Understanding the MongoDB Design Architecture and Dissecting the Performance

Amjad Alqahtani
Department of Cybersecurity- College of Science
University of Texas at San Antonio
San Antonio, Texas
amjad.alqahtani@my.utsa.edu

Nicholas Winkelmann
Department of Cybersecurity- College of Science
University of Texas at San Antonio
San Antonio, Texas
nicholas.winkelmann@my.utsa.edu

Caleb Alva
Department of Cybersecurity- College of Science
University of Texas at San Antonio
San Antonio, Texas
caleb.alva@my.utsa.edu

***Abstract*— This project describes the system architecture of MongoDB, a NoSQL (Not Only Structured Query Language). MongoDB is a document-based database management system known for its widespread adoption among NoSQL distributed databases. The project presents an evaluation and an experimental performance analysis of MongoDB's system. The evaluation was conducted in a Cloud Atlas environment using the Yahoo! Cloud Serving Benchmark (YCSB). The results indicate that MongoDB performs well with read operations but faces challenges with writes and scans, which aligns with typical behavior observed in NoSQL database systems.**

I. Introduction

With the emergence of many web applications and cloud computing enhancement, managing data on a single node has become increasingly impractical. To address the challenge of distributing large datasets across multiple machines, NoSQL database systems have emerged as a viable solution. For instance, MongoDB [1], Couchbase [2], Redis [3], Amazon DynamoDB [4], Apache Cassandra [5], Google Cloud Bigtable [6], Neo4j [7], ArangoDB [8], and ScyllaDB [9].

Unlike traditional relational databases, MongoDB addresses the challenge of hosting and managing massive datasets. MongoDB uses a distributed system architecture based on sharding [10]. Sharding enables MongoDB to partition and distribute data across multiple nodes, making it possible to scale horizontally by adding more machines to the cluster. This scalability ensures that MongoDB can efficiently manage growing data volumes.

However, distributing data across nodes is governed by the CAP theorem [11], a well-known theorem that needs to be considered when developing any NoSQL database. The CAP theorem is based on availability, consistency, and fault tolerance, which can be seen in Figure 1.

MongoDB is based on a high availability and partition tolerance while trading off the consistency. The consistency in MongoDB is providing tunable consistency, enabling developers to choose the balance that best suits their application needs. Despite its strengths, MongoDB suffers with write and scan operations in distributed environments, where maintaining coordination and synchronization across nodes can introduce latency.

MongoDB uses a flexible schema design where data is stored in JSON-like documents. This approach not only simplifies data representation but also aligns well with the needs of modern web and cloud applications, which often require agility in handling semi-structured and unstructured data. MongoDB stores data in documents that are considered JSON-like. This means that the structure is volatile and can be changed over time. This JSON-ness also means that document structure is not consistent across different documents.

In addition to its schema flexibility, MongoDB supports high-throughput operations and seamless failover mechanisms. These features make it a popular choice among developers and organizations worldwide for a range of applications.

This project explores MongoDB’s system architecture and evaluates its performance in a cloud-based environment, focusing on its ability to handle read, write, and scan workloads effectively. The evaluation conducted using the Yahoo! Cloud Serving Benchmark (YCSB) [12], highlights MongoDB’s strengths and identifies areas where performance challenges arise.

The remainder of this paper is organized as follows: Section 2 provides the background, Section 3 details MongoDB's system architecture, Section 4 presents the experiment settings of the MongoDB system, and Section 5 provides results and analysis of the experiments. Finally, Section 6 summarizes the key conclusions and outlines future work.

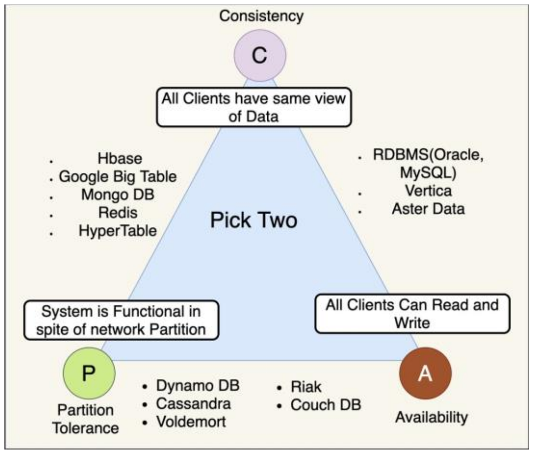


Figure 1: CAP Theorem

II. Background

A wide range of studies, surveys, and technical reports have been conducted on NoSQL (Not Only Structured Query Language) databases, particularly focusing on their architecture, scalability, and performance characteristics. MongoDB is one of the most prominent document-based NoSQL databases. It has been extensively analyzed for its ability to manage distributed systems efficiently. Several works, such as [13], evaluate and analyze CouchDB, MongoDB, and Couchbase. The objective was to help users to select what database is good for their applications.

Furthermore, studies [14] classify MongoDB and other NoSQL databases based on their schema design, noting MongoDB's JSON-like document structure as a significant factor in its flexibility and popularity among developers.

While there has been significant work on comparing NoSQL databases theoretically, fewer studies focus on the performance bottlenecks inherent to systems like MongoDB. Existing research [15] identifies MongoDB's strengths in handling high throughput read operations but also acknowledges challenges with write and scan operations in distributed environments.

The work presented in this project is motivated by the need to better understand MongoDB's performance in real-world scenarios. By evaluating MongoDB's ability to handle read, write, and scan workloads using the Yahoo! Cloud Serving Benchmark (YCSB), this study aims to identify optimization opportunities and provide insights for improving MongoDB's performance in distributed environments.

III. System Architecture of MongoDB

MongoDB is a popular database option, and part of the reason for that is its uniqueness in its database functionality. We are now going to go in-depth into the architecture of MongoDB and how it is able to accomplish its tasks. formatted and styled.

A. *The Document Model*

MongoDB retains the best aspects of relational and NoSQL databases while providing a technology foundation that enables organizations to meet the demands of modern applications [15]. Seeing that MongoDB is built to be an optimized hybrid of sorts, it must have its own unique way of storing data, from here we arrive at the document model. MongoDB opts to use JSON-like documents that can store any type of data. The principle aim behind this is to take away the restrictions and formatting necessary to maintain a standard relational database (tables and columns).

In general, there are three main goals of the document model:

- Intuitiveness. MongoDB maps documents directly to objects in your code, which aims to reduce unnecessary restructuring of data. If data is stored together, it is likely accessed together, and thus this method can increase code efficiency by reducing unnecessary information structuring and one-to-one mapping [16].

- Flexibility. Document schemas are self-described, meaning that there does not need to be any forethought of pre-defining it in the database. Fields within documents vary from document to document and modifying the document is always possible. This allows for more intuitive development of code. The database will not need to run costly `alter table` commands, as MongoDB can allow for multiple versions of documents in the same space. This is great for application development. MongoDB also offers schema validation if you choose to enforce structure rules for your documents [16].

- Universality. JSON, it is all JSON. JSON itself is a language independent standard for data transfer and storage. The versatility of JSON documents allows MongoDB to be useful in a variety of different applications. MongoDB takes JSON one step further though; it specifically uses Binary JSON. This unique form of JSON allows for more specific use of data types that can be better represented in binary data over string values. This assists applications in their processing tasks [16].

B. *Providing Availability and Scalability*

One of the many wonderful aspects of MongoDB is its distributed architecture. This design choice leads to great benefits in the fields of availability and scalability.

In terms of availability, MongoDB utilizes replica sets. These replica sets enable the user to create up to 50 copies of your data. This allows for a large amount of data safety, especially since these copies can be spread out over multiple data centers, nodes, and regions. MongoDB specifically tailored replica sets for resilience. To bolster this, MongoDB can utilize what is called "write concern" to write data to multiple replicas. This helps ensure data availability, although this may impact performance. These write concerns can customized to the user's discretion. Replica sets aren't just used for data resiliency, however. Replicas can be used to scale read operations, which in turn can minimize queries waiting on a specific node.

MongoDB is able to be scaled vertically to larger or smaller instances. Along with this, MongoDB has rolling restart functionality to minimize downtime. Native sharding is a tool that MongoDB uses to help automatically scale your instances write operations. This sharding can also be used to scale your application should it outgrow a single server. Below is Figure 2 which shows sharding graphically. Sharding has many forms:

- Ranged Sharding involves storing documents across shards according to the shard key value. Documents with closer shard key values are more likely to be located within the same shard. Great for distance applications [16].
- Hashed Sharding involves distributing documents based on a md5sum performed on the document. Helps to ensure uniform distribution of writing [16].
- Zoned Sharding involves defining specific rules that determine where documents go [16].

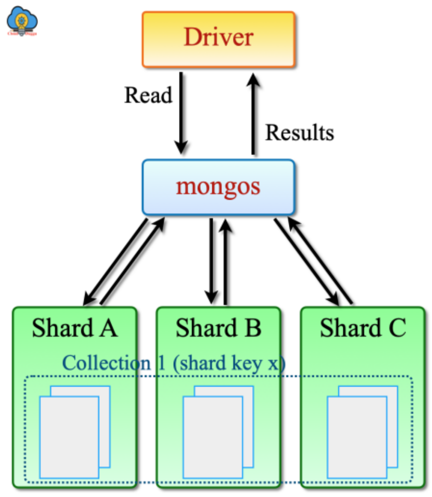


Figure 2: MongoDB Sharding

C. *Privacy and Security*

MongoDB also has a variety of security and privacy benefits due to its design. These are especially important in their application as a cloud-based database solution. Cybersecurity reliability is an important factor when choosing any function for your applications. Here are just a few of the high points of MongoDB's security features:

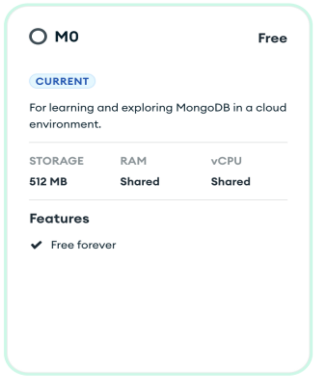
- Authentication: MongoDB utilizes SCRAM-256 with integration into various enterprise security infrastructure to ensure proper authentication [17].
- Authorization: Role-Based Access Controls (RBAC) can be used to limit who has access to certain data [17].
- Auditing: MongoDB has an audit log which can be used by security auditors [17].
- Network Isolation: Cloud database solutions are hosted in their own Virtual Private Cloud (VPC) which means there is no intersection of different user databases [17].
- Encryption: Data within MongoDB can be encrypted through all stages of data flow (transfer, storage, etc.) [17].

IV. Performance Experiment

For our experiment, we deployed MongoDB in a distributed environment using virtual machines, MongoDB Atlas [18]. We conducted performance tests on this cluster by running the Yahoo! Cloud Serving Benchmark (YCSB) [12] against MongoDB. Below, we provide detailed specifications of the experimental setup, including the environment and configuration.

A. *Environment Specifications*

For the testing environment, we had a cluster of VMs that had shared vCPUs, Shared RAM, max storage limit of 512 MB allocated, and shared bandwidth with no specific throughput guarantees. For the database configuration cluster, we utilized the free tier of AWS platform [19]. This gave us an M0 sandbox designed for learning and exploration with limited resources. More specifically the replica set had 3 nodes, and the region was set as us-east-1. The type we had specified was replica set (single cluster) which supported a basic MongoDB deployment with replication for data redundancy. This was illustrated in Figure 3.


M0 Free

CURRENT

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

Features

✓ Free forever

Figure 3: MongoDB Cluster

After the cluster was created, we decided to interact with it using Visual Studio Code. To accomplish this, we installed the MongoDB for VS Code extension. Technically, we opened the command palette and connected to our MongoDB cluster. This was the way to create the playground to run our benchmark against the MongoDB cluster.

To run the Benchmark and evaluate the system of MongoDB, we use an open-source YCSB GitHub repo [20] that allowed us to easily implement YCSB testing with our MongoDB cluster. Cloning GitHub repo [20] onto our testing machine, we now had access to YCSB and the ability to run it on our MongoDB cluster. The last step in configuration is to install a version of Java and Maven. These are necessary for use within our testing environment because Java provides the runtime environment for YCSB, while Maven manages dependencies and builds the workload generator. At this point, a local installation of MongoDB was installed to allow for interaction with our created cluster. Installation was quick and easy thanks to MongoDB's simple installation guide [21].

B. *Experiment Evaluation and Result*

With the fully setup MongoDB environment, YCSB was executed on the cluster to evaluate MongoDB's performance. Specifically, six YCSB workloads (Workload A, Workload B, Workload C, Workload D, Workload E, and Workload F) were tested on the MongoDB cluster.

These workloads are not identical, which provided a benefit to run the tests on different benchmark parameters. With these varied input parameters, it is possible to analyze MongoDB's performance in different areas and thus gather a full picture of its performance. To prepare the database for testing, the YCSB loading command was used to generate all the necessary data for the workloads (A-F). Once data has loaded, the workloads are executed sequentially on the MongoDB cluster using the run commands for each benchmark. This systematic approach facilitated a detailed evaluation of MongoDB's performance under diverse workload conditions.

V. RESULTS AND ANALYSIS

The performance evaluation of MongoDB was conducted by running six standard workloads provided by the Yahoo! Cloud Serving Benchmark (YCSB): Workload A, Workload B, Workload C, Workload D, Workload E, and Workload F. Each workload is designed to simulate a unique application scenario, varying in terms of read, write, update, and scan operations. The following subsections provide a breakdown of the results and an analysis of MongoDB's performance under each workload.

A. *Workload A: Update Heavy*

Workload A consists of 50% read and 50% update operations, simulating use cases like session store updates. The average latency for read and write operations were nearly identical, indicating consistent performance across both operation types in this workload. At the 95th percentile, however, the latency diverged slightly: Read operations: 121 ms Write operations: 112 ms. A significant increase was observed in the 95th percentile latency compared to the average latency. This indicates that while the majority of operations execute efficiently, a subset experiences delays due to system bottlenecks or contention.

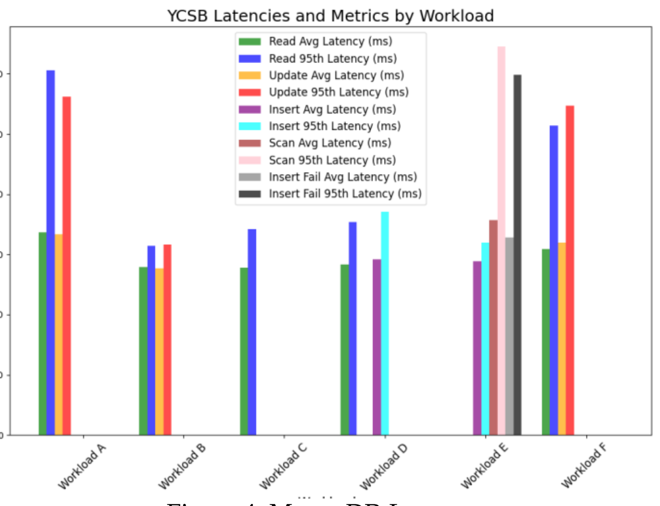


Figure 4: MongoDB Latency

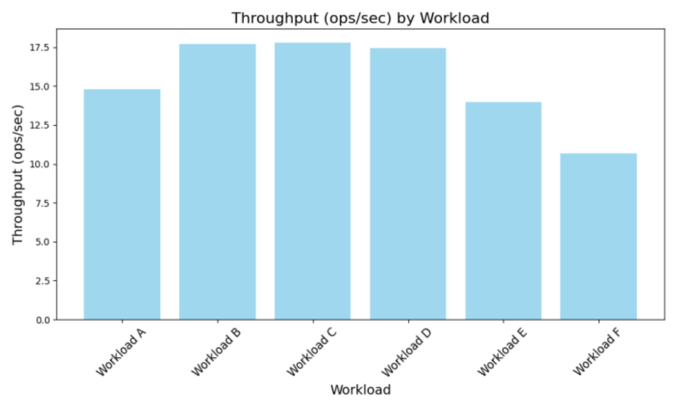


Figure 5: MongoDB Throughput

B. *Workload B: Read Heavy*

Workload B consists of 95% read and 5% update operations, simulates scenarios such as user profile retrieval. MongoDB demonstrates strong performance in this read-heavy workload, achieving an average latency of 56 ms and a throughput of 18 requests per second. These results highlight MongoDB's optimization for read operations. The client can read from any replica in the replica set, which distributed the load to handle high read demands.

C. *Workload C: Read Only*

Workload C consists of 100% read operations. It simulates accessing cached data. MongoDB achieved its highest throughput in this workload with 19 requests per second and minimal latency of 55 ms. The absence of write operations allowed MongoDB to fully leverage its distributed architecture for reads. It utilizes replica sets for read load balancing. This workload underscores MongoDB's strength in handling applications requiring high read scalability.

D. *Workload D: Read Modify-Write*

Workload D consists of 95% read and 5% insert operations, simulating user-driven actions like adding data to a cart. MongoDB maintained steady performance, with an average latency of 55 ms for reads and 74 ms for inserts. The higher latency 74 ms for modify operations reflects the time required for the initial read, the modification logic, and the coordination necessary to maintain consistency across nodes in the cluster.

E. *Workload E: Scan Heavy*

Workload E consists of 95% scan operations and 5% inserts. It highlights MongoDB's limitations in handling analytical queries like content filtering. The average scan latency of 71 ms, higher than the 58 ms for inserts. Cleanup operations were lightweight at 27 ms, indicating efficient resource management, while insert failures averaged 65 ms due to coordination overhead and potential retries in the distributed settings. These results suggest opportunities to optimize MongoDB's query execution and failure handling mechanisms for improved performance for the analytical workloads.

F. *Workload F: Read-Modify*

Workload F consists of 50% reads and 50% read-modify-write operations, reflects scenarios like updating user preferences. MongoDB has an average read-modify-write latency at 124 ms, significantly higher than the 64 ms for updates. This is due to the added complexity of modifying data and synchronizing across nodes. The overall throughput was the lowest at 10.69 ops/sec, highlighting the performance impact of distributed coordination. While MongoDB’s tunable consistency mitigated some latency, the workload revealed limitations in handling concurrent read and write operations.

VI. Conclusion and Future Work

A. Conclusion

This study evaluated MongoDB's system architecture and its performance using YCSB workloads in a cloud-based environment. MongoDB demonstrated strong capabilities in read-heavy scenarios leveraging features like replica sets. However, challenges emerged in write-intensive and scan-heavy workloads. It highlights inefficiencies in handling distributed synchronization and query execution for analytical operations. These findings emphasize MongoDB's suitability for high-throughput, read-dominant applications. However, MongoDB needs an improvement in its coordination mechanisms to better handle mixed or write-heavy workloads.

B. Future Work

Future research will focus on optimizing MongoDB’s performance in distributed environments by improving transaction coordination for read-modify-write operations and addressing latency issues in scan-heavy workloads.

Additionally, integrating machine learning-driven predictive for sharding strategies could enhance its scalability and efficiency for some use cases including analytical workloads. Comparing MongoDB against other NoSQL databases under similar conditions can provide further insights into its relative strengths and limitations.

References

[1] MongoDB, Architecture Guide. MongoDB, Inc. Available at: https://www.mongodb.com/lp/resources/products/fundamentals/mongodb-architecture-guide (Accessed: 27 November 2024).

[2] Couchbase, [Whitepaper] Couchbase Under the Hood: An Architectural Overview. Couchbase, Inc. Available at: https://www.couchbase.com/content/capella/server-arc-overview (Accessed: 27 November 2024).

[3] Da Silva, M. D., & Tavares, H. L. (2015). Redis Essentials. Packt Publishing Ltd.

[4] M. Elhemali, N. Gallagher, B. Tang, N. Gordon, H. Huang, H. Chen, J. Idziorek, M. Wang, R. Krog, Z. Zhu, and others, Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service, in 2022 USENIX Annual Technical Conference (USENIX ATC 22), 2022, pp. 1037–1048.

[5] N. Palmer, M. Sherman, Y. Wang, and S. Just, Scaling to build the consolidated audit trail: a financial services application of Google Cloud Bigtable, Fidelity National Information Services Inc., 2015.

[6] J. Guia, V. G. Soares, and J. Bernardino, Graph Databases: Neo4j Analysis, in Proc. ICEIS (1), 2017, pp. 351–356.

[7] D. Fernandes, J. Bernardino, and others, Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB, Data, vol. 10, p. 0006910203730380, 2018.

[8] N. Suneja, ScyllaDB optimizes database architecture to maximize hardware performance, IEEE Software, vol. 36, no. 4, pp. 96–100, 2019.

[9] S. Solat, Sharding Distributed Data Databases: A Critical Review, *arXiv preprint arXiv:2404.04384*, 2024.

[10] S. Gilbert and N. Lynch, Perspectives on the CAP Theorem, Computer, vol. 45, no. 2, pp. 30–36, 2012.

[11] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, Benchmarking cloud serving systems with YCSB, in Proc. 1st ACM Symp. Cloud Computing, 2010, pp. 143–154.

[12] I. Carvalho, F. Sá, and J. Bernardino, Performance evaluation of NoSQL document databases: Couchbase, CouchDB, and MongoDB, Algorithms, vol. 16, no. 2, p. 78, 2023.

[13] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, NoSQL databases: Critical analysis and comparison, in 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), 2017, pp. 293–299.

[14] A. Phaltankar, J. Ahsan, M. Harrison, and L. Nedov, *MongoDB Fundamentals: A hands-on guide to using MongoDB and Atlas in the real world*. Packt Publishing Ltd, 2020.

[15] MongoDB, MongoDB: The Definitive Guide. O’Reilly Media. Available at: https://www.oreilly.com/library/view/mongodb-the-definitive/9781449381578/ (Accessed: 27 November 2024).

[16] J. Kumar and V. Garg, Security analysis of unstructured data in NoSQL MongoDB database, in 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), 2017, pp. 300–305.

[17] *MongoDB atlas* (no date c) *MongoDB*. Available at: https://www.mongodb.com/cloud/atlas/register (Accessed: 27 November 2024).

[18] Amazon Web Services, Amazon Web Services (AWS) - Cloud Computing Services. Amazon.com, Inc., 2024. Available at: https://aws.amazon.com/ (Accessed: 27 November 2024).

[19] mongodb-labs, YCSB/ycsb-mongodb, GitHub. Available: https://github.com/mongodb-labs/YCSB/tree/main/ycsb-mongodb. Accessed: Nov. 28, 2024.

[20] mongodb-labs, YCSB/ycsb-mongodb. GitHub. Available at: https://github.com/mongodb-labs/YCSB/tree/main/ycsb-mongodb (Accessed: 28 November 2024).

[21] MongoDB, What is MongoDB Atlas?. MongoDB, Inc. Available at: https://www.mongodb.com/docs/atlas/ (Accessed: 27 November 2024).