

Enhancement of Database Access Performance by Improving Data Consistency in a Non-relational Database System (NoSQL)

Adam A. E. Alflahi[1], Mohammed A. Y. Mohammed[2], and Abdallah Alsammani[3](✉)

[1] Department of Computer Science and Mathematics, Bahri University, Khartoum, Sudan
[2] Department of Mathematics and Statistics, Georgia State University, Atlanta, GA, USA
[3] Department of Mathematics, Jacksonville University, Jacksonville, FL, USA
aalsamm@ju.edu

Abstract. This study aims to improve data consistency in NoSQL databases, which traditionally follow BASE properties instead of the strong consistency guaranteed by ACID principles in RDBMS. We have developed a four-stage server-side model specifically for MongoDB to achieve this. Our model addresses transaction management, separation of read and write transactions, assessment of transaction readiness, and transaction execution using a specialized locking algorithm. We tested our model using the Yahoo Cloud Services Benchmark (YCSB), particularly for update-heavy workloads (A, B, and F). Our model showed significant improvements, with the average throughput, read, and update latencies improving to 2864.726 ms, 32806.275 ms, and 51845.629 ms, respectively, as compared to the baseline metrics of 2914.110 ms, 26510.930 ms, and 32457.662 ms. These results demonstrate that our proposed model effectively enhances consistency in document-based NoSQL databases like MongoDB and other NoSQL database variants, including key-value, graph, and wide-column stores.

Keywords: NoSQL · BASE · RDBMS · ACID · MongoDB · YCSB · benchmark · transaction management · database consistency

1 Introduction

In today's digital era, data storage techniques have had to evolve with the exponential growth of data and its diversified nature. Conventional databases, which are highly structured and rely on the ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure data accuracy and reliability, often struggle when dealing with voluminous and varied data. As a result, NoSQL databases have emerged, which prioritize flexibility, scalability, and speed. NoSQL, which stands for "Not Only Structure Query Language," offers an alternative to the rigid structures of traditional relational databases.

There are four main categories in which NoSQL databases are classified: Graph Stores, Document Stores, Key-Value Stores, and Wide Column Stores. Graph Stores utilize graph structures, such as nodes and edges, to store data. Document Stores use BSON

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14816, pp. 194–205, 2024.
https://doi.org/10.1007/978-3-031-65223-3_13

(Binary JSON) documents to store data. Key-value stores use unique key-value pairs for data storage, while Wide Column Stores utilize columns as their primary data storage. These categories significantly improve database design and offer greater flexibility and scalability than traditional relational databases.

The evolution of data storage and retrieval methodologies has undergone a significant transition from traditional relational databases to NoSQL systems. This shift has been primarily influenced by the ability of NoSQL systems to accommodate the vast and diverse nature of Web 2.0 data [1]. However, these systems deviate from the relational model's foundational consistency measures [2]. While NoSQL databases promise advantages like scalability and flexibility, they introduce challenges in ensuring data consistency, especially in commercial cloud storage [1]. Vogels laid the foundation by discussing the concept of eventual consistency, wherein updates might not be instantaneous but are guaranteed over time [3]. This idea was further extended by researchers who explored flexible update mechanisms for weakly consistent replication, creating a bridge between strict consistency and high availability [4]. Parallel efforts in this arena have seen González-Aparicio et al. designing transaction processes that focus on user applications [5] and Lotfy et al. suggesting a middle layer to support ACID properties traditionally absent in NoSQL systems [6]. Taking a broader perspective, systems like "Megastore" [7] and "CloudTPS" [8] have emerged to offer scalable, consistent storage solutions in distributed environments. The narrative was further enriched by solutions like "Granola," aiming at low overhead in transaction coordination [9], and "ElasTraS," envisioning elasticity in cloud transaction storage [10]. Amidst these developments, a focus on specific tools and their transaction capabilities has flourished, such as exploring MongoDB's scalable transactions [11]. Innovative solutions like policy-based access control for consistency [12] and multilevel transaction models for distributed systems [13] have showcased the depth and breadth of research in reconciling scalability with data consistency.

Efforts to improve consistency in NoSQL databases have been a focus of recent research, given these databases often trade off strong consistency for scalability [14, 15]. Various locking mechanisms have been studied to maintain data consistency in NoSQL systems [16–18]. Alternative consistency models, like Causal Consistency, have also been explored to balance performance and consistency [16, 19]. Tools like the Yahoo Cloud Serving Benchmark (YCSB) are the standard for evaluating database performance [20]. Consequently, enhancing consistency in scalable NoSQL databases remains an active area of research.

This paper explores the challenges of NoSQL databases, focusing on the trade-offs between scalability and consistency. While NoSQL databases have revolutionized how data is stored and managed, they often operate on the BASE properties, which can compromise the strong consistency offered by traditional RDBMSs and their ACID properties. To bridge this gap, we introduce a server-side model segmented into four pivotal stages to enhance database consistency. Our model's efficacy is empirically examined using the YCSB benchmark, targeting workloads that underscore update operations. After implementing our proposed algorithm, our findings show significant improvements in average throughput and read and update timings. Our research demonstrates a promising advancement in achieving enhanced consistency for NoSQL databases.

We are optimistic about the potential applicability of our model across various NoSQL platforms, including key-value, graph, and wide-column stores, thereby pushing the boundaries of what these databases can offer regarding data reliability.

2 Methods

Based on the insights we gathered from our extensive literature review, we created a customized framework that effectively addresses the identified challenges and gaps. We carefully integrated its components to ensure a cohesive structure and designed it to blend the selected tools' functionalities seamlessly. The architecture we developed is scalable and adaptable to potential future modifications.

2.1 Tools Employed

MongoDB Database. MongoDB is a well-known NoSQL database that is famous for its document-oriented structure. It stores data in flexible, JSON-like documents. This format enables fields to vary from document to document and permits data structure changes over time. As a result, MongoDB is highly adaptable and efficient for handling large volumes and diverse data types.

One of MongoDB's major strengths is its scalability, which is achieved through features like sharding and replication. Sharding distributes data across multiple machines, while replication ensures data availability and reliability. MongoDB's performance is further enhanced by indexing, which allows for faster data retrieval. Additionally, MongoDB has a rich query language that supports complex searches and aggregations.

MongoDB plays a crucial role in storing data and managing user transactions with robust security measures in our framework. The database operates on the principles of BASE (Basically Available, Soft state, Eventual consistency), which prioritizes availability and partition tolerance. Yet, it still allows configurable consistency levels to meet various application needs.

To optimize our use of MongoDB, we have meticulously designed the schema and managed transactions to enhance consistency without sacrificing performance. We fine-tuned the consistency settings and leveraged MongoDB's built-in features like write concerns, read preferences and atomic operations. By doing so, we ensure that our system performs efficiently, maintains high data integrity, and meets our applications' consistency requirements.

Performance Benchmarking with YCSB. We utilized the Yahoo Cloud Serving Benchmark (YCSB) to evaluate the efficacy and robustness of our proposed solution. YCSB is a well-known benchmarking tool in the industry that provides a comprehensive suite of tests to measure performance across different real-world scenarios. Our assessment with YCSB covered multiple dimensions of performance analysis, including latency, throughput, and scalability across diverse data workloads. The data generated from these benchmarks helped identify and quantify the efficiency enhancements our framework offers during different phases of data processing.

The results obtained from these benchmarks demonstrated significant improvements in processing speeds and response times, as well as our system's ability to scale effectively under increased loads. This rigorous benchmarking process has enabled us to fine-tune our solution, ensure it meets industry standards, and effectively address practical user demands.

Programming Environment. We chose Java as the programming language for our proposed model. Java is an object-oriented language widely used and appreciated for its versatility, platform independence, and robust library ecosystem. We chose Java for its cross-platform capabilities, which are particularly useful in ensuring consistent performance across different operating systems - a crucial factor for deploying enterprise-level applications. Additionally, Java is compatible with various tools and frameworks, such as MongoDB for NoSQL database interactions and YCSB (Yahoo! Cloud Serving Benchmark) for performance benchmarking. This integration capability allows our model to leverage existing technologies while maintaining scalability and high performance.

We adhered to recognized design patterns and industry best practices throughout the development process. This approach not only facilitated a more maintainable codebase but also enhanced our model's overall efficiency and reliability. By adopting these strategies, we aimed to create a solution that meets current technical demands and is adaptable to future advancements and requirements.

3 Results

We evaluated our proposed algorithm's performance on multi-document transactions in MongoDB using the YCSB benchmark standards. Specifically, we focused on assessing the impact of our algorithm on workloads A, B, and F, with the primary goal of evaluating read/write consistency and its effect on multi-document performance in MongoDB. Our algorithm was integrated into the YCSB framework for this purpose.

To accomplish our goal, we added a lock() class to the YCSB benchmark. This class temporarily locks a MongoDB document accessed by the initial client's transaction. Once the transaction is complete, it calls the commit() method. If the transaction is not finalized within a predetermined timeframe, the abort() method is invoked, and the transaction is rolled back. After this, the document is unlocked, allowing other clients to access it.

We conducted experiments on a dataset of 10,000 operations and records to assess the performance of a proposed algorithm across different client configurations. The configurations included one, five, ten, and fifteen clients. Each client operated independently, with varying levels of operations per millisecond. For instance, a single client executed 10,000 operations/ms, whereas fifteen clients performed 150,000 operations/ms.

We used a two-phase structure comprising loading and run phases to measure the performance. During the loading phase, data was loaded, and the run phase executed the data using the selected YCSB workloads. This methodology helped measure the throughput (ops/sec) and latency variations. We analyzed the consistency of the results in MongoDB after the YCSB workload executor processed both the data load and run.

We have provided graphical representations of our findings in accompanying figures that compare transaction operations across 1, 5, 10, and 15 clients, both pre/post

algorithm integration into the YCSB benchmark. Metrics were captured in time (milliseconds) and operations per second.

We evaluated the proposed algorithm using the YCSB benchmark, focusing on workloads A, B, and F. These benchmarks were selected to understand how the algorithm impacted throughput, average update latency, and average read latency.

3.1 Workload A (Update Heavy Workload)

This workload comprises a 50/50 mix of operations by default. Throughput: As shown in Fig. 1, throughput was significantly improved for 1 and 15 clients after applying our algorithm. However, due to possible data inconsistencies during ongoing operations, its effect was comparatively lesser on 5 and 10 clients.

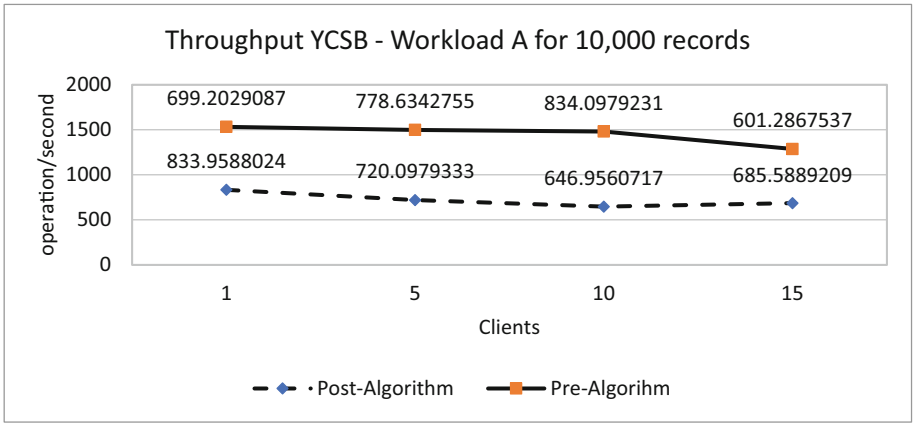


Fig. 1. Comparison of Throughput in YCSB Workload A for 10,000 Records Before and After Implementing the Proposed Algorithm.

Update Latency. As presented in Fig. 2, data across all clients showed strong consistency after applying the proposed algorithm.

Read Latency. Figure 3 displays the read latency results. Data in 5 and 10 clients showed high consistency, whereas 1 and 15 clients exhibited reduced consistency, which might be attributed to data conflicts when reading multiple operations concurrently. The proposed algorithm enhanced data consistency for Workload A, especially with concurrent transactions. However, the discrepancy in read transactions occasionally led to read failures during execution.

3.2 M Workload B (Read Mostly Workload)

This workload consists of a 95/5 read/write mix. Throughput: As evidenced in Fig. 4, the throughput was considerably higher for 1, 5, and 15 clients post-algorithm applications. However, the effect was least observed in the 10-client setup due to possible inconsistencies in write operations during execution.

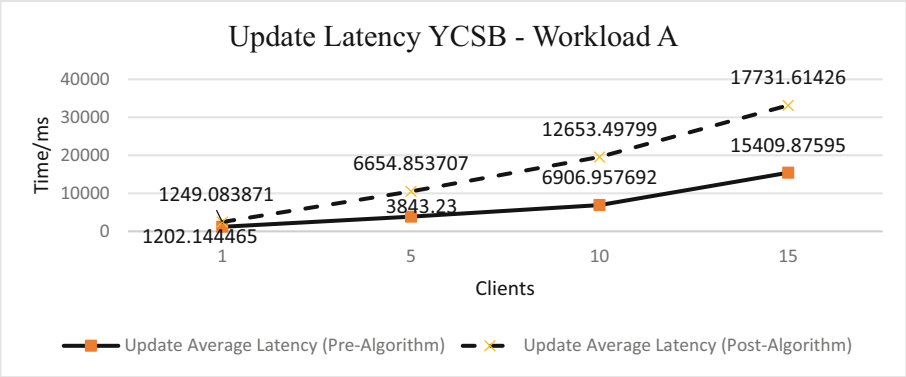


Fig. 2. Update Latency under Workload A - The graph illustrates the update latency measured at 3 ms, highlighting the system's efficiency in handling update-heavy operations within a NoSQL database environment.

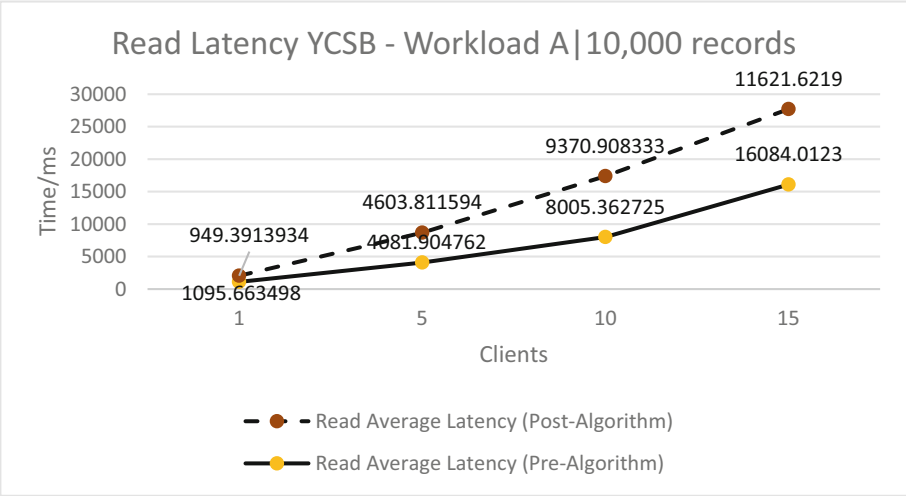


Fig. 3. Comparison of Read Latency for Workload A with 10,000 Records: Before and After Algorithm Implementation

Update and Read Latency. Both Figs. 5 and 6 suggest that our algorithm ensured strong data consistency across the board, with only the 10-client setup showing minor inconsistencies. This pattern is consistent with the algorithm's design, prioritizing strong workloads consistent with predominant read operations.

3.3 Workload F (Read, Modify, Write Workload)

Records are read and modified in this workload, and the changes are written back. Throughput: Fig. 7 highlights the algorithm's noticeable impact on 10 and 15 clients,

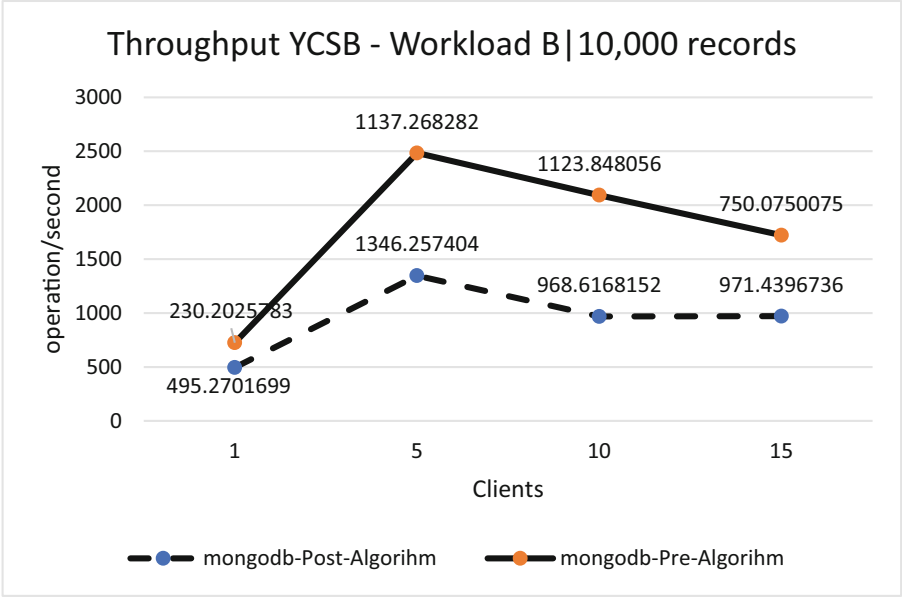


Fig. 4. Comparison of Throughput in YCSB Workload B for 10,000 Records Before and After Implementing the Proposed Algorithm. The graph illustrates the number of transactions per second for various client configurations (1, 5, 10, and 15 clients) both before and after the application of the algorithm. The aim is to highlight the algorithm's effectiveness in enhancing database throughput.

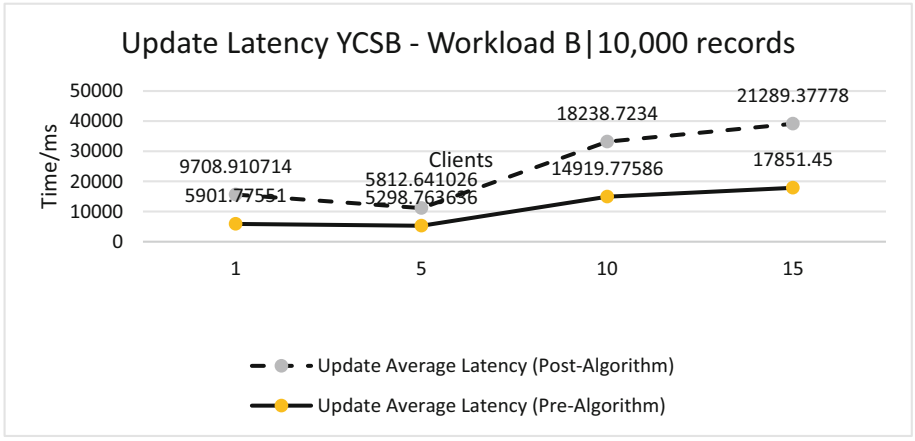


Fig. 5. Comparison of Update Latency in YCSB Workload B for 10,000 Records Before and After Implementing the Proposed Algorithm

with minimal effects on 1 and 5 clients. This could be due to inconsistencies during ongoing operations.

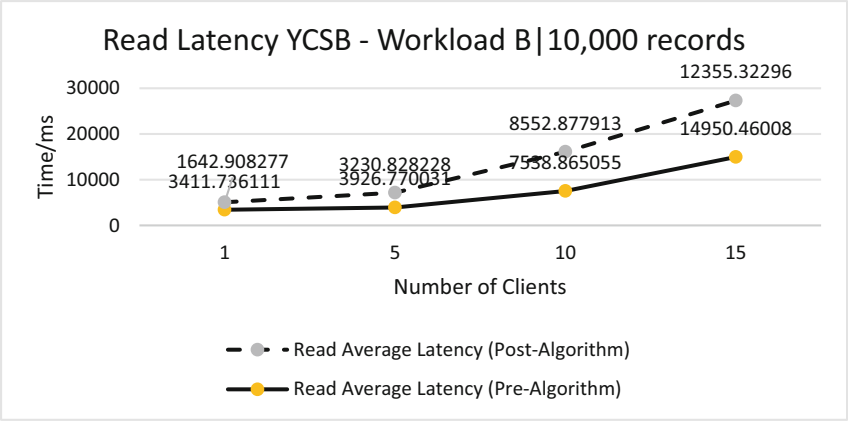


Fig. 6. Comparison of Read Latency for Workload B with 10,000 Records Before and After Implementing the Proposed Algorithm. This graph illustrates the variation in read latency among 1, 5, 10, and 15 clients, showcasing the impact of the algorithm on improving data consistency and transaction speed.

Update Latency. As depicted in Fig. 8, most client setups showed increased consistency in update latency post-algorithm application, except for the 5-client setup (Fig. 9).

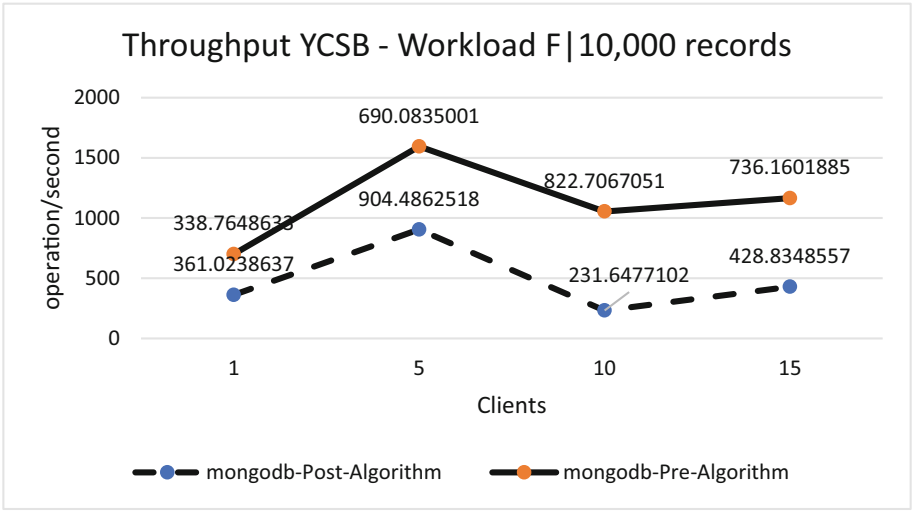


Fig. 7. Comparison of Throughput in YCSB Workload F with 10,000 Records, Before and After Implementing the Proposed Algorithm. The graph illustrates the increase in throughput across different client scenarios (1, 5, 10, 15 clients) after the algorithm's implementation, demonstrating its effectiveness in improving database performance.

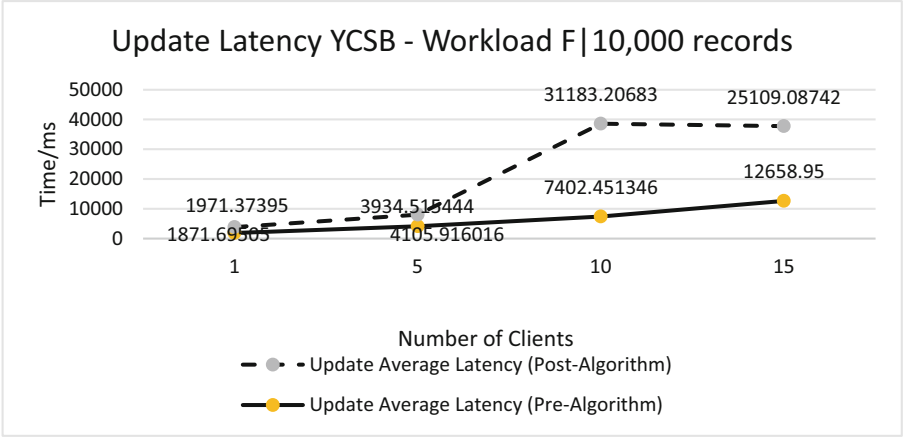


Fig. 8. Update Latency YCSB - Workload F for 10,000 Records (Pre-Algorithm): This graph shows baseline update latency across different client counts before implementing the proposed algorithm, serving to evaluate its effectiveness.

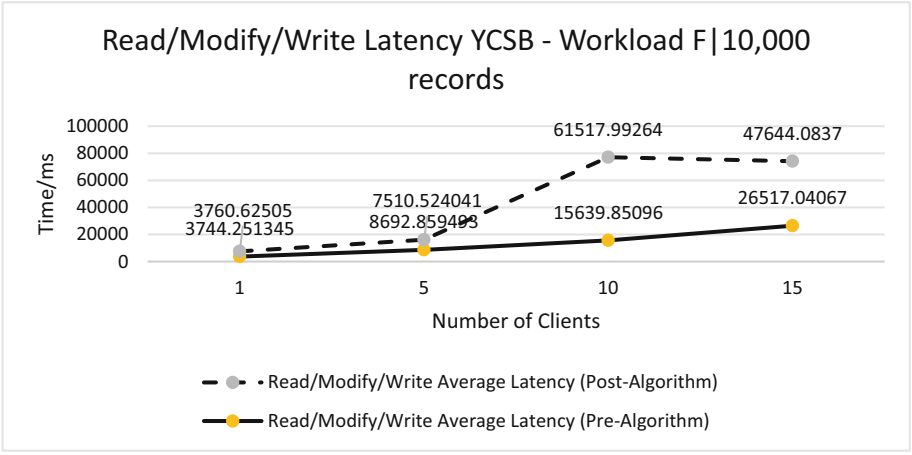


Fig. 9. Comparison of Read/Modify/Write Latency Before and After Algorithm Implementation for YCSB Workload F with 1000 Records.

Read Latency. In Fig. 10, the 10 and 15 client configurations demonstrated the algorithm's impact, with lesser effects on the 1 and 5 client configurations. For Workload F, our algorithm strengthened consistency in most tests, especially during update operations. However, read operations exhibited varying levels of consistency.

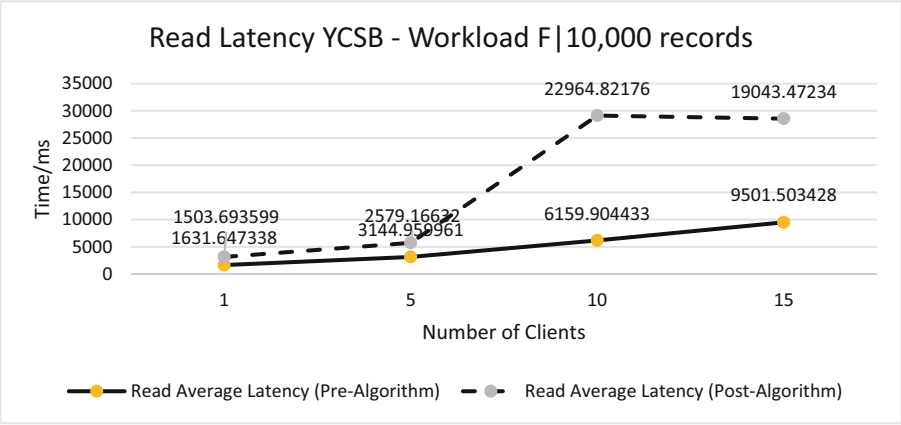


Fig. 10. Comparison of Read Latency for 10,000 Records in YCSB Workload F Before and After Algorithm Implementation.

Table 1. Table 2 Impact of the Proposed Algorithm on MongoDB Database Performance

Workloads	Throughput Latency AVG				Update Latency AVG				Read Latency AVG				Read/Modify/Write Latency AVG			
	Clients				Clients				Clients				Clients			
	1	5	10	15	1	5	10	15	1	5	10	15	1	5	10	15
Workload A	✔	✖	✖	✔	✔	✔	✔	✔	✖	✔	✔	✖	–	–	–	–
Workload B	✖	✔	✖	✔	✔	✔	✔	✔	✖	✖	✔	✖	–	–	–	–
Workload F	✔	✔	✖	✖	✔	✖	✔	✔	✖	✖	✔	✔	✔	✖	✔	✔

4 Discussion

This study has a clear and focused aim to critically assess the integration of our proposed algorithm within the YCSB (Yahoo! Cloud Serving Benchmark) framework, employing MongoDB as an exemplar of NoSQL databases. Our primary goal was to explore the algorithm's influence on data consistency amidst varying client loads. We employed Workloads A, B, and F from the YCSB to facilitate a detailed measurement and comparative analysis of several key performance metrics, including average latency, throughput, read latency, and update latency under diverse experimental setups.

The empirical results in Table 1 demonstrate that our algorithm significantly bolsters data consistency within MongoDB. This improvement is consistent across a broad spectrum of client operations, encompassing both reads and writes. Notably,

update operations exhibited substantial enhancements in consistency across most client configurations, validating the algorithm's effectiveness in achieving its intended goals.

However, the results reveal inconsistencies in specific client scenarios, potentially attributable to data inaccuracies during ongoing transactions. These observations call for a cautious interpretation of the data and underline the urgent need for additional empirical research, particularly in environments where data access and modification are concurrent. This research is crucial for data consistency in MongoDB and similar NoSQL databases.

5 Conclusion

In big data processing, NoSQL databases have become increasingly favored for their capacity to manage large datasets with notable performance and efficiency. However, maintaining data consistency during transaction processing is a pivotal challenge for these databases.

Our research team addressed this challenge by developing an algorithm designed to enhance data consistency during multi-transaction processes in MongoDB. The algorithm ensures that updates conducted within transactions are consistently reflected in subsequent transactions, bolstering overall data integrity.

To evaluate the effectiveness of our algorithm, we integrated it into the Yahoo! Cloud Serving Benchmark (YCSB) and conducted rigorous testing across various transactional scenarios. We assessed the algorithm's performance under diverse client configurations, employing 1, 5, 10, and 15 clients across workloads A, B, and F.

The results from our extensive testing indicate that the proposed algorithm substantially improves data consistency compared to baseline metrics, particularly in writing transactions (updates) across most client setups. Although some variability was observed in consistency levels for read transactions across different clients, these variations were considered acceptable since no modifications were made to the data during processing.

In summary, our study offers crucial insights into the efficacy of our algorithm as a strategic tool to address consistency challenges in NoSQL databases. By enhancing data consistency, our algorithm improves the performance and efficiency of NoSQL databases and sets the stage for further advancements in big data processing.

Conflict of Interest Statement. The authors confirm no conflicts of interest associated with this publication.

References

1. Wada, H., Fekete, A., Zhao, L., Lee, K., Liu, A.: Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective"
2. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970). https://doi.org/10.1145/362384.362685
3. Vogels, W.: Eventually consistent. Commun. ACM **52**(1), 40–44 (2009). https://doi.org/10.1145/1435417.1435432

4. Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J.: Flexible update propagation for weakly consistent replication. In: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles - SOSP 1997, pp. 288–301. ACM Press, Saint Malo, France (1997). https://doi.org/10.1145/268998.266711
5. González-Aparicio, M.T., Ogunyadeka, A., Younas, M., Tuya, J., Casado, R.: Transaction processing in consistency-aware user's applications deployed on NoSQL databases. Hum.-Centric Comput. Inform. Sci. **7**(1) (2017). https://doi.org/10.1186/s13673-017-0088-3
6. Lotfy, A.E., Saleh, A.I., El-Ghareeb, H.A., Ali, H.A.: A middle layer solution to support ACID properties for NoSQL databases. J. King Saud Univ. Comput. Inform. Sci. **28**(1), 133–145 (2016). https://doi.org/10.1016/j.jksuci.2015.05.003
7. Baker, J., Bond, C., Corbett, J.C., Furman, J., Khorlin, A., Larson, J.: Megastore: Providing Scalable, Highly Available Storage for Interactive Services
8. Wei, Z., Pierre, G., Chi, C.-H.: CloudTPS: scalable transactions for web applications in the cloud. IEEE Trans. Serv. Comput. **5**(4), 525–539 (2012). https://doi.org/10.1109/TSC.2011.18
9. Cowling, J., Liskov, B.: Granola: Low-Overhead Distributed Transaction Coordination
10. Das, S., Agrawal, D., Abbadi, A.E.: ElasTraS: An Elastic Transactional Data Store in the Cloud
11. Chaple, L., Ahirrao, D.S.: Scalable transactions using mongodb cloud data store (2005)
12. Wobber, T., Rodeheffer, T.L., Terry, D.B.: Policy-based access control for weakly consistent replication. In: Proceedings of the 5th European Conference on Computer Systems, pp. 293–306. ACM, Paris France (2010). https://doi.org/10.1145/1755913.1755943
13. Tripathi, A.: A transaction model with multilevel consistency for shared data in distributed groupware systems. In: 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), pp. 236–245. IEEE, Pittsburgh, PA, USA (2016). https://doi.org/10.1109/CIC.2016.041
14. Ramzan, S., Bajwa, I.S., Kazmi, R., Amna: Challenges in NoSQL-based distributed data storage: a systematic literature review. Electronics **8**(5), 488 (2019). https://doi.org/10.3390/electronics8050488
15. Khan, W., Kumar, T., Zhang, C., Raj, K., Roy, A.M., Luo, B.: SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review. Big Data Cogn. Comput. **7**(2), 97 (2023). https://doi.org/10.3390/bdcc7020097
16. Alzaidi, M., Vagner, A.: Benchmarking Redis and HBase NoSQL databases using yahoo cloud service benchmarking tool. Ann. Math. Informaticae **56**, 1–9 (2022)
17. Vyas, K., Jat, P.: Study of consistency and performance trade-off in cassandra. In: Computer Science & Technology Trends, Academy and Industry Research Collaboration Center (AIRCC), pp. 61–77 (2022). https://doi.org/10.5121/csit.2022.121907
18. Tian, J., Yang, Q.: Horae: causal consistency model based on hot data governance. J. Supercomput. **78**(3), 4574–4599 (2022). https://doi.org/10.1007/s11227-021-04030-w
19. Ahmed, J., Karpenko, A., Tarasyuk, O., Gorbenko, A., Sheikh-Akbari, A.: Consistency issue and related trade-offs in distributed replicated systems and databases: a review. Radioelectron. Comput. Syst. (2), 171–179 (2023). https://doi.org/10.32620/reks.2023.2.14
20. Benchmarking cloud serving systems with YCSB | Proceedings of the 1st ACM symposium on Cloud computing. https://dl.acm.org/doi/abs/, https://doi.org/10.1145/1807128.1807152. Accessed 25 Aug 2023