Journal of Distributed Computing and Systems(JDCS)
Vol.6, Issue 2, 2024, Pages 25-32

Comparison of Relational database storage engines with Artificial Intelligence algorithm leveraging RAID technique to enhance performance

Mostafa Ramezani Eshtajarani[1], Seyed Behnam Hoseini[2] *

[1]*Department of Criminal and Criminology, Faculty of Law and Political Sciences, University of Allameh Tabataba'i, Tehran, Iran.*

[2]*Department of Software Engineering, Faculty of Computer Engineering, University of Pooyesh, Qom, Iran.*

Article History:
Received: 10 November 2023
Accepted: 16 March 2024
Available online: 29 March 2024

Abstract

The ever-increasing volume of data demands efficient storage solutions. This research investigates optimizing cloud data storage using Artificial Intelligence (AI) monitoring to achieve high query processing rates (Queries Per Second). We compare the performance of MariaDB and MongoDB database engines, focusing on data compression, query execution time and CPU usage. Our approach utilizes AI for real-time monitoring and potential optimization strategies. Employing the TPC-H benchmark, we demonstrate that MongoDB achieves an average compression rate that is 43% superior to that of MariaDB. Conversely, MariaDB outperforms MongoDB in query execution speed, exhibiting an average performance that is 2.7 times faster, as well as in CPU usage, where it demonstrates an average reduction of 5.9 times lower consumption. These findings suggest a trade-off between compression efficiency and query performance when choosing between these database engines.

Keywords: Cloud Storage, Data Compression, MongoDB, MariaDB, Artificial Intelligence.

I. Introduction

In the field of data management, the use of Artificial Intelligence (AI) algorithms has gained significant attention in recent years. This is particularly evident in the comparison of relational database storage engines with AI algorithms. Relational DataBase Management Systems (RDBMS), such as Oracle, IBM and Microsoft, have long been the go-to choice for managing structured data. However, the advent of AI has led to the exploration of new approaches in data storage and retrieval. This article aims to explore and compare the benefits and limitations of using relational database storage engines with AI algorithms [1].

In today's era of exponential data growth, traditional relational databases struggle to efficiently handle the influx of unstructured and semi-structured data. This challenge is expected to become even more pronounced with the rise of Web 3 technologies. Web 3, envisioned as a decentralized and user-centric internet and is expected to generate vast amounts of diverse data, including user-controlled data and data associated with decentralized applications. These new data types may not conform to the rigid structures of traditional relational models, driving the need for more flexible data management solutions [2].

The explosion of data in the digital age, fueled in part by Web 2.0 and the anticipated growth of Web 3, necessitates the development of new database solutions. Not only Structure Query Language (NoSQL) databases have emerged as a powerful response to this challenge. These databases offer greater scalability, parallel processing capabilities and can be deployed as database-as-a-service solutions, making them well-suited for handling the diverse and ever-growing data landscape [3].

To answer this question, it is important to understand the fundamental differences between various data storage approaches. One such approach is document-oriented data storage. In this model, data is stored in the form of documents, akin to a sub-class of key-value storages. Document-based storages bear some similarities to relational databases., but with a notable distinction, while relational databases store data in multiple tables, document-oriented databases store the data in a single instance for a given object. This difference allows for greater flexibility and scalability, making document-oriented databases well-suited for handling dynamic and ever-evolving data sets [4]. Relational database storage engines, with their proven track record of stability and reliability, can still provide strong support for AI algorithms. Their structured nature allows for well-defined schemas, supporting complex queries and aggregations. However, when it comes to handling unstructured or semi-structured data, such as text, images, or video, relational databases can be less efficient. The rigid structure of tables and predefined relationships can limit the flexibility required for AI algorithms to extract meaningful insights from diverse data sources [5].

Document-oriented databases, exemplified by MongoDB and CouchDB, provide flexibility in storing and retrieving

* Corresponding Author: Seyed Behnam Hoseini (behnam.hoseini1989@gmail.com)

document-based data, making them well-suited for content management systems, blogging platforms and collaboration tools. On the other hand, graph-oriented databases, such as Neo4j and Amazon Neptune, excel in managing and traversing complex relationships between data elements. This makes them an excellent fit for social networks, recommendation engines and fraud detection systems [6].

This research investigates the performance of MariaDB and MongoDB, two popular DataBase Management Systems (DBMS). We observe a close competition in efficiency dimensions like query processing speed between these relational and non-relational databases. This competition motivates the exploration of further optimization, particularly in storage space utilization.

This paper makes the following contributions:

- Comprehensive analysis: We conduct a comprehensive analysis of existing Hard Disk Drive (HDD) based databases, focusing on the performance characteristics of the WiredTiger and InnoDB storage engines.
- AI-driven storage optimization: We propose a real-time AI monitoring system that tackles the challenge of balancing fast Input/Output (I/O) during data import with efficient storage space utilization.
- Enhanced MongoDB performance for large data imports: We demonstrate how AI can improve the performance and efficiency of MongoDB specifically when handling large data insertions or imports.

The remaining of this paper is organized as follow. Section 2 surveys the recent related works. Section 3 introduces the Artificial Intelligence algorithm with TPC-H benchmark. Section 4 presents a detailed case study to illustrate the workflow of the proposed approach. Section 5 reports the experimental evaluation of the proposed approach. Finally, Section 6 concludes the paper.

II. RELATED WORKS

Data compression plays a crucial role in optimizing database performance. By reducing the amount of data stored, compression significantly lowers Input/Output operations. This translates to faster query execution times and improved overall database efficiency. Furthermore, optimizing queries alone may not necessarily address I/O bottlenecks. Data compression offers a two-level optimization approach. It not only reduces storage requirements on the storage layer but can also lead to more efficient queries by minimizing the amount of data that needs to be processed. These storage space savings can be financially beneficial by reducing storage hardware costs and functionally beneficial by allowing storage of larger datasets on existing hardware.

Commercial systems such as Oracle's Times Ten [7], IBM's solidDB [8] and VoltDB [9] have gained popularity due to their efficiency in handling small-scale databases. These systems are suitable when the amount of data is smaller than the available physical memory. However, when the data exceeds the memory capacity, performance problems arise. To tackle this issue, recent research has focused on developing systems specifically designed to handle large-scale databases. One such research system is HYRISE [10], a main-memory database management system developed at the Hasso Plattner Institute in Germany. The system utilizes AI algorithms to optimize query execution and data storage. By analyzing the query workload, HYRISE intelligently caches frequently accessed data in memory, which improves the overall performance of the system. Additionally, HYRISE employs AI algorithms to compress and encode data, further enhancing storage efficiency.

Another notable research system is H-Store [11], which was developed at the Massachusetts Institute of Technology (MIT). H-Store aims to provide high scalability and fault tolerance by utilizing a distributed architecture. With the integration of AI algorithms, H-Store dynamically adapts its data distribution strategy to optimize performance based on the workload. This self-tuning capability enables H-Store to efficiently handle large-scale databases.

HyPer [12], another research system, focuses on providing high-speed transaction processing for analytical workloads. By combining in-memory processing with AI algorithms, HyPer is able to achieve exceptional performance. The system leverages predictive analytics and machine learning techniques to enhance query optimization and processing. This intelligent approach allows HyPer to adaptively allocate resources based on the workload, resulting in improved query response times.

MonetDB [13], developed at the Centrum Wiskunde & Informatica in the Netherlands, is a columnar database management system that utilizes AI algorithms for query optimization and indexing. By understanding the data access patterns, MonetDB employs machine learning algorithms to dynamically adjust the data layout and index structures. This intelligent indexing technique drastically reduces query response times and enhances overall system performance.

In the contemporary software landscape, the proliferation of complex systems across diverse platforms has ushered in a new era where performance considerations reign supreme. The advent of artificial intelligence algorithms has further accentuated the need for efficient and high-performing software components, particularly in the realm of relational database storage engines. This article seeks to elucidate the interplay between software performance and artificial intelligence, with a specific focus on the comparison of relational database storage engines in this dynamic environment [14]. Relational Database Storage Engines and Artificial Intelligence: Within the realm of artificial intelligence applications, relational database storage engines play a crucial role in enabling efficient data storage and retrieval mechanisms. The performance of these storage engines directly impacts the speed and efficiency of AI

algorithms that rely on vast amounts of data for training and inference. By comparing the performance characteristics of different relational database storage engines, developers can make informed decisions regarding the selection of the most suitable platform for their AI applications [15].

The article [16] outlines the methodology employed to compare MongoDB and Oracle, encompassing performance benchmarks, feature assessments and real-world use cases. Through a comprehensive evaluation framework, the authors aim to provide a holistic view of the strengths and limitations of each database system, enabling readers to make informed decisions based on their specific requirements. The comparative analysis delves into various aspects of MongoDB and Oracle, including data modeling capabilities, query performance, scalability, security features and ecosystem support. By juxtaposing the strengths and weaknesses of each database system, the article offers valuable insights into the trade-offs involved in selecting between MongoDB and Oracle for different applications and workloads [16].

The authors in [17] "NoSQL Databases: MongoDB vs Cassandra" serves as a valuable resource for database professionals and organizations navigating the landscape of NoSQL technologies. By presenting a detailed comparison of MongoDB and Cassandra across key dimensions, the authors empower readers to make informed decisions that align with their data management goals and performance requirements in the context of modern computing environments [17].

TABLE. I summarizes the comparison between MongoDB and MariaDB storage engine databases. Unlike traditional relational databases that store data in rigid tables, MongoDB uses flexible JSON-like documents. This dynamic structure allows for easier and faster integration of data in specific applications [4]. The research study in [18] "Performance Evaluation of SQL and MongoDB Databases for Big E-commerce Data" contributes valuable insights to the field of database management in e-commerce applications. By presenting a detailed performance evaluation of Structure Query Language (SQL) and MongoDB databases, the authors provide e-commerce businesses and database professionals with valuable information to make informed decisions when selecting a database platform for managing large-scale e-commerce data effectively [18].

The WiredTiger storage engine in MongoDB offers two compression algorithms, snappy and zlib. These algorithms significantly reduce the storage footprint by compressing data on disk. This leads to efficient disk utilization and potentially faster I/O operations, improving overall query performance. Essentially, Artificial Intelligence methods can be employed to further enhance performance. AI can analyze access patterns and predict frequently accessed data or specific query results. By proactively caching this data in memory, AI can significantly reduce disk I/O operations, leading to a faster and more responsive database system.

TABLE I. COMPARING THREE STORAGE ENGINES OF MONGODB AND MARIADB DATABASES.

Features	MMAPv1	WiredTiger	InnoDB
Write Performance	Good	Excellent	Excellent
Read Performance	Excellent	Excellent	Good-Excellent
Compression Support	NO	YES	YES
Query Language Support	YES	YES	YES
Secondary Index Support	YES	YES	YES
Replication Support	YES	YES	YES
Sharding Support	YES	YES	YES
Ops manager and MMs support	YES	YES	YES
Security Controls	YES	YES	YES
Platform Availability	Linux, Windows, Mac OS X, Solaris (x86)	Linux, Windows, Mac OS X	Linux, Windows, Mac OS

III. COMPARISON BETWEEN MONGODB AND MARIADB WITH ARTIFICIAL INTELLIGENCE

WiredTiger is a high-performance storage engine introduced as the default option in MongoDB version 3.2. It offers significant advantages over the previously used MMAPv1 engine (Memory-Mapped Storage Engine Version 1), including improved performance, scalability and concurrency control, allowing MongoDB to handle larger datasets and heavier workloads more efficiently. One key feature of WiredTiger is its support for data compression algorithms. As illustrated in Fig. 1[2], WiredTiger offers two options: Snappy and Zlib. Snappy prioritizes speed, making it ideal for applications where fast read and write operations are critical. While Snappy may not achieve the same level of compression as Zlib, it offers a significant reduction in storage footprint compared to uncompressed data, leading to faster I/O operations and improved overall database performance.

[2] www.mongodb.com

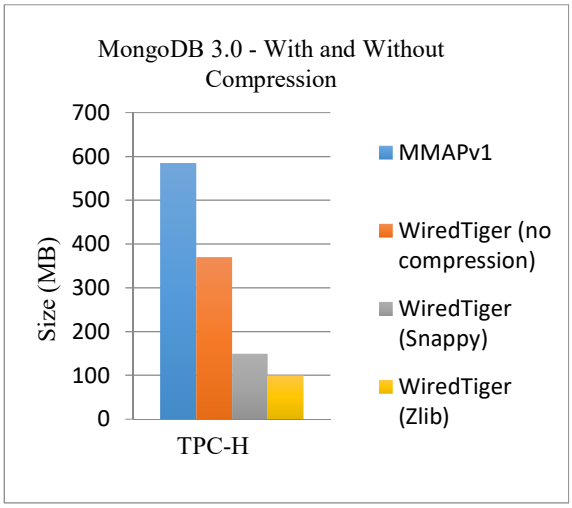


Fig. 1. Compression algorithms in WiredTiger storage engine.

Unfortunately, AI methods are not directly integrated into the core functionalities of storage engines like WiredTiger or InnoDB used by MongoDB and MariaDB. However, there's potential for external AI tools to complement these storage engines in specific areas like query optimization.

While AI is not inherently embedded within storage engines like WiredTiger or InnoDB, external AI tools can be leveraged to analyze query patterns and historical data. This analysis can then be used to suggest or automatically generate optimized query execution plans. This approach has the potential to improve database performance by selecting the most efficient plan for each query, as demonstrated in research [15]. This revised text clarifies that AI acts as an auxiliary tool rather than a core component of storage engines. It also emphasizes the role of research [15] to support the potential benefits.

On the other hand, a few example of how AI can be applied in these contexts:

A. Indexing and Data Organization

AI techniques can be used to automatically identify and create and appropriate indexes based on query pattern and data distribution. This can help improve query performance by reducing the need for full table scans.

B. Anomaly Detection

AI algorithms can monitor database operations and detect anomalies in real-time. This can help identify potential security breaches, performance issues, or unusual patterns in data access.

C. Predictive Analytics

By analyzing historical data and patterns, AI algorithms can make predictions about future data access and query patterns. This information can be used to optimize storage engine configurations or allocate resources more efficiently.

Creating a complete Artificial Intelligence algorithm for database storage engines would require significant time and effort, as it involves complex data analysis and machine learning techniques. However, we are provided a high-level overview of the steps involved in simulating and AI algorithm for query optimization:

- Data collection

Gather a large dataset of historical query logs, including information such as query text, execution time and resource usage.

- Preprocessing

Clean and preprocess the query logs to remove noise and irrelevant information. This may involve techniques such as removing outliers, normalizing data and feature engineering.

- Feature extraction

Extract relevant features from the preprocessed query logs. This features good could include query text, table names, column names, join conditions and various statistics about the queries and their execution.

- Training data preparation

For model development, the preprocessed data will be divided into training and testing sets. The training set will be used to train the AI model and the testing set will assess its performance.

- Model selection

Choose an appropriate machine learning algorithm for query optimization, such as decision trees, random forests or neural networks. Consider the specific requirements and constraints of the database storage engine.

- Model training

Train the selected AI model using the training data. The model will learn patterns and relationships between query features and their corresponding execution plans or performance metrics.

- Model evaluation

Evaluate the trained model using the testing data to measure its performance and identify any potential issues like overfitting or underfitting.

- Model deployment

Once the model is trained and evaluated, integrate it into the database storage engine. This could involve developing a plugin or extension that interacts with the storage engines query optimizer.

- Real-time monitoring and adaptation

Continuously monitor the database systems performance and collect new query logs. Use this data to update and refine the AI model periodically to adapt to changing query patterns or system conditions.

IV. Case Study

This section presents a detailed case study comparing the performance of MongoDB and MariaDB when used with an AI algorithm. We will analyze a large relational database containing a range of data points (between 100,000 and 1,500,000 records) on various attributes such as Region, Country, Item Type, etc. (list all attributes). This case study aims to assess the suitability of each DataBase Management System for handling large datasets and supporting AI-powered analytics [4].

The proposed AI monitoring can potentially improve various performance aspects of these database engines, including data import speed, compression ratio and CPU consumption required to commit transactions. Since both InnoDB and WiredTiger storage engines have the potential for high performance, AI can further enhance their capabilities by pushing the average limits of these metrics:

- Experimental Queries
- Block Management in MongoDB
- Impact of AI Monitoring

By default, an example code snipped to import data into MongoDB using Python:

```
python
import pymongo
import json
# Connect to MongoDB
client = pymongo
.MongoClient("mongodb://localhost:27017/")
db = client["your_database_name"]
collection = db["your_collection_name"]
# Read the data from a JSON file with open ('data.json') as f:
data = json.load(f)
# Insert the data into the collection
collection.insert_mant(data)
print ("Data imported successfully!")
```

Note that this code assumes you have a JSON file with an array of objects that you want to import into MongoDB. Each object in the array represents a document in the collection.

A. *LOAD DATA INFILE implementation method*

The LOAD DATA INFILE method is significantly faster than traditional INSERT statements when inserting large datasets[3]. However, the second approach can be to compare with the MariaDB relational database. Its run using the following code:

```
LOAD DATA INFILE 'Sales Records.csv' INTO TABLE table-name;
```

B. *RAID (Redundant Array of Independent Disks) implementation method*

In this approach we have read old data parity and write new data and parity. Software RAID affects overall system performance.

- For RAID 1+0, the disk load (read + write)

= (1200 * 2/3) + (1200 * (1/3) * 2) = 800 + 800 = 1600 IOPS (Input/Output Operations Per Second)

For example In RAID 1+0 solution penalty backend I/O require on storage is 1600 IOPS.

V. Experimental Evaluation Of Three Level Optimization With Artificial Intelligence

To enhance AI monitoring's effectiveness, we deployed it on both MongoDB and MariaDB databases. This section delves into the experimental setup, workload characteristics, the specific AI employed and concludes with a performance comparison.

A. *Experiment setup*

TABLE. II demonstrates summaries two different types of hardware details for databases.

TABLE II. HARDWARE RESOURCES For IMPLEMENTING MongoDB And MariaDB DATABASES.

Database	Software	CPU	RAM	Storage
MongoDB v5.0.10	Studio 3T	Intel Corei3 380M (4*2.53 GHz)	6GB	Hard Disk Drive (HDD)
MariaDB v10.1.25	phpMyAd-min	Intel Core2Duo T9400 (2*2.53 GHz)	2GB	Hard Disk Drive (HDD)

B. *Workload*

To ensure a standardized and comparable evaluation, we employed the well-known TPC-H benchmark [19], which has been widely used in reputable research on database performance [2, 4, 15]. Our chosen workload for MariaDB involves the "customers" table, while for MongoDB, we utilize the "Sales Records" collection. Both datasets contain a substantial amount of data, with the "customers" table holding 3,200,000 rows and the "Sales Records" collection containing 3,100,000 rows. Each table/collection schema consists of a specific number of columns (11 for "customers" and 14 for "Sales Records"). Fig. 2 illustrates the schema of the created collections. We specifically chose these tables/collections from the TPC-H benchmark because they represent commonly used data structures in sales analysis, which aligns with our research focus on AI-powered optimization for this domain.

[3] dev.mysql.com

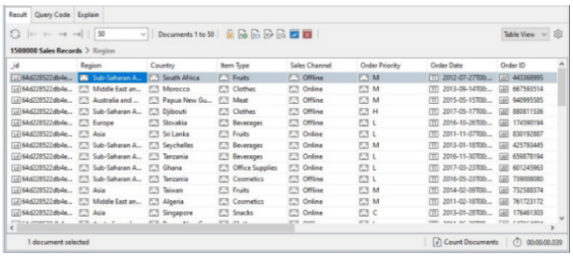


Fig. 2. Import csv file to MongoDB relational database.

C. Experiment artificial intelligence

We base all experiments on two variants of the following query:

- Import from CSV file to MongoDB collection;
- LOAD DATA INFILE '100000 Sales Records.csv' INTO TABLE customers;

Block Management is essential in multiple domains. Firstly, it facilitates the retrieval of missing records and their subsequent addition to the cache when it is empty, thereby enhancing access speed. Secondly, Block Management employs various data compression techniques available in MongoDB, which can lead to a substantial reduction in storage requirements, with the potential to decrease storage space by up to 80%.

The scenario proposed by the author regarding the comprehensive monitoring of artificial intelligence over critical performance metrics of two widely-used relational databases may effectively constrain the analysis to the proportions of data import speed, compression rates and the degree of CPU utilization required for transaction commitment.

As the InnoDB and WiredTiger storage engines enhance their performance capabilities, artificial intelligence has the potential to further elevate the average performance thresholds.

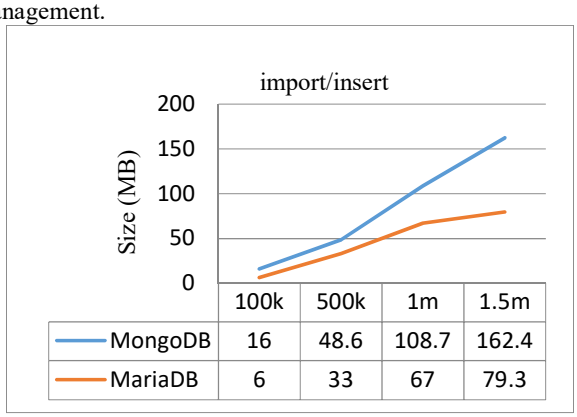
D. Performance comparison

This section presents the experimental evaluation comparing the effectiveness of data import in MongoDB and data insertion in MariaDB. We focus on three key performance metrics: data compression ratio, query execution time and CPU consumption. The experiment involves two datasets: "Sales Records" and "customers". We evaluate both databases for handling horizontal scaling (adding new rows) and vertical scaling (adding new columns) of data. Data is loaded using the appropriate methods (import for MongoDB and insert for MariaDB) into these tables. Following the import/insert operations, we create various analytical queries and compare the performance of both databases when executing these queries.

1) Data compression

Figure 3 illustrates the storage space usage and compression ratio for both MongoDB and MariaDB after the data import process. As shown in the Figure 3, The total storage space used by the MongoDB database is 335.7 MB, while the total data size is 945.5 MB. This translates to an impressive compression ratio of approximately 64.5%. In comparison, the MariaDB database utilizes 153.1 MB of storage space for 194.9 MB of data, resulting in a compression ratio of 21.45%. These results clearly demonstrate that MongoDB achieves significantly higher data compression efficiency (around 43% improvement) compared to MariaDB.

Furthermore, incorporating AI monitoring into this process has the potential to further optimize storage space utilization. AI can analyze data access patterns and identify opportunities for additional compression techniques or data reorganization, leading to even more efficient storage management.



	100k	500k	1m	1.5m
MongoDB	16	48.6	108.7	162.4
MariaDB	6	33	67	79.3

Fig. 3. Data compression improvements for all collections and tables.

2) Query execution time

Figure 4 illustrates the average data import speed for both MongoDB and MariaDB during the experiment. As shown in the Figure 4, MongoDB imports data at an average rate of 7,477 rows per second, while MariaDB exhibits a significantly faster ingestion speed of 20,248 rows per second. This translates to MariaDB being approximately 2.7 times faster than MongoDB for data import tasks.

While MongoDB achieves a higher compression ratio, as discussed earlier, MariaDB demonstrates a clear advantage in terms of data loading speed. Here, incorporating AI monitoring presents an exciting opportunity. AI can potentially analyze data patterns and optimize the data import process, potentially leading to significant improvements in data ingestion speed for MongoDB and potentially narrowing the gap between the two databases.

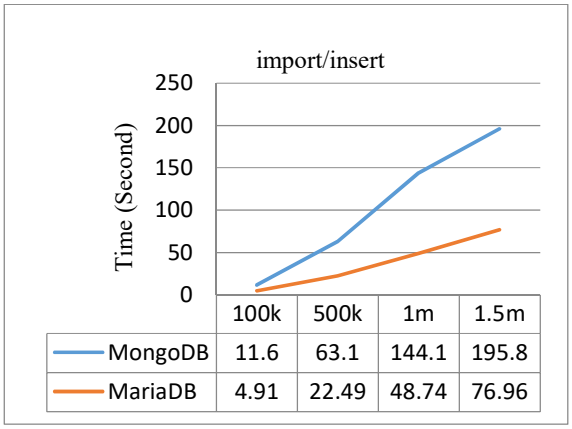


Fig. 4. Execution time improvements for all collections and tables.

3) CPU consumption

Figure 5 illustrates the CPU consumption by both MongoDB and MariaDB during the data import process. Even though the MongoDB database was allocated twice the number of CPUs compared to MariaDB, we observed significantly lower CPU usage by MariaDB. As shown in the Figure 5, MariaDB exhibits a 5.90 times reduction in CPU consumption compared to MongoDB for this task.

This result highlights a trade-off between compression efficiency and CPU usage. While MongoDB achieves a higher compression ratio (as discussed earlier), it comes at the cost of greater CPU consumption during data import. Here, AI monitoring presents another potential benefit. By analyzing data patterns and optimizing the data import process, AI could potentially help reduce CPU usage in MongoDB, leading to a more balanced performance profile.

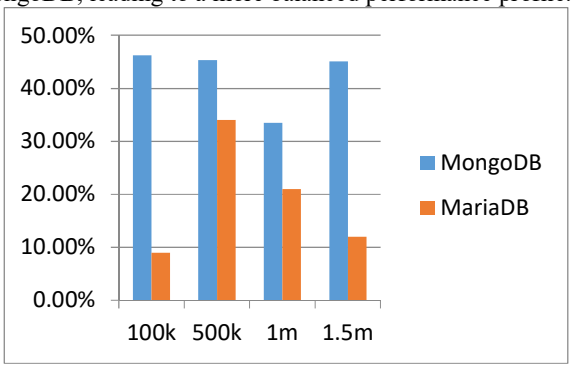


Fig. 5. CPU consumption improvements for all collections and tables.

VI. CONCLUSION

In this research, we investigated the challenges of optimizing storage efficiency for large datasets in cloud environments. We compared the performance of two popular DataBase Management Systems (DBMS): MongoDB (Not only Structure Query Language) and MariaDB (relational). The evaluation employed the TPC-H benchmark and involved real-time monitoring with an AI algorithm to assess data compression, import speed, query execution time and CPU consumption.

The experimental results revealed several key insights. MongoDB achieved a significantly higher compression ratio (43% on average) compared to MariaDB. However, MariaDB demonstrated a clear advantage in data import speed (2.7 times faster) and CPU consumption (5.9 times lower) during the import process. This highlights a trade-off between compression efficiency and data ingestion performance.

REFERENCES:

[1] Rouse, M., "database management system (DBMS)." Definition from WhatIs. com, 2015.

[2] Gundreddy, R.R., "Performance Evaluation of MMAPv1 and WiredTiger Storage Engines in MongoDB: An Experiment." 2017.

[3] Tudorica, B.G. and C. Bucur. "A comparison between several NoSQL databases with comments and notes." in 2011 RoEduNet international conference 10th edition: Networking in education and research. 2011. IEEE.

[4] Chickerur, S., A. Goudar, and A. Kinnerkar. "Comparison of relational database with document-oriented database (mongodb) for big data applications." in 2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA). 2015. IEEE.

[5] Kumar Kaliyar, R. "Graph databases: A survey." in International Conference on Computing, Communication & Automation. 2015. IEEE.

[6] Akhtar, A., "Popularity Ranking of Database Management Systems." arXiv preprint arXiv:2301.00847, 2023.

[7] Lahiri, T., M.-A. Neimat, and S. Folkman, "Oracle TimesTen: An In-Memory Database for Enterprise Applications." IEEE Data Eng. Bull., 2013. 36(2): p. 6-13.

[8] Lindström, J., et al., "IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability." IEEE Data Eng. Bull., 2013. 36(2): p. 14-20.

[9] Stonebraker, M. and A. Weisberg, "The VoltDB Main Memory DBMS." IEEE Data Eng. Bull., 2013. 36(2): p. 21-27.

[10] Grund, M., et al., "Hyrise: a main memory hybrid storage engine." Proceedings of the VLDB Endowment, 2010. 4(2): p. 105-116.

[11] Kallman, R., et al., "H-store: a high-performance, distributed main memory transaction processing system." Proceedings of the VLDB Endowment, 2008. 1(2): p. 1496-1499.

[12] Kemper, A. and T. Neumann. "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots." in 2011 IEEE 27th International Conference on Data Engineering. 2011. IEEE.

[13] Boncz, P.A., M. Zukowski, and N. Nes. "MonetDB/X100: Hyper-Pipelining Query Execution." in Cidr. 2005.

[14] Ousterhout, J., "Always measure one level deeper." Communications of the ACM, 2018. 61(7): p. 74-83.

[15] Fedorova, A., et al. "Performance comprehension at WiredTiger." in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018.

[16] Boicea, A., F. Radulescu, and L.I. Agapin. "MongoDB vs Oracle--database comparison." in 2012 third international conference on emerging intelligent data and web technologies. 2012. IEEE.

[17] Abramova, V. and J. Bernardino. "NoSQL databases: MongoDB vs cassandra." in Proceedings of the international C* conference on computer science and software engineering. 2013.

[18] Aboutorabi[a], S.H., et al. "Performance evaluation of SQL and MongoDB databases for big e-commerce data." in 2015 international symposium on computer science and software engineering (CSSE). 2015. IEEE.

[19] Council, T.P.P., "TPC-H, a decision support benchmark." 2014, Technical report, TPC, www.tpc.org.

Mostafa Ramezani Eshtajarani holds a Master's degree in Law with a specialization in Criminal Law and Criminology from Allameh Tabataba'i University, Tehran, which he obtained in 2019. He also received his Bachelor's degree in Law from Zabol University, in 2013. Currently, he is the director of a legal institute. He is interested in conducting interdisciplinary research and artificial intelligence.

Seyed Behnam Hoseini is an outstanding graduate with Bachelor's and Master's degrees in Computer Engineering, specializing in Software, from Islamic Azad University of Tehran East and Pooyesh University, respectively, in the years 2013 and 2018. He holds a specialized certification in ISMV4 (EMC Information Storage and Management Version 4.0) from Arjang Higher Education Institute in 2022. His research interests include database systems, artificial intelligence and network security.