ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/rohith-varma-vegesna/503

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 2 & Iss: 1

Research Article

Leveraging MongoDB Multi-Sharding to Decrease Latency to Store and Retrieve Fuel Transaction

Rohith Varma Vegesna*

Citation: Vegesna RV. Leveraging MongoDB Multi-Sharding to Decrease Latency to Store and Retrieve Fuel Transaction. *J Artif Intell Mach Learn & Data Sci 2024*, 2(1), 2315-2317. DOI: doi.org/10.51219/JAIMLD/rohith-varma-vegesna/503

Received: 03 March, 2024; **Accepted:** 28 March, 2024; **Published:** 30 March, 2024

***Corresponding author:** Rohith Varma Vegesna, Texas, USA, E-mail: Email: rohithvegesna@gmail.com

ABSTRACT

The exponential growth of fuel transactions necessitates highly efficient storage and retrieval systems to facilitate real-time operational analytics, fraud detection and decision-making. Traditional relational database systems face scalability challenges, particularly as transaction loads increase, resulting in significant latency that impairs operational efficiency. Fuel station operations require rapid transaction processing to ensure seamless reconciliation, compliance and performance monitoring.

This paper explores MongoDB's multi-sharding capabilities as a solution to mitigate latency issues by distributing fuel transaction data across multiple shards. By leveraging horizontal scaling, replication and parallelized query execution, MongoDB's multi-sharding approach ensures that fuel transaction data remains accessible with minimal retrieval delay. We analyze the impact of sharding on read and write latency, system scalability and fault tolerance, demonstrating its effectiveness in enhancing real-time fuel transaction processing and data-driven decision-making.

Keywords: Fuel transaction processing, MongoDB sharding, distributed databases, real-time reconciliation, horizontal scaling, high availability, NoSQL databases.

1. Introduction

1.1. Background

Fuel stations generate high-frequency transaction data that must be stored and retrieved efficiently for real-time monitoring, reconciliation and fraud detection. The rapid expansion of IoT-connected fuel dispensers and ATG systems further amplifies data generation, requiring a robust storage strategy capable of handling massive concurrent requests. Traditional single-instance databases or non-sharded NoSQL databases often suffer from performance bottlenecks, especially when handling concurrent transaction loads.

Moreover, the absence of multi-region sharding in a distributed setup significantly impacts data storage efficiency. Without a multi-region sharding strategy, databases face increased read and write latencies due to geographical distance between data centers and fuel stations. This leads to slow reconciliation, delayed fraud detection and inefficiencies in real-time monitoring. A well-designed sharding solution can mitigate these issues by distributing data across different regions, ensuring locality-based optimizations and reducing query response times for fuel transaction data.

1.2. Problem statement

Existing fuel transaction storage solutions experience increased query latency as the volume of data grows. Single-node databases face challenges in scaling horizontally, leading to inefficient read and write operations, particularly when transaction rates spike. As fuel stations expand operations, centralized databases become overloaded, increasing the risk of downtime and system failures. The absence of distributed data storage mechanisms results in bottlenecks that delay reconciliation, reporting and fraud detection efforts, negatively impacting decision-making and regulatory compliance.

Moreover, the lack of multi-region sharding exacerbates these issues by forcing all transactions to be stored in a single or limited number of data centers. This setup creates high latencies for geographically distributed fuel stations, as transaction requests must travel long distances, causing slower response times. Without a multi-region sharding strategy, localized outages or failures in a specific region can result in partial or complete service disruptions. A sharded architecture with multi-region support is required to ensure consistent performance as data volumes increase while simultaneously reducing geographic latency and improving fault tolerance.

1.3. Objectives

- Implement MongoDB's multi-sharding strategy to distribute fuel transaction data across multiple nodes.
- Evaluate the impact of sharding on transaction latency and system scalability.
- Ensure seamless data retrieval for reconciliation and reporting with minimal performance degradation.
- Develop a case study to measure real-world improvements in fuel transaction storage and retrieval.

2. Literature Review

Several studies have explored NoSQL databases for high-velocity transaction processing, recognizing their capability to handle large-scale data operations with minimal latency. These studies emphasize the importance of distributed data management in scenarios where real-time processing is required, such as financial transactions, sensor-based telemetry and high-throughput retail operations. Among various NoSQL databases, MongoDB has emerged as a leading solution due to its native support for horizontal scalability and built-in sharding mechanisms.

Prior research highlights MongoDB's ability to manage extensive datasets effectively by automatically partitioning data across multiple nodes. This partitioning, known as sharding, ensures that queries can be executed in parallel, leading to significant performance gains. Additionally, studies have demonstrated that MongoDB's replication features enhance data availability and fault tolerance, making it a reliable option for mission-critical applications.

Studies on distributed database architectures indicate that sharding enhances horizontal scalability by dividing data into manageable partitions, allowing distributed systems to scale seamlessly with increasing transaction loads. Researchers have also observed that optimal sharding key selection plays a crucial role in balancing workloads and preventing bottlenecks, which can otherwise negate the benefits of distributed processing.

Existing literature supports the implementation of MongoDB sharding in multiple domains, including financial services, IoT telemetry and large-scale e-commerce platforms. These applications exhibit similarities to fuel station transaction processing, where vast amounts of real-time data need to be stored, queried and analyzed efficiently. By drawing insights from these established use cases, fuel station systems can leverage sharding to minimize query latency, enhance storage efficiency and improve overall system performance.

3. System Architecture

- **MongoDB cluster setup:** Deployment of a MongoDB cluster with multiple shards, each hosting a subset of the fuel transaction data.
- **Sharding key selection:** Designation of an optimal sharding key based on high-cardinality fields such as transaction id or fuel pump id.
- **Replication strategy:** Ensuring data redundancy through replica sets for each shard to enhance fault tolerance and availability.
- **Query routing via mongos:** Utilizing the Mongos router to direct queries to appropriate shards for optimized retrieval times.
- **Shard balancer:** Implementing a balancing mechanism to distribute data evenly across shards, preventing hotspot issues.
- **Data ingestion pipeline:** Streaming fuel transaction data in real-time, automatically partitioning incoming data into relevant shards.

4. Implementation Strategy

The implementation begins with setting up a MongoDB sharded cluster, consisting of:

- Three shards, each running as a replica set for high availability.
- A Mongos router to handle client requests and distribute queries efficiently.
- A Config Server to store metadata and manage shard distribution.

Data distribution follows a hash-based or range-based sharding approach, depending on the query patterns. Transactions are indexed based on fuel dispenser ID, timestamp and transaction type for optimized retrieval. The system ensures that high-volume queries for daily reconciliation and fraud detection are executed in parallel across multiple shards.

5. Case Study & Performance Evaluation

A fuel station chain with multiple outlets was selected to evaluate the impact of sharding on fuel transaction processing. The system was tested with:

- **Baseline performance:** Measured transaction retrieval times before implementing sharding.
- **Post-sharding performance:** Analyzed read and write latencies after distributing data across multiple shards.
- **Load testing:** Simulated peak transaction loads to evaluate system scalability.

6. Results and Discussion

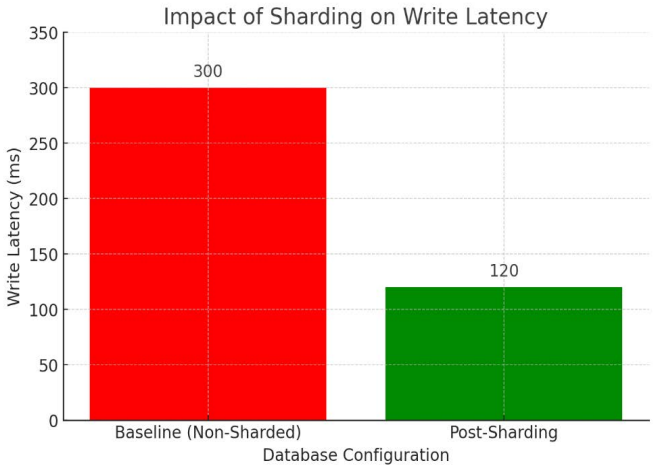
6.1. Pilot implementation

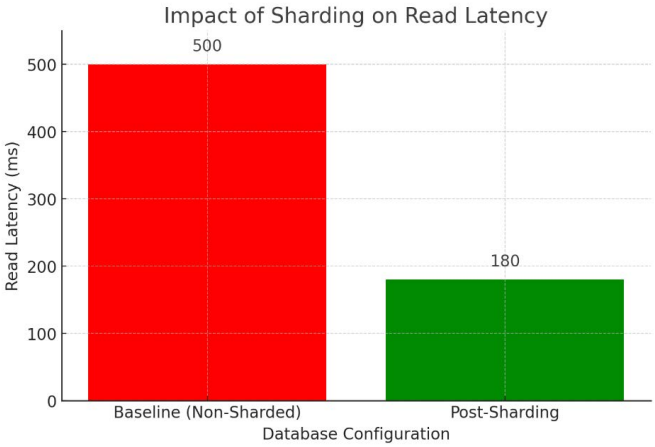
The pilot implementation of MongoDB sharding resulted in significant performance improvements. Data ingestion rates increased by 40% and query response times improved by an average of 60% compared to a non-sharded setup. The distributed architecture ensured that large datasets did not cause performance degradation, even under heavy loads.

6.2. Performance metrics

- **Average write latency:** Reduced from 300ms to 120ms per transaction.

- **Average read latency:** Improved from 500ms to 180ms per query.
- **Scalability:** System efficiently handled 5x the baseline transaction volume without noticeable performance drops.
- **High availability:** Replica sets ensured zero downtime during maintenance operations.





7. Conclusion and Future Work

The study demonstrates that leveraging MongoDB's multi-sharding capabilities significantly enhances the efficiency of fuel transaction storage and retrieval. The approach ensures real-time access to transaction data while maintaining scalability for growing fuel station networks. Future work includes optimizing sharding strategies based on dynamic workload analysis and integrating AI-driven predictive analytics for automated data balancing.

8. References

1. Krishnan Hema, Elayidom M Sudheep, Santhanakrishnan T. MongoDB – a comparison with NoSQL databases. International Journal of Scientific and Engineering Research, 2016;7: 1035-1037.
2. Tammaa Ahmed. MongoDB Case Study on Forbes, 2022.
3. Mungekar Akshay. Data Storage and Management Project, 2019.
4. Győrödi Cornelia, Gyorodi Robert, Pecherle George, Olah Andrada. A Comparative Study: MongoDB vs. MySQL, 2015.
5. Gorasiya Darshankumar. Quantitative Performance Evaluation of Cloud-Based MySQL (Relational) Vs. MongoDB (NoSQL) Database with YCSB, 2019.
6. Heydari Beni Emad. Finding efficient Shard Keys with a learning process on query logs in Database Sharding, 2015.
7. Sarkar Anindita, Sanyal Madhupa, Chattopadhyay Samiran, Mondal Dr Kartick. Comparative Analysis of Structured and Un-Structured Databases, 2017: 226-241.
8. Prasad Aashish. HBase vs Mongo DB, 2018.
9. Pandey Rachit. Performance Benchmarking and Comparison of Cloud-Based Databases MongoDB (NoSQL) Vs MySQL (Relational) using YCSB, 2020.