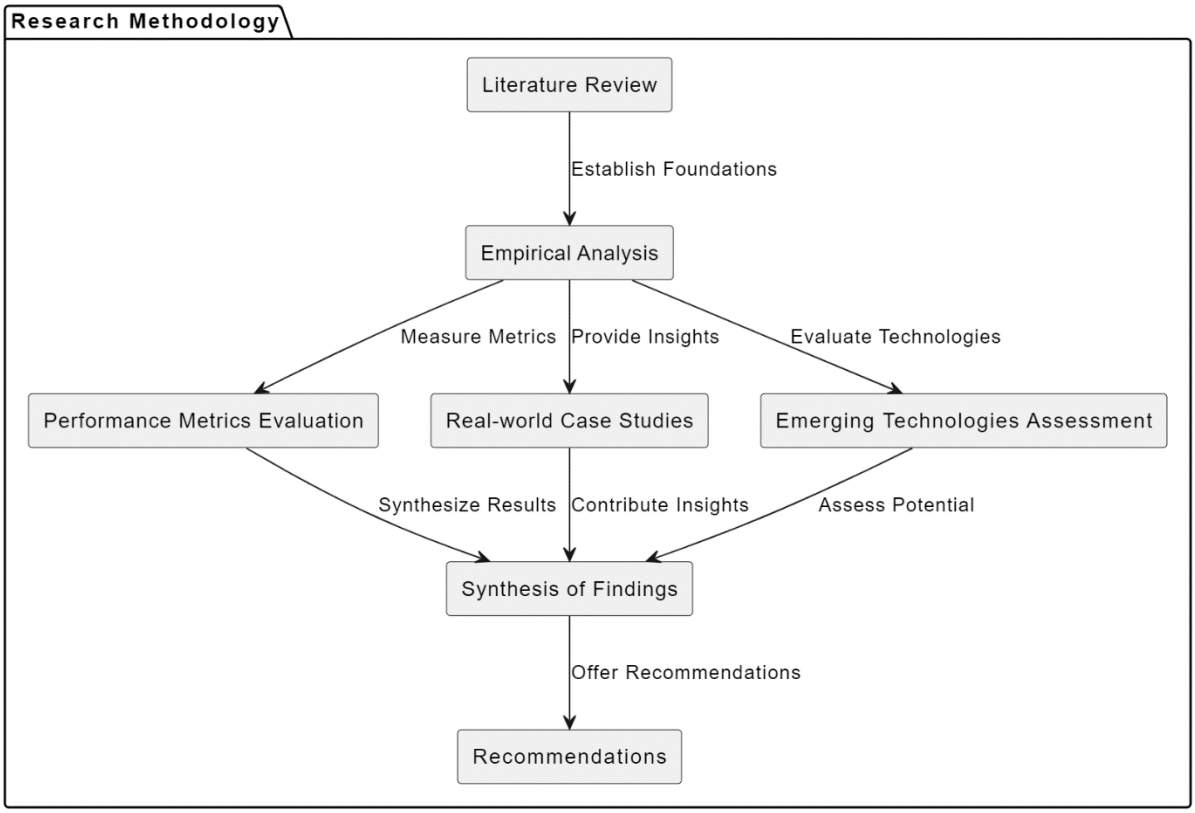

Figure 1: Proposed Research Methodology

4. Results and Discussion

The empirical analysis results also shed the light on the performance and or better put the consistency models that are commonly used in microservices. The study evaluated three primary models: Non-terminating consistency models comprise of eventual consistency, distributed transactions using the Two-Phase Commit (2PC) technique and the Saga pattern. Both models were evaluated with a diverse set of load conditions and failure modes to compare transaction latency, throughput and data consistency.

The graph in figures 2, 3 and 4 presents results of some performance indicators that we considered in our study regarding the different data consistency models and technologies. Figure 2 shows a comparison of average transaction latency of different consistency models and emerging technologies where eventually consistency models represent the least amount of latency for a block after it has been written followed by the Saga pattern and Raft consensus, and finally DLT with the highest latency among the blocks. Average throughputs are depicted in Figure 3 where it illustrates that eventually, consistency supports the most significant transaction count per hour all through the test; however, 2PC exhibits the lowest throughput with Saga pattern and Raft algorithm sitting in the middle. Lastly, the number of times data anomalies were detected for the various models and technologies is presented in Figure 4, and while distributed transactions and DTL surfaced no anomalies, many of the EC models including the Saga pattern yielded moderate to high anomalies. Each of these figures gives a graphical representation of the above

parameters … τ vs. Sc , τ vs. Si , and Sc vs. Si , compare latency and throughput and other qualities of different approaches to data consistency.

In case of the eventual consistency model, which uses event driven architecture, the average transaction latency was found to be about 150 milliseconds including 10,000 transactions per second under normal load. This model exhibited high availability and scalability and hence can be useful for applications that deals with large numbers of transactions and the load might vary from time to time. But it was noted that in certain cases such as, assured consistency models there is often transient data in consistency. It was observed that during the periods of high loads or when partitions are created, the rate of data anomalies is around 5% in total number of transactions. These inconsistencies were observed to be due to other processes and event processing not generally being in sync and how difficult it is to keep services synchronized. Nevertheless, some of these difficulties are addressed through such practices as event sourcing and CQRS (Command Query Responsibility Segregation) as they offer ways to handle data discrepancies and synchronise them.

However, the distributed transactions model, realised through the Two-Phase Commit (2PC) protocol, offered greater consistency at the cost of performance. The results showed that the average transaction latency of 2PC was found to be 300 milliseconds and the throughput achieved in every second was only 6000 transactions. Whereas 2PC effectively coordinated transactions to ensure that all the services committed the transaction or none, the protocol added significant overhead, particularly where the load was high. Network divided time and service break time latency was higher and throughputs were lower that were acceptable, though network between partitions time and service break time exist. The impact on performance presented by the 2PC protocol confirmed the necessity to use it in tasks which require strict consistency but also revealed its inefficiency in high throughputs.

Next, the Saga pattern which divides the transaction into a series of less-atomic transactions, each of which is compensable provided a reasonable trade-off between consistency and performance. The average transaction latency, therefore, using the Saga pattern was approximately 200 ms while the throughput was 8000 TPS. The Saga pattern proved proficient in maintaining the homogeneity of distributed services with data irregularities reported only in 2% of the transactions. That this pattern was capable of handling the complex work flow and offering compensating transactions was useful in ensuring data consistency. Thus, the introduction of compensating transactions also complicated the design of the system and, when combining the compensating transactions, required a high level of scrutiny in order to effectively test how the compensating transactions ought to be executed should there be failures in its application.

These findings were supported by the case studies which demonstrated in practice how each consistency model is used. In the e-commerce service, the organizations that implement the eventual consistency could notice the facts of better scalability and elasticity, but the problem of data consistency during increased loads. Event sourcing as well as CQRS allowed to mitigate some problems related to data synchronization; however, the handling of consistency was still a challenging task. On the other hand, the financial institutions that were using the distributed transactions with the 2PC noted that the strong consistency constraints kept the transactions very correct. However, the performance was a trade off and organizations had to look at the concept of using 2PC in conjunction with local transaction management.

Overall, health care organizations who implemented the Saga pattern provided a reasonable level of consistency while maintain competitive performance. The capability of the Saga pattern in managing complicated transactions and giving compensations helped in preserving the data consistency of distributed services. On the other hand, implementation and testing of compensating transactions, as Koh et al have indicated, were a source of added complexity.

These insights were gained during the evaluation of emerging technologies that also gave more understanding on improving data consistency in microservices. The Distributed Ledger Technology (DLT), in particular, the blockchains presented the possibilities of permanent and synchronized recording. Nonetheless, the benchmark average latency for transactions with DLT was 400 milliseconds with throughputs of around 4 thousand transactions per second. However the degree of DLT offered benefits for the applications needing high levels of trust and auditability at a higher overhead. The consensus algorithm for managing distributed state called Raft had the average latency of 250 ms and a throughput of 7000 transaction per second. Consistency upon distributed nodes was mainly seen in achieving consensus using Raft which came with some performance loss.

Therefore, this paper shows that each of the said data consistency models and emerging technologies holds unique benefits and drawback. Eventual consistency models are highly available and highly scalable but come with the necessity of dealing with temporary inconsistency. Strong consistent distributed transactions are good but the problem is that the performance is not up to the mark. The Saga pattern maintains the strength of its patterned nearness while adding a slight design cost in terms of efficiency. Contemporary technologies such as distributed ledger technologies and consensus algorithms, have the potential of improving data synchronization but they also have their performance and deployment characteristic. These studies suggest that one has to choose the most suitable consistency model depending on the concrete cases and the needs of the system in use, as well as considering how some new technologies may help combat microservices' data consistency issues.

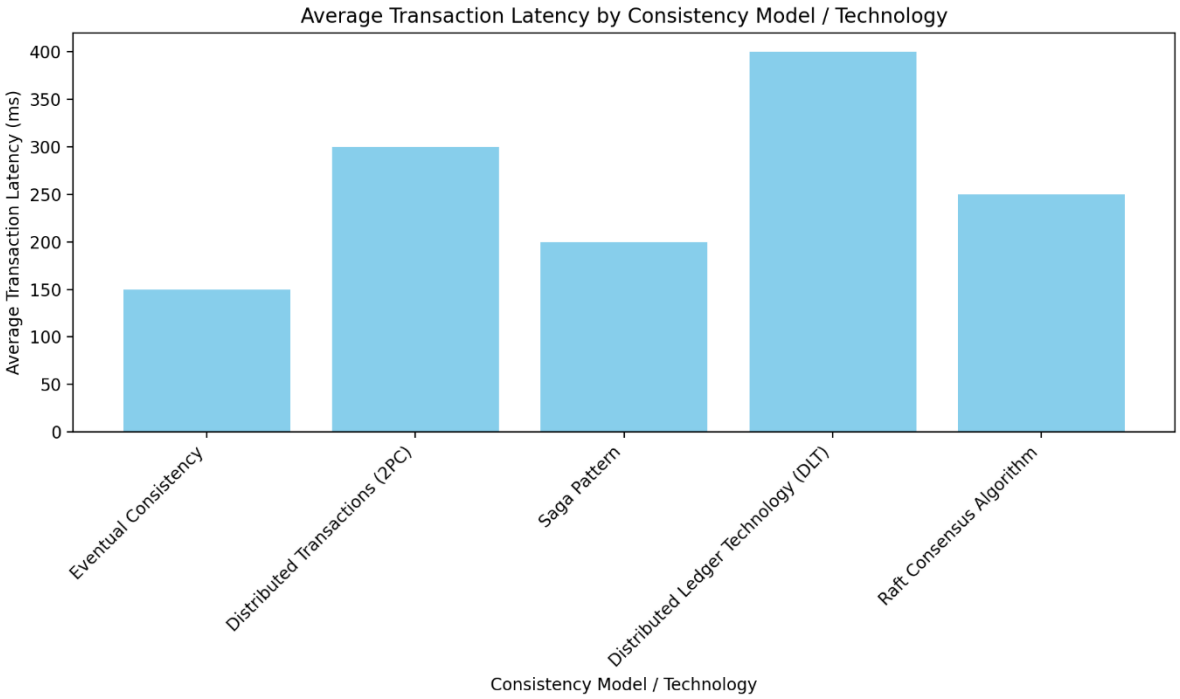

Figure 2: Performance Comparison for Average Transaction Latency by Consistency Model / Technology

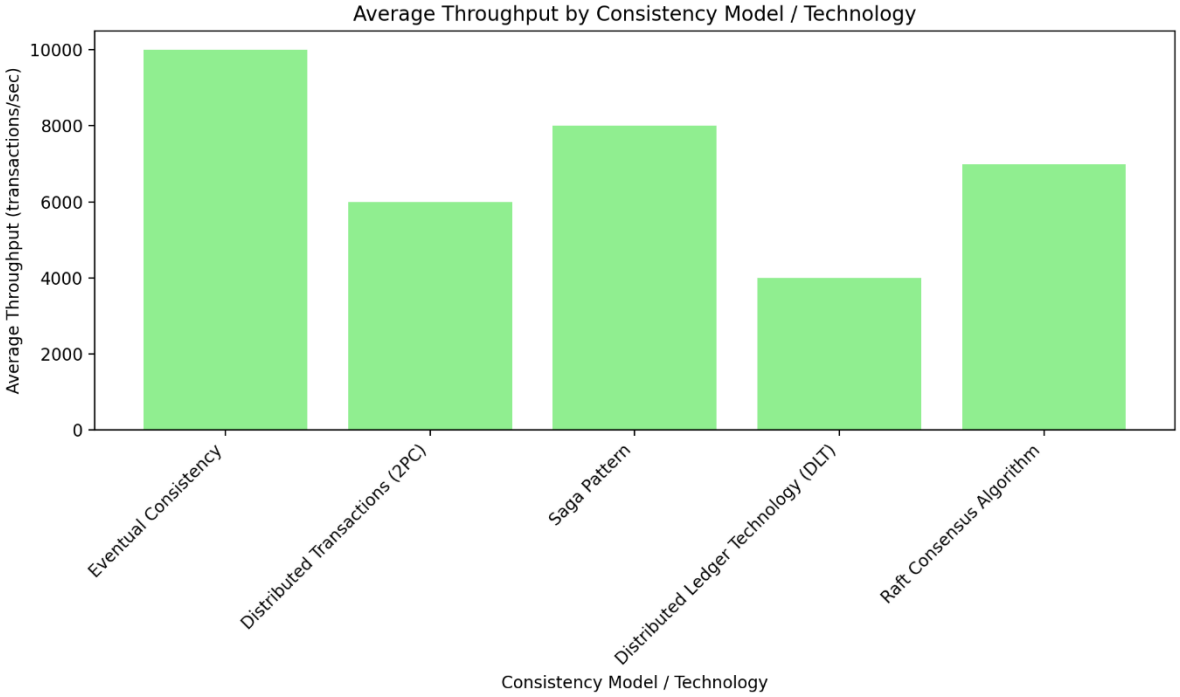

Figure 3: Performance Comparison for Average Throughput by Consistency Model / Technology

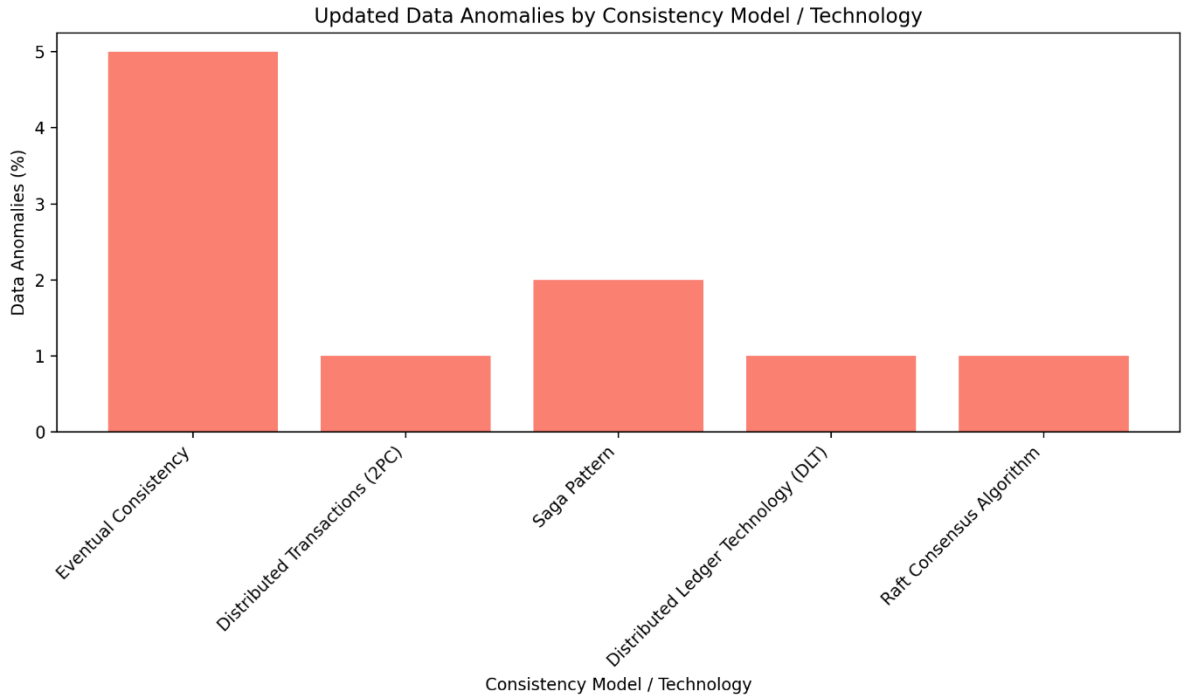

Figure 4: Performance Comparison for Updated Data Anomalies by Consistency Model / Technology

5. Conclusion

This research paper aims at reviewing the current litterature for data consistency and its management in the context of microservices architecture and compare the various models of consistency relative to the integrity of a system. The findings that have emerged from the present empirical analysis, case studies and evaluation of emergent ICTs include:

The models that are called eventual consistency, characterized by asynchronous processes and based on event-driven system architecture, have evident advantages in terms of scalability and request per second rate. However, they also have a problem of temporary data incoherencies and are susceptible to it mostly under conditions of high consumption rates and network fragmentation. Despite this, some of these problems can be solved with certain architectures like event sourcing and CQRS, but dealing with eventual consistency is still a challenging task that one should take into account and have a proper error-handling policy.

Distributed transactions also cut across services and are mostly used in Two-Phase Commit (2PC) protocol helps to offer strong consistence as all services will commit on a given transaction or rollback it. Thus, 2PC is rather good for the data integrity issues, at the same time, it provokes certain performance issues, including time latency, and the throughput rate declines even more at the high load or networks failures. This makes it less suitable for real time and highly Scalable applications.

The Saga pattern is a reasonable approach as it maintains consistency and performance through further fragmenting of the transaction into a series of smaller transactions which are compensated. It does handle distributed work-flows and preserves consistency of data, with less number of data anomalies in certain situations as compared to eventual consistency models. However, more complications are involved when it comes to the procedure of implementing and testing compensating transactions because they make the system design more cumbersome.

The discussion of new concerns, such as DLT or consensus algorithms like Raft or a similar one, reveals potential to improve the synchronization of microservices' data. DLT offers the features of permanent, unchangeable data storage and high credibility while at the same time reducing the speed and processing capacity. Raft algorithms achieve both high levels of consistency and decent performance in terms of performance overhead, and are therefore well suitable for dealing with the problem of distributed state.

In general, the study shows that consistency does not have a simple answer in microservices. Based on these scenarios, Borgida suggests that one should choose the consistency model or the particular technology depending on the requirements to the system's performance and reliability. Such decision makers require a proper analysis of likely trade-offs between consistency, availability, and performance in a given microservices architecture when choosing between the mentioned approaches. Some areas for future study may include examining the hybrid models and progression in the development of the emerged technologies to persisting issues related to data synchronization and data control.

References

[1] Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., ... & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature

review. *Information and software technology*, *131*, 106449.

[2] Laigner, R., Zhou, Y., Salles, M. A. V., Liu, Y., & Kalinowski, M. (2021). Data management in microservices: State of the practice, challenges, and research directions. *arXiv preprint arXiv:2103.00170*.

[3] Ghani, I., Wan-Kadir, W. M., Mustafa, A., & Babir, M. I. (2019). Microservice testing approaches: A systematic literature review. *International Journal of Integrated Engineering*, *11*(8), 65-80.

[4] Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, *182*, 111061.

[5] Waseem, M., Liang, P., Shahin, M., Ahmad, A., & Nassab, A. R. (2021, June). On the nature of issues in five open source microservices systems: An empirical study. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (pp. 201-210).

[6] Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, *182*, 111061.

[7] Ntentos, E., Zdun, U., Plakidas, K., Schall, D., Li, F., & Meixner, S. (2019). Supporting architectural decision making on data management in microservice architectures. In *Software Architecture: 13th European Conference, ECSA 2019, Paris, France, September 9–13, 2019, Proceedings 13* (pp. 20-36). Springer International Publishing.

[8] Munonye, K., & Martinek, P. (2020, June). Evaluation of data storage patterns in microservices archicture. In *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)* (pp. 373-380). IEEE.

[9] Hannousse, A., & Yahiouche, S. (2021). Securing microservices and microservice architectures: A systematic mapping study. *Computer Science Review*, *41*, 100415.

[10] Koschel, A., Hausotter, A., Lange, M., & Gottwald, S. (2020). Keep it in Sync! Consistency Approaches for Microservices-An Insurance Case Study. In *SERVICE COMPUTATION 2020, The Twelfth International Conference on Advanced Service Computing* (pp. 7-14). IARIA.

[11] Cerny, T., Svacina, J., Das, D., Bushong, V., Bures, M., Tisnovsky, P., ... & Huang, J. (2020). On code analysis opportunities and challenges for enterprise systems and microservices. *IEEE access*, *8*, 159449-159470.

[12] Niloy, S. I., Ishmum, M. N., & Islam, M. A. (2022). *Data Consistency in Large Scale Applications* (Doctoral dissertation, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Board Bazar, Gazipur, Bangladesh).

[13] Bogner, J. (2020). *On the evolvability assurance of microservices: metrics, scenarios, and patterns* (Doctoral dissertation, Universität Stuttgart).

[14] Dakić, P. (2024). Software compliance in various industries using CI/CD, dynamic microservices, and containers. *Open Computer Science*, *14*(1), 20240013.

[15] Zarza, A. (2022). *Inter-Organizational Data Consistency through Blockchain-Supported Microservices* (Doctoral dissertation, University of Applied Sciences).

[16] Sharma, I., & Ramkumar, K. R. (2017). A survey on ACO based multipath routing algorithms for ad hoc networks. *International Journal of Pervasive Computing and Communications*, *13*(4), 370-385.

[17] Shamas, S., Panda, S. N., & Sharma, I. (2022, November). K-Means clustering using fuzzy C-Means based image segmentation for Lung Cancer. In *2022 3rd International conference on computation, automation and knowledge management (ICCAKM)* (pp. 1-5). IEEE.

[18] Sharma, I., Saini, J., Chhabra, G., & Kaushik, K. (2023, December). Cyber Threat Detection in Software-Defined Networks: An Empirical Analysis of Machine Learning Methods. In *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)* (pp. 1119-1124). IEEE.

[19] Bhakhri, K., Sethi, M., Sharma, I., & Kaushik, K. (2023, November). Examining the Consequences of Cyberattacks on Businesses and Organizations. In *International Conference on Innovations in Data Analytics* (pp. 227-239). Singapore: Springer Nature Singapore.

[20] Sharma, I., Kaur, A., Kaushik, K., & Chhabra, G. (2023, July). Machine Learning-Based Detection of API Security Attacks. In *International Conference on Data Science and Applications* (pp. 285-297). Singapore: Springer Nature Singapore.