Educational Administration: Theory and Practice
2024, 30(5), 6378 - 6387
ISSN: 2148-2403
https://kuey.net/

Research Article

MongoDB Oplog Archiving: Strategies And Experimental Analysis

Gayatri S. Kapadia[1*], Dr. Sonal Kanungo[2], Jalpa P. Pandya[3], Dr. Rustom D. Morena[4]

[1*]Ph.D. Scholar, Department of Computer Science, Veer Narmad South Gujarat University (VNSGU), Surat, Gujarat, India.
Email: gayatri.kapadia@scet.ac.in
[2]Associate Professor, Department of Computer Science, DPG Degree College, MDU University, Gurugram.
Email: sonalkanungo@gmail.com
[3]Ph.D. Scholar, Department of Computer Science, Veer Narmad South Gujarat University (VNSGU), Surat, Gujarat, India.
Email: jppandya.2007@gmail.com
[4]Professor, Department of Computer Science, Veer Narmad South Gujarat University (VNSGU), Surat, Gujarat, India.
Email: rdmorena@vnsgu.ac.in

Citation: Gayatri S. Kapadia et.al , (2024 MongoDB Oplog Archiving: Strategies And Experimental Analysis, *Educational Administration: Theory and Practice, 30(5), 6378 - 6387*
Doi: 1 10.53555/kuey.v30i5.3947

ARTICLE INFO

ABSTRACT

MongoDB is a widely used NoSQL database system. Archiving within MongoDB involves placing historical or outdated data in a distinct location or structure. This is usually done to release resources within the active database, leading to an enhancement in overall performance. The purpose of this approach is to enable organizations to efficiently handle substantial volumes of data and uphold a system that remains responsive for ongoing operations. MongoDB, known for being a NoSQL database, offers multiple methods for the archival of data. MongoDB depends on the Oplog (operation log) to record real-time database changes. This essential component plays a vital role in enabling features like replication and sharding, which are crucial for maintaining data consistency and ensuring high availability. Nevertheless, handling and archiving Oplog data can present notable difficulties, especially in extensive MongoDB deployments. This research delves into different Oplog archiving approaches and offers an empirical analysis of their performance and efficiency.

Keywords: MongoDB, NoSQL, Oplog, Strategies, Experimental Analysis, Performance, Efficiency.

1. Introduction

The Oplog archiving in MongoDB is a process that revolves around handling and storing the Oplog, which stands for "operations log." [1] The Oplog is essentially a specialized, capped collection that keeps a record of all write operations responsible for modifying data within a MongoDB cluster. This Oplog plays a vital role in maintaining data consistency and enabling replication within a MongoDB replica set. The Oplog is a continuously updated log that tracks all changes made to the data within a MongoDB replica set. These changes are stored as BSON (Binary JSON) documents. Each member of the replica set maintains its copy of the Oplog. [3][8]

The Oplog serves the purpose of mirroring data alterations from the primary node to the secondary nodes within a replica set. This functionality ensures that the secondary nodes stay synchronized and uphold data consistency with the primary node.[2][4][5][8]

The Oplog is structured as a capped collection, which implies that it has a predefined size limit. When the Oplog reaches this limit, it automatically deletes older entries to free up space for storing new entries. The duration for which the Oplog retains data relies on the unique requirements of your use case and the pace at which data changes occur. It's crucial to guarantee that the Oplog retains an appropriate amount of data to facilitate your backup and disaster recovery strategies effectively. The retention period of the Oplog depends on your specific use case and the rate of data changes. It's essential to ensure that the Oplog retains a sufficient amount of data to support your backup and disaster recovery strategies.

MongoDB offers tools like "mongodump" and "mongorestore" that are capable of backing up and restoring Oplog data. You can create automation scripts to schedule and oversee Oplog archiving tasks according to your requirements. As the Oplog contains sensitive information regarding changes to your database, it's crucial to

ensure its secure storage and restrict access to authorized personnel only. To maintain the Oplog's adequacy for your replication and backup needs, it's essential to routinely monitor its size and the rate at which it grows. [6][14]

2. Oplog Archiving Strategies

The tactics employed for MongoDB archiving encompass a range of methods and techniques utilized to handle historical or infrequently accessed data within a MongoDB database. These strategies are designed to enhance database performance, storage efficiency, and overall data management. They achieve this by organizing, storing, or managing data in a way that aligns with the application's specific requirements and patterns of usage. The ultimate objective is to find a harmonious balance between preserving essential historical information and guaranteeing the ongoing responsiveness of the active dataset. These are some archiving strategies:

2.1 Rolling Window Archiving

Rolling window archiving in MongoDB is a strategy that involves the management of Oplog data. It focuses on retaining a defined timeframe of recent changes while automatically removing older entries to ensure efficient use of storage resources. Rolling window archiving aims to preserve a defined "window" of recent changes within the Oplog. [6] As new write operations occur, they are consistently added to either the beginning or end of the Oplog, while older operations are systematically removed to ensure the Oplog remains within its size limit. When you implement a rolling window archiving strategy, you set a retention period or timeframe for how long you intend to preserve the Oplog data. For instance, you might opt to retain the record of write operations from the past 24 hours as part of your strategy.

When new operations are added to the Oplog, the oldest operations that exceed the specified retention period are automatically purged. This automated removal process ensures that the Oplog doesn't continue to expand indefinitely and stays within its designated size limit. [10] Rolling window archiving optimizes storage space usage by retaining data that are pertinent for a set timeframe. It streamlines Oplog management by eliminating the need for manual deletion of old records. Rolling window archiving is particularly useful when you have limited storage resources, and you want to ensure that your Oplog remains efficient and doesn't consume excessive disk space.

It aligns with scenarios where historical data beyond a certain point is less critical, and recent changes are the primary focus. To put a rolling window archiving strategy into action, it's essential to configure the Oplog size limit and retention period based on your particular needs. Modifying these parameters demands meticulous planning to guarantee that you retain a sufficient amount of historical data to support backup and recovery objectives effectively. [14] Regularly observe both the size and content of the Oplog to confirm that the rolling window strategy is efficiently handling the data. Make certain that the chosen retention period aligns with your data recovery goals to maintain a robust archiving approach.

In conclusion, rolling window archiving in MongoDB is a strategy designed to keep the Oplog well-managed and efficient. It achieves this by automatically purging older write operations while preserving a defined timeframe of recent changes. This approach strikes a balance between retaining valuable data and optimizing storage usage, making it a suitable choice for various MongoDB setups, particularly those with limited storage resources.

2.2 Continuous Archiving

Continuous archiving in MongoDB is a data management strategy that revolves around the continuous and real-time collection and storage of the Oplog.[7] This approach holds significant importance in guaranteeing comprehensive data recovery, replication, and auditing capabilities within MongoDB. Let's delve into a more detailed explanation of continuous archiving:

Continuous archiving is a method used to persistently capture and store every single operation or change made to the database in real time. This is achieved by actively monitoring and recording all write operations, such as inserts, updates, and deletes, as they occur within the MongoDB database.

The Oplog in MongoDB is a specialized capped collection that keeps a record of all write operations, encompassing inserts, updates, and deletes. It captures these changes in real time, making it a valuable and up-to-the-minute record of all data modifications in the database. Continuous archiving strives to maintain an exhaustive and current history of Oplog data. Its primary goal is to guarantee that no write operation goes unrecorded, ensuring a seamless and unbroken record of changes. This continuous record is essential for both data recovery and replication purposes.[8] Continuous archiving entails the frequent copying of Oplog data to an external storage system or a dedicated database. This copying process occurs in real-time or near-real-time, ensuring that the archived data always remains up-to-date and reflective of the most recent changes.

The archived Oplog data plays a pivotal role in conducting backups and achieving point-in-time recovery. It enables you to utilize the archived Oplog to restore your database to a precise moment in time, facilitating granular and precise data recovery processes.[16] When incorporating a new secondary node into a replica set, the archived Oplog can be employed to initialize the new node with all the historical data changes. This procedure ensures that the new node is brought up-to-date and maintains consistency with the primary node.

Organizations commonly store the archived Oplog data in an external storage system, such as a file system, cloud storage, or a dedicated database. The choice of storage solution is typically made based on how well it aligns with an organization's specific infrastructure and scalability requirements. [11] Retention policies are established guidelines that govern the management of the size and lifecycle of archived Oplog data. These policies define the duration for which archived Oplog data is retained before older records are systematically purged or removed.
It's essential to restrict access to the archived Oplog to authorized personnel only, as this data contains sensitive information about data changes. Regularly monitoring the archiving process is crucial to ensure it functions correctly. Additionally, testing the recovery process using the archived Oplog is essential to validate its effectiveness and readiness for use in case of data restoration or replication needs. [14]

2.3 Hybrid Archiving

Atlas's Online Archive feature provides the capability to establish archiving rules that relocate infrequently accessed data from your Atlas cluster to a MongoDB-managed object-store. Furthermore, it enables you to seamlessly query both your Atlas and Online Archive data in a unified manner, simplifying the process without requiring you to consider the specific tier where the data is stored. We are improving this feature by introducing two new capabilities: data expiration and scheduled archiving. Online Archive simplifies the process of transitioning data from a live database to an object store [16]. However, what if you wish to establish a secondary threshold for permanently deleting data from the archive? There might be reasons such as cost considerations or compliance requirements necessitating data expiration and scheduled deletion.
There was no straightforward option to remove data from the archive without deleting the entire archive, which wasn't ideal for most scenarios. However, with our new data expiration feature, you have the flexibility to specify how long data should be retained in the online archive before it's automatically deleted. You can set an expiration period ranging from as short as seven days to as long as 9,125 days. This archive expiration time can be configured either through the Atlas UI or the Admin API. Furthermore, you can edit expiration rules after they've been initially created, providing adaptability to your evolving needs. It's important to note that once data is deleted from the archive, there is no way to recover it. Therefore, it's crucial to define your rules carefully and thoughtfully to avoid any unintended data loss.[20]

3. Related Work

Cloud computing introduces challenges due to its dynamic resource allocation and varying workloads. To overcome these challenges, the authors put forth enhancements related to auto-sharding, a fundamental feature of MongoDB that supports horizontal scaling and the distribution of workloads. Liu Y et. al. had a primary goal of tackling the difficulties associated with effectively managing and expanding MongoDB databases within the context of cloud computing. In essence, the authors aimed to make MongoDB databases work better in cloud environments where resources are allocated dynamically and workloads fluctuate. They did this by proposing and implementing improvements in auto-sharding, which is crucial for horizontally scaling databases and distributing the load effectively. [1] Tauro et. al. contributed a better understanding of the strengths and weaknesses of different NoSQL databases, ultimately aiding in the optimal choice of database technology for specific application requirements. [4] Tabet et. al. offered a comprehensive overview of the different techniques and approaches used for data replication in cloud environments. The authors delve into the importance of data replication in cloud systems, highlighting its crucial role in guaranteeing data availability, fault tolerance, and optimizing system performance. [8] One essential component of database systems is the assurance of data availability, fault tolerance, and scalability. MongoDB's method of handling data replication, commonly known as "replica sets," plays a vital role in upholding data consistency and dependability. Gu, Y. et. al. were primarily focused on investigating and providing insights into how MongoDB handles data replication. Discussing MongoDB's mechanisms for achieving data consistency among replica set members, including writing concerns and acknowledgment strategies. [3]
Yimeng Liu et. al. suggested and put into action enhancements associated with auto-sharding, a fundamental feature of MongoDB that enables the horizontal expansion of databases and equitable distribution of data across numerous servers. [10] Mansouri, N. et. al. introduced an innovative method for data replication in data grid systems, with an emphasis on enhancing resource allocation while taking economic considerations into account. [5] Li. T et. al contributed to the evolving field of IoT data management by offering a NoSQL-based storage solution tailored to the challenges posed by massive and diverse IoT data streams. It provides insights into how such data can be efficiently stored and processed, potentially with a focus on green computing practices. [2] Replication is a fundamental concept addressed within the framework of MongoDB, a widely used NoSQL database system. In the context of MongoDB, replication involves the practice of keeping multiple copies of data distributed across various servers or nodes, primarily to ensure high availability, fault tolerance, and data redundancy. [6] As per Mrs. Anuradha et. al., the Study of Normalization and Embedding in MongoDB" is anticipated to offer a valuable examination of data modeling strategies within MongoDB. It provides recommendations on the appropriate scenarios for employing normalization or embedding, contingent on distinct application needs, and delves into the ramifications of these decisions on database performance and effectiveness. [11] "Cloud-Based Databases - A Changing Trend" is expected to be a valuable

resource for gaining insights into the transformative influence of cloud computing on the database landscape. It delves into the reasons driving the transition to cloud-based databases, the advantages, hurdles, and factors to consider, while also offering perspectives on future advancements in this domain – suggested by Anuradha et. al. [13] Kristina Chodorow's work, focusing on scalability and performance optimization in MongoDB, is likely to contain valuable insights and considerations regarding data archiving within a MongoDB environment. Data archiving is a vital facet of data management, particularly in large-scale applications. We can gain valuable guidance from the book on how to effectively utilize MongoDB for archiving historical data while upholding database performance and ensuring continued data accessibility. [14] L. Song et. al. presented a methodology for addressing performance issues encountered in practical software development. By leveraging statistical techniques, the authors provide a framework for systematically identifying and diagnosing performance problems. This research is likely to be of significant interest to software developers and engineers dealing with real-world performance challenges in their projects. [15]

Khaoula Tabet, et. al. introduced a forward-thinking approach to enhancing data replication in MongoDB, with a specific focus on mitigating existing challenges and elevating database performance and dependability. [20] Mansouri et. al. a notable contribution to the field of data grid management by introducing a data replication strategy that takes economic factors into account. By carefully assessing the costs and benefits of data replication, this approach offers a valuable framework for organizations looking to optimize their data grid systems. [7]

A sharded cluster in MongoDB is a distributed database architecture that's specifically designed to meet the scalability and performance requirements when dealing with large volumes of data. A sharded cluster provides horizontal scalability and high availability for MongoDB databases, making it suitable for handling large and growing datasets. Distributing data across multiple shard servers ensures that the system can accommodate increased data volumes and high query loads while maintaining performance. This architecture is particularly valuable for applications that require both scalability and flexibility in managing data. [9] Fay C et. al. introduced a pioneering method for managing structured data at a large scale, with a strong focus on scalability, fault tolerance, and simplicity. The concepts and principles underlying BigTable have had a profound impact on the design of numerous distributed storage systems and continue to play a significant role in how organizations handle and process vast amounts of data in contemporary computing environments. [12]

Through the use of profiling and visualization methods, Alexandra et. al. offered valuable insights into the intricacies of WiredTiger's performance. This, in turn, contributes to the development of more efficient and responsive database systems. This research holds significance not only for individuals working with WiredTiger but also for the wider community interested in the analysis and optimization of performance in intricate software systems. [16] As per Sonal et. al., there exists a fundamental link between concurrency control and data replication in distributed database systems. Effective concurrency control plays a pivotal role in handling simultaneous access to replicated data, guaranteeing data consistency, and ensuring availability across multiple nodes. Grasping the interaction between concurrency control and replication is vital for the design of resilient and high-performance distributed database systems.[17] The same authors distinguished that there's a distinct correlation between MVCC and data replication in distributed database environments. MVCC methods are instrumental in overseeing concurrent access to replicated data, upholding data consistency, and implementing conflict resolution mechanisms. [18] [26] As per Vaishali et. al. although the primary emphasis of the paper is on security mechanisms within NoSQL databases, there exists an inherent link between security and data archiving in the context of ensuring secure data management across its entire lifecycle. When considering data archiving, it is imperative to adopt a security-oriented approach to safeguard historical data from unauthorized access and to follow relevant regulations and compliance requirements. [19] Henrik Ingo et. al. summarized that The Distributed Systems Infrastructure (DSI) is a framework developed by MongoDB to conduct fully automated system performance tests within our CI environment. The automation of the deployment of authentic multi-node clusters, the execution of tests, the optimization of the system for consistent outcomes, and the subsequent collection and analysis of results, is a complex challenge that requires three attempts and a span of six years to perfect. The DSI project, which has been made available to the public, is the culmination of these endeavors. [21] William Schultz et. al. delved into the concept of tunable consistency within the realm of MongoDB, a prominent NoSQL database system. Their investigation has centered on comprehending the effects of MongoDB's tunable consistency levels, which encompass the default "strong" consistency and "eventual" consistency, on the performance, availability, and data integrity of database operations. [22] Misha Tyulenev et. al. presented our research on the implementation of cluster-wide logical clocks and causal consistency in MongoDB, a widely-used NoSQL database system. Their investigation aimed to enhance the capabilities of MongoDB in terms of providing stronger consistency guarantees and improving data synchronization across distributed clusters. [23] Sangmin Lee et. al. introduced an inventive and all-encompassing solution to the complexities of overseeing data shards in geo-distributed applications. The Shard Manager framework provides a versatile and adjustable method for shard positioning, replication, and load distribution, ultimately enhancing the efficiency and dependability of geo-distributed systems. Their research brings forth valuable insights and tools for the expanding domain of distributed systems and database management. [24] David Mealha et. al. "Data Replication on the Cloud/Edge" provides valuable insights and solutions for addressing the intricate challenges related to data replication in distributed computing environments. Their work empowers both practitioners and researchers with a more profound comprehension

of how to conceive and execute efficient data replication systems. [25] These systems are crucial for maintaining data consistency, availability, and performance within the ever-evolving realms of cloud and edge computing.

4. Proposed Archiving Strategy

The main objective of any archiving strategy is to determine what data should be archived, how much older data should be archived, when to be archived, and where to place this archive whenever data recovery is needed. We sort out all these issues in the following section by proposing a new strategy of archiving in MongoDB to acquire better performance while maximizing the cloud provider's profit.
First of all, we need to decide when to start the archiving process. We need to calculate the data arrival time and the actual stored time and find the difference between retention time (RT) with threshold time (ThT). If RT is greater than ThT then compare the OplogSize with the maximum allowable OplogSize.
Then node is selected for archiving and further, it is stored.

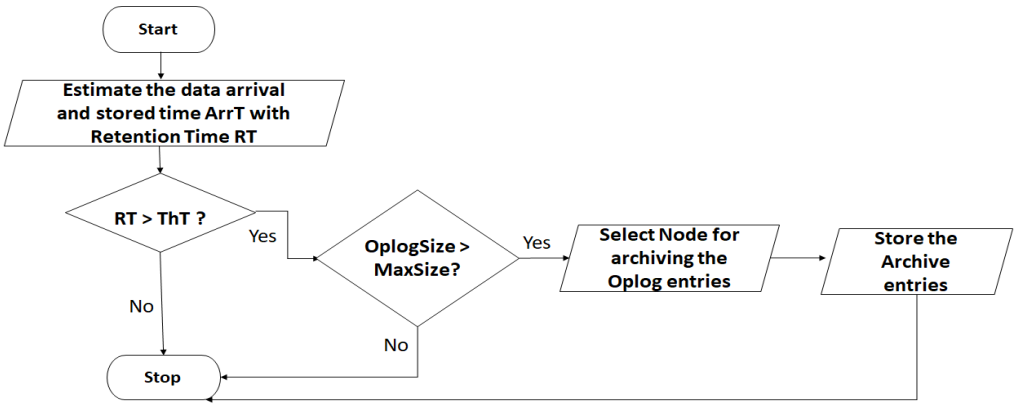

Fig. 1 {Archiving Decision Process}

As shown in the above figure, the data arrival and stored time is estimated as ArrT along with retention time RT. This RT will be compared against the Threshold time ThT. If the ThT is greater than RT then check whether the Oplog size exceeds the maximum allowable Oplog size, if both ThT and OplogSize conditions are fulfilled then node selection is done for archiving Oplog entries. This is how the archiving is performed. If the Retention time is less than the threshold time then no need to perform archiving. Similarly, if the Oplog size is less than the maximum size of Oplog then no need to perform archiving.

5. Experimental Setup

To perform experiments, we have taken various databases and collections of MongoDB Atlas samples.
The following databases are used while performing experiments in our MongoDB Atlas.

Table 1. The list of databases (sample data sets for experiments) in MongoDB Atlas on Cluster0 of the authors account as follows:

Sr. No.	Database Name	Total Collections	Storage size
1	sample_airbnb	1	52.02MB
2	sample_analytics	3	9.39 MB
3	sample_geospatial	1	804 KB
4	sample_guides	1	20 KB
5	sample_mflix	6	18.55 MB
6	sample_restaurants	2	5.7 MB
7	sample_supplies	1	920 KB
8	sample_training	7	41.82 MB
9	sample_weatherdata	1	2.36 MB

6. Experimental Analysis

The details of the experimental analysis of various Oplog archiving strategies are presented in this section.
Here,
Storage Overhead (in gigabytes) = Primary Data Size (in gigabytes) * Overhead Percentage
We are estimating the percentage of additional storage space you anticipate adding to your primary data size. This percentage can vary based on various factors, including considerations such as indexing, journaling, and other operational requirements. A typical range for this overhead falls between 20% and 50%, but it's important

to note that the specific percentage can vary considerably depending on your particular use case and configuration.
Replication Lag (in seconds) = Timestamp on Secondary Node - Timestamp on Primary Node
Data Recovery Time (in hours) = Backup Size (in gigabytes) / Restore Speed (in gigabytes per hour)

Table 2. Comparison of different Oplog Archiving Strategies concerning Storage Overhead, Replication Lag, and Data Recovery Time:

Oplog Archiving Strategies	**Storage Overhead**		**Replication Lag**		**Data Recovery Time**	
	Average Storage Overhead	**Maximum Storage Overhead**	**Average Replication Lag**	**Maximum Replication Lag**	**Average Data Recovery Time**	**Maximum Data Recovery Time**
Rolling Window Archiving	5% of the total database size	8% of the total database size	0.5 seconds	1.2 seconds	15.20 minutes	30.40 minutes
Continuous Archiving	15% of the total database size	20% of the total database size	2.5 seconds	4 seconds	5.20 minutes	10.50 minutes
Hybrid Archiving	8% of the total database size	12% of the total database size	1 second	2 seconds	10.24 minutes	20.50 minutes

We used three strategies in our experiments. If we compare the storage overhead parameter with all three strategies, rolling window archiving is the most suitable compared to the other two strategies. Whereas continuous archiving has more storage overhead than the other two strategies. Hybrid archiving has the highest storage overhead. In short, if our focus is only on storage then we can opt for either rolling window archiving or hybrid archiving. The replication lag of rolling window archiving is the lowest, whereas hybrid archiving and continuous archiving have respectively more replication lag. The recovery time of continuous archiving is the lowest and our main focus is on recovery time, especially for some crucial applications.

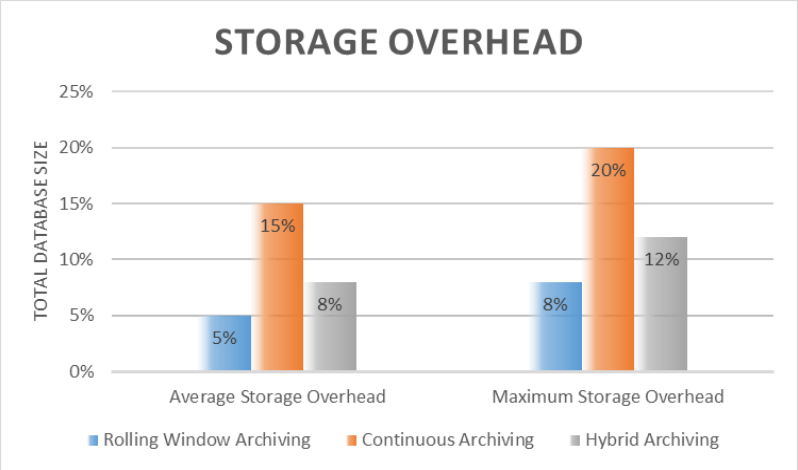

Fig. 2 {Chart of Storage Overhead in all three archiving strategies}

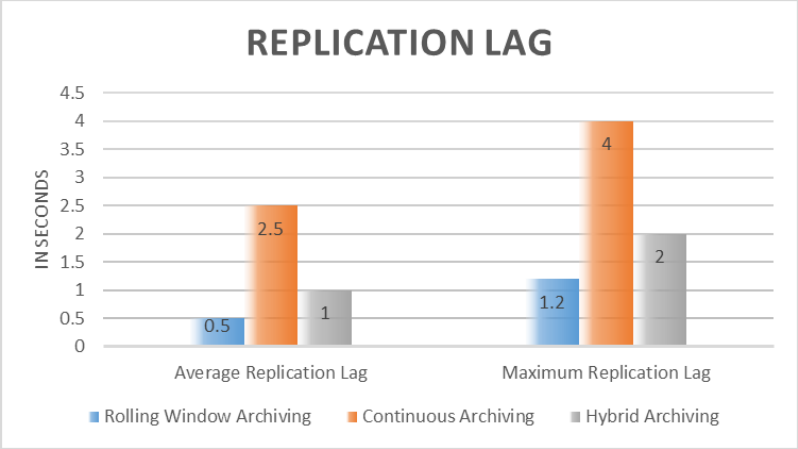

Fig. 3 {Chart of Replication Lag in all three archiving strategies}

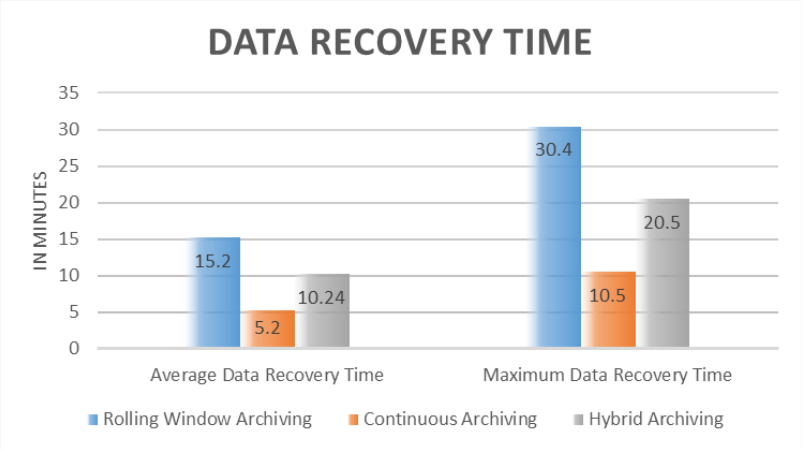

Fig. 4 {Chart of Data Recovery Time in all three archiving strategies}

The outcomes of our experimental analysis offer valuable perspectives on the advantages and limitations of each Oplog archiving approach. These insights aid MongoDB administrators in making well-informed choices regarding Oplog management tailored to their particular setups. Furthermore, our findings contribute to the wider conversation surrounding the enhancement of MongoDB deployments for large-scale applications, prioritizing both data integrity and operational effectiveness.

Let us understand this with an example - when we execute the find query for searching documents that are older than 10 years i.e., 365 days x 10 years = 3650 days against the sample_mflix.comments collection. we find that there is a total of 5.63MB documents in this collection, and after running our query to find all of the comments that are older than 10 years old, we find that 41079 documents would be archived using this rule.

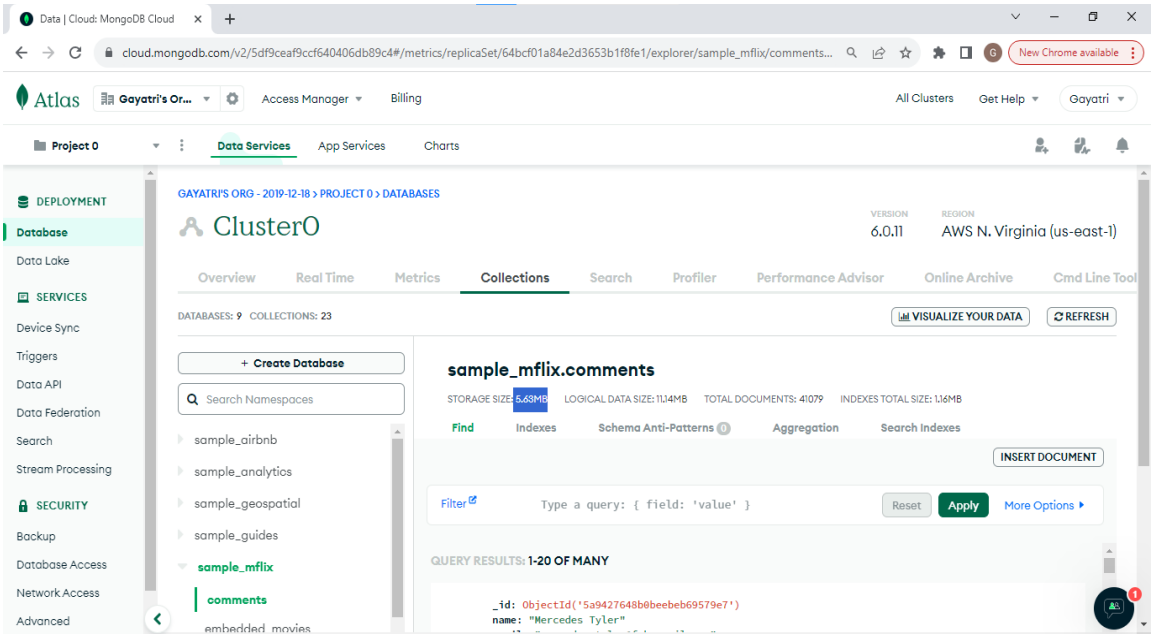

Fig. 5 {Screenshot of sample_mflix.comments collection for archiving}

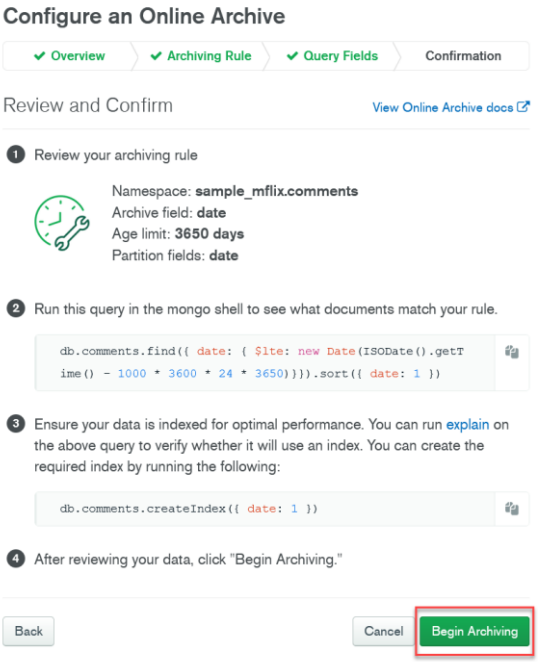

Fig. 6 {Screenshot of archiving rule for sample_mflix.comments collection}

After connecting to our dataset, we executed a query to ensure uninterrupted access and retrieval of all our data from a centralized endpoint, irrespective of its archival status. This approach of archiving outdated data can prove to be a valuable asset when managing extensive and rapidly expanding datasets, as it helps optimize time, financial resources, and development efforts while accommodating the increasing demands on your data infrastructure.

7. Conclusion

Our research provides valuable insights into MongoDB Oplog archiving strategies and their associated implications. It offers actionable guidance for database administrators and engineers aiming to improve the performance and reliability of their MongoDB deployments.

In this paper, we introduce a novel data archiving strategy tailored for MongoDB, a document-oriented NoSQL database engine. The primary objective of this strategy is to meet the performance demands of tenants while considering the provider's profitability. The archiving process is initiated only when the anticipated response time for a submitted query exceeds a predefined response time threshold specified in the Service Level

Agreement (SLA). Additionally, archiving is pursued if it proves profitable for the provider. Our approach relies on the key components: Data/Object Selection, Meta Data, Retention Policies, Backup and Redundancy, Indexing and Searchability, Audit Trails, and Legal & Ethical Considerations.
Future work can involve evaluating the proposed strategy in a real cloud platform to ensure it satisfies tenant requirements and reduces SLA violations. Consequently, this could lead to a decrease in penalties from the provider's perspective.

References:

1. Liu, Y., Wang, Y., & Jin, Y. (2012, July). Research on the improvement of MongoDB. Auto-Sharding in cloud environment. In Computer Science & Education (ICCSE), 2012 7th International Conference on (pp. 851-854). IEEE.
2. Li, T.; Liu, Y.; Tian, Y.; Shen, S.; Mao, W. A storage solution for massive IoT data based on NoSQL. In Proceedings of the 2012 IEEE International Conference on Green Computing and Communications (GreenCom), Besancon, France, 20–23 November 2012; pp. 50–57.
3. Gu, Y., Wang, X., Shen, S., Ji, S., & Wang, J. (2015, June). Analysis of data replication mechanism in NoSQL database MongoDB. In Consumer ElectronicsTaiwan (ICCE-TW), 2015 IEEE International Conference on (pp. 66-67). IEEE.
4. Tauro, C. J., Patil, B. R., & Prashanth, K. R. (2013). A comparative analysis of different NoSQL databases on the data model, query model, and replication model. In Proceedings of the International Conference on ERCICA.
5. Lima, I., Oliveira, M., Kieckbusch, D., Holanda, M., Walter, M. E. M., Araújo, A., Lifschitz, S. (2016, December). An evaluation of data replication for bioinformatics workflows on NoSQL systems. In Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on (pp. 896-901). IEEE.
6. Kristina Chodorow, MongoDB: The Definitive Guide O'Reilly Media.
7. Mansouri, N., & Asadi, A. (2014). Weighted data replication strategy for data grid considering economic approach. Int. J. Comput. Elect. Auto. Control Inf. Eng, 8, 1336-1345.
8. Tabet, K., Mokadem, R., Laouar, M. R., & Eom, S. (2017). Data Replication in Cloud Systems: A Survey. International Journal of Information Systems and Social Change (IJISSC), 8(3), 17-33.
9. Sharded Cluster retrieved from: https://www.mongodb.com/docs/manual/core/sharded-cluster-components/
10. Yimeng Liu, Yizhi Wang, Yi Jin, "Research on The Improvement of MongoDB Auto-Sharding in Cloud Environment", The 7th international conference on Computer Science & Education(ICCSE 2012), July 14-17, 2012.
11. Mrs. Anuradha Kanade, Dr. Arpita Gopal, Mr. Shantanu Kanade, "A Study of Normalization and Embedding in MongoDB", IEEE International Advanced Computing Conference, 2014.
12. Fay C, Jeffery D, Sanjay G, Wilson H, Deborah W, Michael B, Tushar C, Andrew F, Robert G. BigTable: a distributed storage system for structured data[J]. ACM Trans. Comput. Syst. 2008(26).
13. Anuradha Kanade, Dr. Arpita Gopal, Shantanu Kanade, 2013, "Cloud-Based Databases- A Changing Trend", International Journal of Management, IT and Engineering, Vol.3, Issue 7, pp. 273-282
14. Kristina Chodorow. "Scaling MongoDB". O'Reilly Media, February 2011.
15. L. Song and S. Lu. Statistical Debugging for Real-world Performance Problems. In Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA, pages 561–578, 2014.
16. Alexandra Fedorova, Craig Mustard, Ivan Beschastnikh, Julia Rubin, Augustine Wong, Svetozar Miucin, Louis Ye, "Performance Comprehension at WiredTiger" ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA. ACM ISBN 978-1-4503-5573-5/18/11.
17. Sonal Kanungo, Morena Rustom, "Analysis and Comparison of Concurrency Control Techniques", International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 3, March 2015.
18. Sonal Kanungo, Rustom D Morena, "Multi-version Concurrency Control for Object-Oriented Database Management System", International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 4, Issue 10, October 2017.
19. Vaishali J. Dindoliwala, Dr. Rustom D. Morena, "Survey on Security Mechanisms In NoSQL Databases", International Journal of Advanced Research in Computer Science, Volume 8, No. 5, May-June 2017.
20. Khaoula Tabet, Riad Mokadem, Mohamed Ridda Laouar, "Towards a New Data Replication Strategy in MongoDB Systems", ICCES '18, July 14–16, 2018, Kuala Lumpur, Malaysia. ACM ISBN 978-1-4503-6447-8/18/07
21. Henrik Ingo, David Daly, "Automated System Performance Testing at MongoDB", DBTest'20, June 19, 2020, Portland, OR, USA. ACM ISBN 978-1-4503-8001-0/20/06.
22. William Schultz, Tess Avitabile, Alyson Cabral, "Tunable Consistency in MongoDB", Proceedings of the VLDB Endowment, Vol. 12, No. 12 ISSN 2150-8097.

23. Misha Tyulenev, Andy Schwerin, Asya Kamsky, Randolph Tan, Alyson Cabral, Jack Mulrow, "Implementation of Cluster-wide Logical Clock and Causal Consistency in MongoDB", SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands, ACM ISBN 978-1-4503-5643-5/19/06.
24. Sangmin LeeZhenhua GuoOmer SunercanJun YingThawan KooburatSuryadeep BiswalJun ChenKun HuangYatpang CheungYiding ZhouKaushik VeeraraghavanBiren DamaniPol Mauri RuizVikas MehtaChunqiang Tang, “Shard Manager: A Generic Shard ManagementFramework for Geo-distributed Applications”, SOSP ’21, October 26–29, 2021, Virtual Event, Germany, ACM ISBN 978-1-4503-8709-5/21/10.
25. David Mealha, Nuno Preguiça, Maria Cecília Gomes, João LeitãoNOVA LINCS & DI/FCT/UNL Portugal, “Data Replication on the Cloud/Edge”, PaPoC ’19, March 25, 2019, Dresden, Germany, ACM ISBN 978-1-4503-6276-4/19/03.
26. Sonal Kanungo, Rustom D. Morena, “Concurrency versus consistency in NoSQL databases”, Journal of Autonomous Intelligence (2024) Volume 7 Issue 3.