IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

Achieving High-Speed Data Consistency in Financial Microservices Platforms Using NoSQL Using Nosql (Mongodb, Redis) Technologies

Bhushan Chaudhari, Santhosh Chitraju Gopal Verma, Srinivasa Rao Somu
Independent Researcher
bhushan.chaudhari@gmail.com and schitraju36@gmail.com

Abstract: *The transition from monolithic to microservices architecture has revolutionized software development, enabling greater scalability, flexibility, and deployment agility, especially in complex and high-demand sectors like financial services. However, this architectural evolution introduces significant challenges related to data consistency and integrity across distributed services. This review explores the integration of NoSQL databases—specifically MongoDB and Redis—within microservices-based financial platforms, addressing how they enhance data consistency, availability, and performance. By analyzing microservices principles, design challenges, and NoSQL databases' function in contemporary data architectures, the paper illustrates strategic approaches such as polyglot persistence, real-time analytics, and distributed data management techniques. A detailed comparison between MongoDB and Redis is presented, highlighting their use cases, scalability, and consistency models. Finally, this review demonstrates how NoSQL technologies enable scalable, independent, and resilient data management in financial microservices systems.*

Keywords: Microservices Architecture, Monolithic Architecture, Data Consistency, Financial Platforms, NoSQL Databases, MongoDB, Redis

I. INTRODUCTION

The move to microservices from monolithic architecture has allowed large technology companies to improve their scalability, agility, and deployment efficiency. The inherent challenge of working with independently deployed services is maintaining data consistency, particularly in speed and consistency-critical financial platforms. In this paper, explored the different approaches to data consistency and how NoSQL technologies, in particular MongoDB and Redis, can be used to address these issues. Explored microservices principles, the significance of data consistency, and finally, the flexibility of NoSQL systems to highlight the possibilities for reliable high-performance data storage in financial microservices architectures.

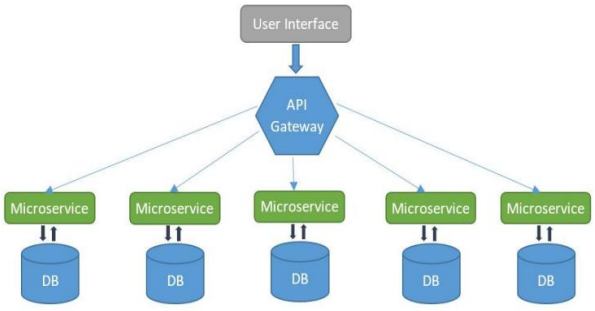

Figure 1: Microservice Architecture

As Martin Fowler notes, Netflix, eBay, Amazon, Twitter, PayPal, and other computer and programming giants have switched from monolithic to microservices architectures, demonstrating the numerous advantages microservices provide Agile and DevOps teams. Microservices lacks a clear definition and a common model that can be learnt and

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

applied to all systems using the microservices architecture style [1]. However, it was shown that the majority of microservices shared certain traits. Figure 1 illustrates a typical microservices use structure; a microservice sits between a database and an API endpoint.

Data integrity and trustworthy decision-making depend on data consistency, which guarantees correctness and coherence across several systems. It is among the eight aspects of data quality. If two or more values are the same in multiple places, the data is said to be consistent. On the other hand, inconsistent data can erode these advantages, resulting in compromised data integrity and untrustworthy decision-making [2]. One characteristic of a distributed shared data storage system is data consistency. It outlines the conduct that is permitted in relation to data access activities. Sequential legacy source code can access data in a database, shared memory, or repository without experiencing problems with data consistency. Data consistency issues arise when such sequential programming is migrated to microservices since several microservice instances communicate with a data store concurrently.

The term "Not Only SQL," or "NoSQL," describes a broad and well-known category of non-relational data management systems where databases are not primarily Table-based and seldom use SQL for data manipulation. When handling massive volumes of data, NoSQL database management systems are beneficial since they do not require a relational paradigm [3]. As a result, non-relational databases, or NoSQL databases, were developed. NoSQL databases can quickly transform odd data and circumvent SQL's rigidity by replacing "organized" storage with more flexible storage.

A. Structure of the Paper

The structure of the paper is as follows: A summary of the Microservices Architecture is given in Section II. The incorporation of NoSQL into contemporary data systems is covered in Section III. Scalable data management in financial microservices is examined in Section IV. Organizational data processing issues are examined in Section V. A review of pertinent literature is provided in Section VI. Future research directions are discussed in Section VII's conclusion.

II. OVERVIEW OF MICROSERVICES ARCHITECTURE

The term "microservices architecture" describes an application development architectural approach. A big application may be divided into smaller, independent components using microservices, each of which has its own area of responsibility. A microservices-based application may invoke several internal microservices to build its answer to a single user request. A good example of a microservices architecture is a container, which allows you to concentrate on creating the services rather than managing dependencies. Containers are typically used to build microservices in modern cloud-native applications [4].

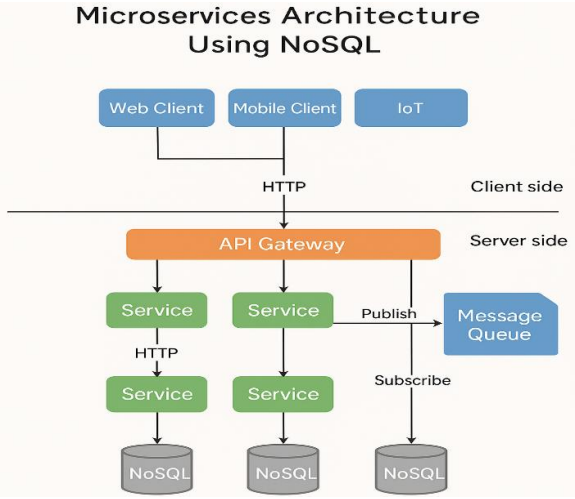

Figure 2: Microservices Architecture Using NoSQL

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

A typical microservices architecture utilizing NoSQL databases to achieve scalability, flexibility, and high-speed data consistency is shown in Figure 2. To ensure efficient, decoupled communication between services, a Message Queue is employed, supporting both publish and subscribe mechanisms [5]. This enhances real-time data handling and improves system responsiveness, a critical need for financial applications operating at high volumes. The architecture is divided into two primary layers:

- **Client Side:** Interacts with the system through various interfaces including Web Client, Mobile Client, and IoT devices, all communicating via HTTP protocols [6].
- **Server Side:** Requests are routed through an the main point of entry for all incoming client traffic is the API Gateway. It forwards requests to corresponding independent services, which are responsible for processing the requests and interacting with their associated NoSQL databases.

A. Core Design Principles and Challenges of Microservices Architecture

Microservices architecture introduces a paradigm shift in the way modern applications are developed and deployed. It promotes modularity, agility, and scalability attributes that are critical for real-time, high-availability systems like financial platforms [7]. However, with these advantages come specific design principles to follow and challenges to address.

Single Responsibility Principle

Each microservice should have a single responsibility, enlarging only a single, well defined business capability. In this case, this is a good follow up from object-oriented design, as this principle ensures that services stay focused and within reason.

Decentralized Governance

The Microservices architecture advocates decentralization of technology governance. That is, the development teams have the freedom to select the most appropriate programming languages, frameworks, databases, etc., as per requirements of the service they are building.

Independent Deploy Ability

A core aspect of the micro service manifesto and one of the most critical foundational technologies is independent deploy ability. It helps in releasing quickly and at the same time decreases the risk in monolithic large releases.

Fault Isolation

Microservices are fault tolerant. It should not be possible for one service to take down the whole application if one fails. Techniques like circuit breakers, timeouts and retries help isolate this fault such that cascading failure is avoided and the system is able to keep functioning when individual services are having issues.

Data Management Challenges in Microservices

While microservices bring in a lot of advantages, their use also complicates things, especially on the data management fronts. While microservices are distributed, they present inherent challenges of data consistency, transaction management and schema evolution.

III. INTEGRATION OF NOSQL IN MODERN DATA ARCHITECTURES

The integration of NoSQL in modern data architectures is a key development in handling the increasing volume, variety, and velocity of data. Unlike traditional relational databases, NoSQL offers flexibility in managing unstructured, semi-structured, or rapidly changing data [8][9]. By supporting schema-less design, horizontal scaling, and distributed data storage, NoSQL databases are well-suited for applications requiring high performance, real-time processing, and scalability. They play a critical role in modern data architectures, particularly in big data environments, cloud computing, and Internet of Things (IoT) systems, where traditional relational databases may struggle to meet demands efficiently.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

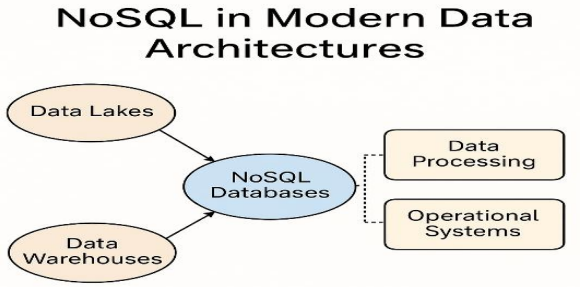

Figure 3: NoSQL in Modern Data Architectures

The function of NoSQL databases in contemporary data structures. NoSQL databases act as a central component that integrates both Data Lakes and Data Warehouses, facilitating seamless intake and administration of vast amounts of varied data. From this central hub, data is channeled to Data Processing systems for analysis and to Operational Systems for real-time transactional support shown in Figure 3. This architectural model highlights the flexibility and scalability of NoSQL databases in managing data that is semi-structured, unstructured, and structured, which makes them a crucial part of contemporary data-driven environments [10][11].

A. Key Strategies for Integrating NoSQL in Modern Data Architectures

Integrating NoSQL databases into modern data architectures requires a strategic approach to maximize their strengths in handling diverse data types and large-scale environments. Key strategies include [12]:

- **Polyglot Persistence:** Polyglot persistence is the practice of utilizing many database types in a single system or application to capitalize on their individual advantages. This method improves flexibility and performance by enabling organizations to select the optimal database technology for every use case.
- **Data Lakes and Data Warehouses:** In data lakes and data warehouses, modern data architectures frequently integrate SQL and NoSQL databases. Because NoSQL databases are scalable and flexible, they are frequently used in data lakes, which hold unstructured, raw data. Because of their powerful querying capabilities, SQL databases are frequently used in data warehouses, which hold organized and processed data.
- **Microservices Architecture:** The design of microservices encourages the development of applications as a group of loosely linked services [13]. The database technology that best suits each microservice's requirements can be used. For example, a logging service may utilize a NoSQL database due to its rapid write throughput, but a user authentication service may use a SQL database for its transactional needs.
- **Real-Time Analytics and Big Data: Big** data applications and real-time analytics gain from the combination of SQL and NoSQL databases. NoSQL databases provide the scalability to manage massive amounts of rapidly changing data, whereas SQL databases offer dependable and consistent transaction processing. With this combination, businesses can process and analyses data instantly, obtaining insights in real time.

B. Comparison of MongoDB and Redis

MongoDB and Redis are two widely used NoSQL databases, but they are created with various use cases in mind, providing unique benefits and features [14]. Here's a comparison of the two shown in Table I:

Comparison of MongoDB and Redis in Financial Microservices Architectures

Feature / Criteria	MongoDB	Redis
Type	Document-based NoSQL Database	In-memory Key-Value Store
Primary Use Case	Storage and retrieval of semi-structured data (JSON)	Caching, real-time analytics, session management
Persistence	Persistent storage with optional in-memory performance boost	Primarily in-memory with disk persistence options
Speed	High performance for CRUD operations	Extremely fast due to in-memory data access

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

Scalability	Horizontal scaling via sharding	Horizontal scaling via clustering and partitioning
Replication	Supports replica sets for high availability	Master-slave and cluster-based replication
Language Support	Rich query language with indexing support	Simple commands and scripting with Lua
Performance Optimization	Indexes, aggregation pipelines, flexible schema	In-memory data access, TTL (Time to Live) keys

C. Data Consistency Techniques in NoSQL

In a distributed system that operates over a wide network it may happen that some nodes disappear from time to time [15]. In their example of a social networking application if it's running on a relational database then due to unavailability of any node will result in breaking some or many parts of the application as a traditional relational database enforces strong consistency [16].

- **Read Your Own Writes Consistency:** As the name suggests, a client can see updates done by him instantly while he can neither see updates done by others nor others can see updates done by him immediately.
- **Session Consistency:** This type is same as type (1) with the distinction is that a client can only view his updates if the read request and write request are completed during the same session.
- **Casual Consistency:** The sequence in which other clients completed memory operations, such as reads and writes, is visible if they are tangentially linked. The sequence in which the processes are carried out may alter if they are not informally connected.
- **Monotonic Read Consistency:** This kind of consistency ensures that clients will read updated data in a gradual order in future requests. It implies that a newly written value is allowed to replicate before it is read by any other client.
- **Partitioning:** Partitioning can be used when data has to be duplicated for dependability or when it surpasses a single machine's capacity. Additionally, it is used while load balancing. Data partitioning must be used if, as an example, their social networking service draws a large number of users, leading to massive data collection as a result of increasing activity.

IV. ENABLING SCALABLE AND INDEPENDENT DATA MANAGEMENT IN FINANCIAL MICROSERVICES USING NOSQL

In the context of modern financial systems, microservices architecture allows different components of a financial application, such as user management, transaction processing, fraud detection [17][18], and reporting, to function independently while maintaining overall system cohesion. NoSQL databases are essential for facilitating scalability, flexible, and independent data management in these systems. Here's how:

A. Scalability Through Sharding and Replication

Scalability was a priority while designing NoSQL databases. Therefore, they grow horizontally by breaking the data down to be distributed across hardware, where the database can collect and process it [19]. Complicated tasks such as sharing and replication make this possible.

- **Horizontal Scaling with NoSQL:** NoSQL databases for horizontal scaling means no database scales beyond the capacities of one server; therefore, NoSQL Databases provide horizontal scaling by sharing, which is the process of dividing data among several servers.
- **Data Replication for High Availability:** Replication entails creating multiple copies of the data and saving them onto multiple servers for the goal of high availability. This enables the system to continue operating even in the event that some of the servers fail.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

B. Data Independence and Polyglot Persistence

Another huge benefit NoSQL databases provide in a microservices architecture is data independence and polyglot persistence. This implies that each microservice can select the database type and schema that best fit its unique requirements, whether that's a document store for flexible JSON data, a graph database for intricate connections, or a key value store for straightforward caching.

- **Choosing the Right Database for Each Service:** With microservices architecture, along with NoSQL databases, you can select the database suitable for that service. For instance, if you are building a user session service, for fast access you can pick up a key value store like Redis, or for schema flexibility if it's a product data service, a document database like MongoDB [20].
- **Schema Flexibility and Evolution:** NoSQL databases, in particular document and wide-column stores, have flexible schemas which are easy to evolve over time. In a microservices environment services are developed and updated independently so this is very important.

C. Fault Tolerance and Resilience

NoSQL databases are by design fault tolerant and resilient which is critical in a distributed microservices world.

- **Handling Data Store Failures:** 'Easily failing' is a design principle of NoSQL databases. They can continue to operate by means of replication and automatic failover mechanisms even if some of the servers in the cluster become unavailable.
- **Data Replication and Disaster Recovery:** In addition to offering availability, data replication is of great importance to disaster recovery. Organizations maintain multiple copies of the data at different locations to allow quick recovery from data center outage, or some other catastrophic events [21].

D. Performance Optimization in NoSQL

NoSQL performance optimization is important because, without it, microservices will not be able to endure high traffic loads and provide a smooth user experience [7].

- **Data Modeling for Performance:** As such, keys are used to achieve optimal performance in NoSQL databases; however, effective data modeling is critical for that performance to be realized. Unlike relational databases that normally tend to normalize data, NoSQL databases need denormalization in order to minimize need for joins, which are expensive in a distributed environment.
- **Indexing Strategies:** Proper indexing is important for fast query performance in any database and NoSQL databases can’t take a pass on this either. Each NoSQL database supports different indexing options, and they can decide the right index based on the application’s query patterns. Proper indexing can save on query latency and improve the whole system as a whole.

V. ORGANIZATIONAL CHALLENGES IN DATA PROCESSING IN FINANCIAL SERVICES

Financial services data processing is intricate and heavily regulated, including vast amounts of historical, sensitive, and real-time data [22]. Several challenges can impact efficiency, accuracy, and security in these environments:

A. Data Silos Caused by Organizational Silos

One major obstacle to making business choices is data silos. They result from departmental silos that are often local in nature, such as those found in marketing, operations, finance, and risk. The broad picture is lost, which hinders both internal teamwork and a client's whole perspective. These departments provide the data that is essential to the creation and redesign of agile, iterative products and processes.

B. The Increasing Need for Real-Time Processing

The majority of data processing in conventional systems is batch-based. Over time, as digitization has risen, the emphasis has turned to improving the consumer experience. Better customer experiences in financial services are shown by real-time product and solution discovery, pricing, personalization, and process adaption. SLAs and quicker

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

transaction response times are also important requirements. This has made it necessary to bring analytics and data closer to online processing platforms [23]. Traditional systems, however, are not made for these kinds of highly scaled operations.

C. Always-on Availability and Performance

Mission-critical applications require constant availability, particularly those that interact with consumers. As technology has advanced, more and more interactions and procedures have shifted to digital platforms. Performance and availability have become more important as a result of growing volumes and the shift to distributed architectures. Despite the lengthy history of load balancing and clustering, there is a need for contemporary approaches to their management. Fault tolerance and load balancing that are automated and algorithmically handled are urgently needed.

D. Consistency of Data

Data integrity becomes crucial when NoSQL databases that coexist with relational databases are used more and more in current systems. Data may undergo unanticipated changes during storage, retrieval, and processing, such as deletion, conflicting writes, or stale reads, which might cause inconsistent data across systems. Historically, older NoSQL suppliers have not been able to create highly consistent real-time systems [24].

E. Processing data for AI/ML in real-time

Streaming data continuously is necessary for real-time ML algorithms. The organization must aggregate historical data from many sources and use ML algorithms in real time. In fraud analysis, for instance, machine learning helps identify and discover millions of trends in a user's past purchases rather than just a select handful that are recorded by rules. Machine learning algorithms applied to raw data can anticipate fraud in a large number of transactions.

VI. LITERATURE OF REVIEW

A thorough analysis of academic studies and literature on Achieving High-Speed Data Consistency in Financial Microservices Platforms Using NoSQL is included in this part, along with the Executive Summary in Table II.

In aims Alflahi, Mohammed and Alsammani (2024) As contrast to the strong consistency ensured by ACID principles in RDBMS, to improve data consistency in NoSQL databases, which are often constructed with BASE features. They present a thorough four-phase server-side paradigm specifically designed for MongoDB. Transaction management, read and write transaction bifurcation, transaction readiness evaluation, and transaction execution using a customized locking mechanism are all covered by this approach. In particular, the mean read, update, and throughput latencies. These findings show how well their suggested strategy works to improve consistency across document-based NoSQL databases like MongoDB as well as other NoSQL database variations including key-value, graph, and wide-column stores [25].

Isha and Vishalakshmi Prabhu (2023) Microservices architecture (MSA) is a widely used technique for building autonomous and maintainable components of big systems. Numerous projects have been completed recently to migrate monolithic systems into MSA and use it in future development areas such as the IoT. The majority of current effort is on MSA for the system, with but little consideration given to the architecture used to implement the underlying services. A kind of Domain Driven Design (DDD) called clean architecture (CA) divides business logic into distinct components for independent testing. In Android systems, clean architecture is frequently utilized [26].

Mallidi, Sharma and Vangala (2022) Scalability, message retention, numerous consumers, replication, message order, and TCP protocol support are some of the benefits that set streaming systems apart from message queues. In order to obtain data that is almost real-time, banking and financial application backend applications—particularly batch processing systems—move from batch to stream platforms. Consequently, banking and settlement systems reduce the processing/settlement processing time [27].

The Sanchez, Bernal and Parada (2021) is It starts by combining the many kinds of vulnerabilities, assaults, and defenses specific to MongoDB, Redis, and Apache Cassandra. This is done in accordance with the sequence of the goals. A web system prototype that uses a non-relational database to store data will be created and then subjected to a series of attacks outlined in a test plan that aim to add, consult, alter, or remove data. A report detailing the assaults,

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

their application, their outcomes, potential defenses, the security benefits and drawbacks for each management, and the conclusions drawn will be given at the end [28].

In propose Ouyang et al. (2021) and Analyze MongoDB transaction protocols' effective white-box testing procedures in relation to their consistency guarantees. MongoDB is a well-known distributed, document-oriented, general-purpose NoSQL database. Both the formal specification of the consistency guarantees that MongoDB purported to offer and succinct pseudocode of MongoDB transactions in each deployment are absent. they explicitly define and validate MongoDB's transactional consistency algorithms in this study. In particular, they offer a succinct pseudocode for the transactional consistency protocols Wired Tiger, Replicase, and ShardedCluster found in every MongoDB implementation [29].

Table II provides a comparative executive summary of recent research on high-speed data consistency in NoSQL-based financial microservices platforms, emphasizing important areas of interest, conclusions, and research problems

Table II: Executive Summary on Achieving High-Speed Data Consistency in Financial Microservices Platforms Using NoSQL

Reference	Key Focus	Key Findings	Challenges (Limitations & Future Gap)
Alflahi, Mohammed and Alsammani, 2024	Enhancing data consistency in NoSQL (MongoDB)	Introduced a four-stage server-side model (transaction management, read/write bifurcation, readiness assessment, execution with locking). Improved throughput and latency, with broader applicability to key-value, graph, and wide-column NoSQL databases.	Real-world implementation across diverse NoSQL types needs validation. Handling concurrent, large-scale transactions over distributed systems still requires exploration.
Isha and Vishalakshmi Prabhu, 2023	Clean Architecture (CA) within Microservices Architecture (MSA)	Advocates for CA to decouple business logic for independent testing and maintainability, particularly relevant in Android and IoT systems.	Minimal focus on the internal structure and architecture of microservices. Lack of comprehensive comparative studies with alternative service designs.
Mallidi, Sharma and Vangala, 2022	Migration from batch to stream processing in financial applications	Streaming platforms (vs. Message Queues) provide scalability, message retention, ordering, and TCP support—enhancing real-time capabilities in banking systems.	Migration complexities and cost implications not deeply analyzed. Limited case studies on system-specific implementation strategies.
Sanchez, Bernal and Parada, 2021	Security vulnerabilities in NoSQL databases (MongoDB, Redis, Cassandra)	Developed a prototype web system to simulate attacks on NoSQL databases and evaluate protection mechanisms. Delivered a comprehensive report on attack execution, countermeasures, and security pros/cons.	Limited to experimental attacks; lacks real-time detection methods. Broader ecosystem security integration not addressed.
Ouyang et al., 2021	Verification of MongoDB consistency protocols	Formally specified transactional consistency protocols for WiredTiger, Replicaset, and ShardedCluster. Provided pseudocode for each MongoDB deployment.	MongoDB's internal documentation gaps necessitated manual reconstruction. Future work is needed on cross-database transactional consistency and scaling consistency verification.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

VII. CONCLUSION AND FUTURE WORK

Microservices architecture offers significant advantages in design of contemporary financial platforms, providing modularity, flexibility, and effective deployment pipelines. It also presents challenges regarding data consistency due to the nature of distributed and decoupled services. In this paper, analyzed two leading NoSQL technologies: MongoDB and Redis, as a way to address these challenges. MongoDB provides a schema-less, document-based approach that works well for different and growing data, while Redis is a high speed (in-memory) data store, intended for use in real-time applications, like shopping carts and session state. Through layering service capabilities with caching and easy access to performance underpinned by distributed NoSQL databases, financial systems can have both performance and data reliability.

Overall, the significance of selecting corresponding storage mechanisms to microservices is underscored through our findings, and that if eventual consistency is handled well, it does not reflect on system trust. Ultimately, this hybrid approach serves as a valuable means to approach rigorous, high-performance and consistent data management in critical financial applications.

Future work can look into the development of automated methods of checking consistency that fit into microservices with NoSQL systems. Increasing support for ACID-like guarantees with NoSQL systems is an interesting approach that can provide opportunities without sacrificing performance. Moreover, studying hybrid storage - using relational and non-relational databases together in microservices - can open modern compromises against both speed and consistency. The investigation of utilizing machine learning based approaches to predict possible consistency conflicts and resolve in a distributed architecture when appropriate is additionally an area for future consideration. As financial infrastructures continue to scale globally, future investigations should look to consistency across geographical regions of data, encryption, and the validation of real-time transactions. Understanding the comparative performance of NoSQL databases at different workloads and conditions would provide understanding for system architects in financial environments.

REFERENCES

[1] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, and A. Barros, “Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency,” IEEE Softw., vol. 35, no. 3, pp. 63–72, 2018, doi: 10.1109/MS.2017.440134612.

[2] P. Ranjan Bhowmick, “Overview of Nosql Databases,” 2021.

[3] Vashudhar Sai Thokala, “Scalable Cloud Deployment and Automation for ECommerce Platforms Using AWS, Heroku, and Ruby on Rails,” Int. J. Adv. Res. Sci. Commun. Technol., vol. 3, no. 2, pp. 349–362, Oct. 2023, doi: 10.48175/IJARSCT-13555A.

[4] O. Al-Debagy and P. Martinek, “A Comparative Review of Microservices and Monolithic Architectures,” in 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), 2018, pp. 149–154. doi: 10.1109/CINTI.2018.8928192.

[5] V. Bushong et al., “On microservice analysis and architecture evolution: A systematic mapping study,” Appl. Sci., vol. 11, no. 17, 2021, doi: 10.3390/app11177856.

[6] P. Chatterjee, “Cloud-Native Architecture for High-Performance Payment System,” vol. 10, no. 4, pp. 345–358, 2023.

[7] M. K. Goyal and R. Chaturvedi, “The Role of NoSQL in Microservices Architecture: Enabling Scalability and Data Independence,” 2022.

[8] A. O. Thakare, O. W. Tembhurne, A. R. Thakare, and S. Narasimha Reddy, “NoSQL Databases: Modern Data Systems for Big Data Analytics,” Int. J. Electr. Comput. Eng. Syst., vol. 14, no. 2, pp. 207–216, 2023, doi: 10.32985/ijeces.14.2.10.

[9] A. W. S. Aurora, S. Pillai, and V. S. Thokala, “Data synchronisation strategies for distributed web applications using MySQL, MongoDB and AWS Aurora,” vol. 09, no. 01, pp. 779–793, 2023.

[10] A. Moko and P. Asagba, “Big Data and NoSQL Databases Architecture: A Review,” vol. 7, p. 9, 2021.

[11] S. Murri, “Data Security Environments Challenges and Solutions in Big Data,” Int. J. Curr. Eng. Technol., vol. 12, no. 6, pp. 565–574, 2022.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.53
Volume 4, Issue 2, June 2024

[12] Vasudhar Sai Thokala, "Efficient Data Modeling and Storage Solutions with SQL and NoSQL Databases in Web Applications," Int. J. Adv. Res. Sci. Commun. Technol., vol. 2, no. 1, pp. 470–482, Apr. 2022, doi: 10.48175/IJARSCT-3861B.
[13] S. S. S. Neeli, "Leveraging Docker and Kubernetes for Enhanced Database Management," J. Artif. Intell. Mach. Learn. Data Sci., vol. 1, no. 1, p. 5, 2022.
[14] N. Malali, "Using Machine Learning To Optimize Life Insurance Claim Triage Processes Via Anomaly Detection In Databricks: Prioritizing High-Risk Claims For Human Review," Int. J. Eng. Technol. Res. Manag., vol. 6, no. 6, pp. 2456–9348, 2022.
[15] M. Saxenna and V. Singh, "NoSQL Databases- Analysis, Techniques, and Classification," J. Adv. Database Manag. Syst. (EISSN 2393-8730), vol. 1, pp. 1–11, 2014.
[16] A. Goyal, "Optimising Software Lifecycle Management through Predictive Maintenance : Insights and Best Practices," Int. J. Sci. Res. Arch., vol. 07, no. 02, pp. 693–702, 2022.
[17] N. Malali, "Using Machine Learning To Optimize Life Insurance Claim Triage Processes Via Anomaly Detection In Databricks: Prioritizing High-Risk Claims For Human Review," Int. J. Eng. Technol. Res. Manag., vol. 6, no. 6, 2022, doi: https://doi.org/10.5281/zenodo.15176507.
[18] V. Pillai, "Anomaly Detection for Innovators: Transforming Data into Breakthroughs," Lib. Media Priv. Ltd., 2022.
[19] A. Gogineni, "Artificial intelligence-Driven Fault Tolerance Mechanisms for Distributed Systems Using Deep Learning Model," J. Artif. Intell. Mach. Learn. Data Sci., vol. 1, no. 4, 2023.
[20] S. Shah and M. Shah, "Deep Reinforcement Learning For Scalable Task Scheduling In Serverless Computing,"," Int. Res. J. Mod. Eng. Technol. Sci., vol. 3, no. 12, pp. 1845–1853, 2021.
[21] S. S. S. Neeli, "Optimizing Database Management with DevOps: Strategies and Real-World Examples," J. Adv. Dev. Res., vol. 11, no. 1, p. 8, 2020.
[22] A. Gogineni, "Multi-Cloud Deployment with Kubernetes: Challenges, Strategies, and Performance Optimization," Int. Sci. J. Eng. Manag., vol. 1, no. 2, 2022.
[23] Godavari Modalavalasa, "The Role of DevOps in Streamlining Software Delivery: Key Practices for Seamless CI/CD," Int. J. Adv. Res. Sci. Commun. Technol., vol. 1, no. 12, pp. 258–267, Jan. 2021, doi: 10.48175/IJARSCT-8978C.
[24] A. G. Milavkumar Shah, "Distributed Query Optimization for Petabyte-Scale Databases," Int. J. Recent Innov. Trends Comput. Commun., vol. 10, no. 10, pp. 223–231, 2022.
[25] A. A. E. Alflahi, M. A. Y. Mohammed, and A. Alsammani, "Enhancement of Database Access Performance by Improving Data Consistency in a Non-relational Database System (NoSQL)," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 14816 LNCS, pp. 194–205, 2024, doi: 10.1007/978-3-031-65223-3_13.
[26] P. V. Isha and H. Vishalakshmi Prabhu, "An Approach to Clean Architecture for Microservices Using Python," in 7th IEEE International Conference on Computational Systems and Information Technology for Sustainable Solutions, CSITSS 2023 - Proceedings, 2023. doi: 10.1109/CSITSS60515.2023.10334229.
[27] R. K. Mallidi, M. Sharma, and S. R. Vangala, "Streaming Platform Implementation in Banking and Financial Systems," in 2022 2nd Asian Conference on Innovation in Technology, ASIANCON 2022, 2022. doi: 10.1109/ASIANCON55314.2022.9909500.
[28] R. A. G. Sanchez, D. J. M. Bernal, and H. D. J. Parada, "Security assessment of Nosql Mongodb, Redis and Cassandra database managers," in 2021 7th Congreso Internacional de Innovacion y Tendencias en Ingenieria, CONIITI 2021 - Conference Proceedings, 2021. doi: 10.1109/CONIITI53815.2021.9619597.
[29] H. Ouyang, H. Wei, Y. Huang, H. Li, and A. Pan, "Verifying Transactional Consistency of MongoDB," no. 23, pp. 1–23, 2021.