JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 39 (2026) 108–114
Received: 10 October 2025
Accepted: 3 November 2025

Comparative analysis of non-relational databases on the example of Amazon DynamoDB and MongoDB

Michał Sagan*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This article compares two non-relational database systems: Amazon DynamoDB and MongoDB. The aim of the study was to identify the more efficient system. For this purpose, identical databases have been prepared in four environments: MongoDB Local, DynamoDB Local, MongoDB Atlas, and DynamoDB AWS. Research scenarios were designed to measure the execution time of database operations such as searching, adding, editing, and deleting data. The scripts enabling the measurement of these operations have been prepared in the JMeter program. Additionally, based on one of the research scenarios, it has been checked which database system better optimizes queries.

Keywords: performance analysis; Amazon DynamoDB; MongoDB

*Corresponding author
Email address: michal.sagan1@pollub.edu.pl (M. Sagan)

1. Introduction

With the dynamic development of information technologies and the growing demand for processing vast amounts of data, new challenges related to data management have emerged. Traditional relational databases are still widely used [1], yet they are not always able to meet the requirements of modern applications particularly those that must handle large volumes of data while ensuring high availability and flexibility. In response to these needs, non-relational databases such as Amazon DynamoDB and MongoDB have appeared, offering innovative approaches to data management that go beyond the limitations of relational databases. Thanks to their flexibility and scalability, these databases are often preferred in projects that require fast data access and high availability.

Choosing the appropriate database system is crucial for any application, especially in the context of web applications, where speed and reliability play a decisive role. Web applications, accessed through internet browsers, enable users to reach functionalities and data anytime and anywhere, which is one of their key advantages. Nevertheless, the efficiency of these applications largely depends on the performance of the database that handles user queries, manages data, and ensures its consistency.

The aim of this article is to conduct a detailed comparative analysis of two popular non-relational databases: Amazon DynamoDB and MongoDB. Both systems have gained significant popularity in the IT industry. Despite many similarities, such as the document-oriented data storage model and cloud integration, they differ in terms of architecture and data management mechanisms, which makes them interesting subjects of research.

The choice of this research topic is justified by the growing interest in non-relational database technologies. Although there are many publications focusing on individual database systems, comprehensive comparisons that could help application developers choose the appropriate tool for a specific use case remain limited. This article aims to fill that gap by providing a detailed and critical review of both systems.

Based on the obtained results, it will be possible to confirm or reject the following hypothesis: “MongoDB is a more efficient database than DynamoDB in terms of the time required for inserting, retrieving, updating, and deleting data.”.

2. Literature review

The following review of scientific papers and articles presents similar solutions and problems related to the subject of this study. This section also provides information about possible applications of non-relational databases.

In article [2], the authors compared the performance and scalability of three non-relational, document-oriented database systems: Couchbase, CouchDB, and MongoDB. The study discusses the differences between these systems and relational databases, as well as their main characteristics. The authors identified CouchDB as the most efficient in terms of execution time, while also noting that MongoDB achieved the best execution times when the scanning overhead was not considered.

In study [3], the authors compared selected databases in terms of consistency, which is one of the components of the CAP theorem (*Consistency, Availability, Partition Tolerance*) [4]. They analyzed databases from various categories: a key–value database (*Redis*), a column-oriented database (Cassandra), a document database (*MongoDB*), a graph database (*Neo4j*), and a multimodel database (*OrientDB*). In the summary, after completing the analysis, MongoDB was identified as the most consistent database. The authors of the paper [5] also compared both relational and non-relational database systems using the CAP theorem as the evaluation framework.

For the preparation of appropriate test databases in this study, article [6] proved to be particularly useful. The authors of that paper discussed the challenges associated with NoSQL database systems, such as heterogeneity, lack of a predefined schema, and the need for data

modeling. They analyzed existing NoSQL systems and their characteristics, emphasizing the importance of model-based approaches in the design and development of such systems. In response to these challenges, they proposed a high-level data model for NoSQL databases, called NoAM, and a system-independent design methodology. The authors also highlighted the role of modeling in conveying key information throughout various stages of the application lifecycle from requirements analysis to maintenance. Finally, they stressed the importance of modeling as a tool supporting performance management and communication between development teams.

In article [7], the authors analyzed various database systems, relational, non-relational, and hybrid in the context of their use in blockchain technology. They demonstrated that most databases used in blockchain implementations are non-relational, due to their ability to achieve decentralization and high availability. The study also emphasized the need for further development of hybrid solutions that combine the features of SQL and NoSQL systems to provide better support for blockchain technology.

The above-mentioned scientific papers present different approaches to database comparison and broaden the understanding of non-relational database design and development. Comparative analyses of various database systems often focus on performance, which is a crucial aspect in application development. However, the reviewed literature did not reveal studies directly comparing the performance of Amazon DynamoDB with other non-relational databases. This observation served as one of the motivations for selecting the specific database systems analyzed in this study. Furthermore, the literature review was valuable in designing the research plan and presenting the results.

3. Overview of database systems

3.1. Amazon DynamoDB

Amazon DynamoDB [8] is a fully managed NoSQL database service developed by Amazon. The main data structure in DynamoDB consists of tables that contain items. Each item has a primary key, which can be either a partition key or a combination of a partition key and a sort key. Additionally, the primary key must be unique for every item in a table. Items in DynamoDB are stored in JSON (*JavaScript Object Notation*) format. DynamoDB supports various data types such as numbers, strings, and binary data, as well as complex types including lists, maps, and sets. It also allows for data nesting and the use of references to other items. Users can choose between a document-based model and a key-value model, depending on their specific needs. DynamoDB is a fully managed system provided by Amazon Web Services (*AWS*), which means that users do not have to handle server management, hardware configuration, or infrastructure scaling. The service automatically scales its resources to meet changing application demands. Amazon also provides several tools for managing and monitoring DynamoDB, including NoSQL Workbench [9], AWS Command Line Interface (*CLI*), and DynamoDB Console.

3.2. MongoDB

MongoDB [10] is a NoSQL database system developed by MongoDB Inc. Similar to Amazon DynamoDB, MongoDB is designed for storing, processing, and managing large volumes of data, offering both scalability and flexibility. In MongoDB, the main data structure consists of collections, which contain documents. Each document is stored in JSON format. Documents within a collection may have different schemas, meaning they do not have to contain the same fields or structures. Every document in MongoDB has a unique identifier called *_id*, stored as an ObjectId type, which serves as the primary key. MongoDB automatically generates unique identifiers for new documents if they are not provided by the user. MongoDB can be deployed and managed either on-premises or in the cloud. MongoDB Inc. offers a fully managed cloud service called MongoDB Atlas [11] and a client application called MongoDB Compass [12], which enables users to browse data, visualize structures and test queries.

4. Test environment and research scenarios

4.1. Database Schema

Before starting the experiments, three data collections have been created: *Movies*, *Reviews*, and *Users*. Each collection contained 10000 records generated using the Mockaroo [13] service, in such a way that the inserted data reflected real information related to movies, reviews, and users. As a result, each database contained collections with the same number of records and identical data content. The prepared data structures are presented in Tables 1, 2, and 3.

Table 1: Structure of the *Movies* collection

Field	Type	Description
movie_id	Integer	Unique movie identifier
title	String	Movie title
gernerlist	Array[Object]	List of movie genres in the form of objects with the *genre* field
release_year	Integer	Movie release year
director	String	Director
cast	Array[Object]	List of cast members, each object containing the fields *actor_name* and *role*
duration_minutes	Integer	Duration in minutes
language	String	Movie language
rating	Double	Average movie rating
reviews	Array[Object]	List of reviews, each object containing the fields *review_id*, *user_id*, *rating*, *review_text*, *review_date*

Table 2: Structure of the *Reviews* collection

Field	Type	Description
review_id	Integer	Unique review identifier
user_id	Integer	User identifier
movie_id	Integer	Movie identifier
rating	Integer	Movie rating (1–10)
review_text	String	Review content
review_date	Date	Review date

Table 3: Structure of the *Users* collection

Field	Type	Description
user_id	Integer	Unique user identifier
user_name	String	Username
email	String	Email address
age	Integer	User age
join_date	Date	Registration date
favorite_genre	String	Favorite movie genre
reviews	Array[Object]	List of reviews, each object containing the fields review_id, movie_id, rating, review_text, review_date

4.2. Research scenarios

To compare the performance of the two selected database systems, five research scenarios have been prepared. Each scenario represented a different type of database operation commonly performed in applications using non-relational databases. For databases created locally on a computer, all scenarios were tested on datasets of different sizes: 10, 100, 1000, 3000, and 10000 records. In the case of databases created in the cloud, for scenarios 1, 2, and 5, the tests were conducted on smaller datasets: 10, 100, and 1000 records. For each test, the execution time of individual operations was measured. This approach made it possible to evaluate how the systems respond to different levels of workload. Each scenario was repeated 10 times for both database systems, locally and in the cloud. The results were averaged to provide more accurate comparisons and eliminate random deviations. Additionally, before each test execution, the cache memory was cleared to avoid the influence of previously loaded data. The following subsections describe the detailed test scenarios:

4.2.1. Scenario 1: Data insertion

The purpose of this scenario is to evaluate the performance of both systems when inserting new data into the database. This operation is crucial in applications where users regularly add new data.
Test procedure:

- operation description: Inserting new records into the *Reviews* collection,
- repetitions: 10 times,
- number of processed records in local databases: 10, 100, 1000, 3000, 10000,
- number of processed records in cloud databases: 10, 100, 1000.

4.2.2. Scenario 2: Data deletion

This scenario evaluates the performance of data deletion operations, which are important in applications where users can remove information, such as reviews or user accounts.
Test procedure:

- operation description: Deleting records from the *Reviews* collection,
- repetitions: 10 times,
- number of processed records in local databases: 10, 100, 1000, 3000, 10000,
- number of processed records in cloud databases: 10, 100, 1000.

4.2.3. Scenario 3: Data reading with a limit and reading all data without conditions

This scenario examines the performance of read operations, both limited reads and full data retrieval without additional filtering. Such operations are useful in applications that frequently fetch entire datasets or their subsets, for example, when a user browses a single page containing 100 available movies or all reviews.
Test procedure:

- operation description: Retrieving records with a specified limit and retrieving all records from the *Movies* collection without additional conditions.
- repetitions: 10 times,
- number of processed records: 10, 100, 1000, 3000, 10000.

4.2.4. Scenario 4: Data reading with conditions

The purpose of this scenario is to test the performance of conditional data reads, which is particularly important for applications that need to filter data based on various criteria, for example, searching for movies within a specific genre and sorting them by user ratings.
Test procedure:

- operation description:
  - filtering by genre and sorting by rating – retrieving records from the *Movies* collection where the field *genrelist* contains the genre “Horror” and the results are sorted by the highest user ratings,
  - filtering by a nested array – retrieving records from the *Movies* collection where the *cast* array includes an actor with a specified name, for example, “Brad Pitt”,
  - filtering by a nested field – retrieving records from the *Movies* collection where at least one review contains a rating higher than 7,
  - filtering using regular expressions – retrieving records from the *Movies* collection where the movie title begins with the letter “S” or contains the substring “Night”,
  - filtering with combined logical conditions – retrieving records from the *Movies* collection where the *genrelist* field contains the genres “Comedy” or “Sci-Fi” and the user rating is greater than 8,

- repetitions: 10 times,
- number of processed records: The resulting dataset size depends on the applied filter.

4.2.5. Scenario 5: Data updating

The purpose of this scenario is to evaluate the performance of both systems during data modification operations. Update operations are crucial in applications where information changes over time, such as editing user details or updating reviews.
Test procedure:

- operation description: Updating records in the *Users* collection by modifying the *age* field to a random value between 18 and 60,
- repetitions: 10 times,
- number of processed records in local databases: 10, 100, 1000, 3000, 10000,
- number of processed records in cloud databases: 10, 100, 1000.

4.3. Research environment

The experiments were conducted on an Acer Aspire A515-51G laptop with the following specifications:

- processor: Intel Core i5-8250U 1.6GHz,
- graphics card: NVIDIA GeForce MX130 2GB VRAM,
- RAM: 8GB DDR4,
- SSD: 128 GB,
- operating system: Windows 10 Home.

For cloud-based database tests, the Amazon Web Services (*AWS*) platform was used, operating in the *eu-north-1* region. In this environment, both Amazon DynamoDB tables and the MongoDB Atlas managed cluster were tested. Additionally, the correctness of each scenario was verified using MongoDB Compass and NoSQL Workbench tools.

4.4. Test execution

The performance tests were carried out using the free tool Apache JMeter [14], version 5.6.3. This software is designed for advanced performance testing and is well-suited for web applications, databases, web services, and servers. JMeter enables large-scale query generation, simulating load through concurrent operations. Thanks to its support for multiple protocols and the availability of dedicated plugins, JMeter is highly effective for conducting load tests on selected database systems. The tool allows for configuring various research scenarios that simulate interaction with the database, replicating real usage conditions.

Additionally, JMeter offers extensive monitoring capabilities for test results, enabling the collection of the following statistical metrics: average, minimum, maximum, and standard deviation values.

5. Research results

The obtained research results are presented using bar charts for each scenario. These visualizations make it possible to analyze differences in the average execution times of the selected operations.

5.1. Results for scenario 1

Figure 1 presents the average time required to insert records into the MongoDB and DynamoDB databases running locally on the computer.

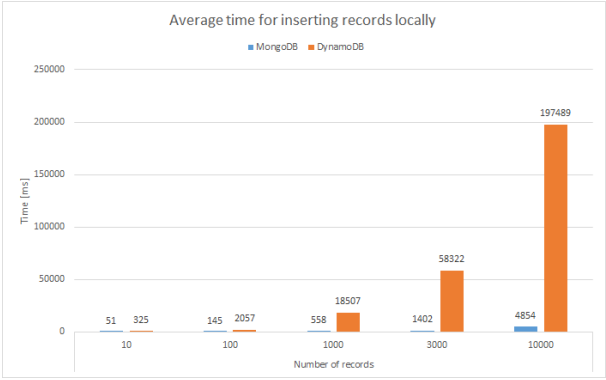

Figure 1: Chart of the average time for inserting records locally into the *Reviews* collection

The data from Figure 1 indicate a significant performance difference between the analyzed database systems in the local environment. When inserting 10 records into DynamoDB, the operations took approximately six times longer than in MongoDB, while for 10000 records, the operation time was about forty times longer.

Figure 2 presents the average time required to insert records into the MongoDB and DynamoDB databases hosted in the cloud.

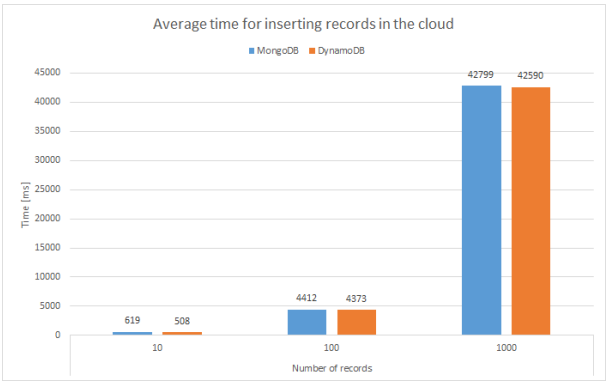

Figure 2: Chart of the average time for inserting records in the cloud into the *Reviews* collection

The data from Figure 2 indicate a comparable performance between the tested database systems in the cloud environment. When inserting 10 records, the average execution time for MongoDB was approximately 22% longer than for DynamoDB. In the tests involving 100 and 1000 records, the differences were marginal, though DynamoDB still demonstrated slightly faster operation times.

5.2. Results for scenario 2

Figure 3 presents the average time required to delete records from the MongoDB and DynamoDB databases running locally on the computer.

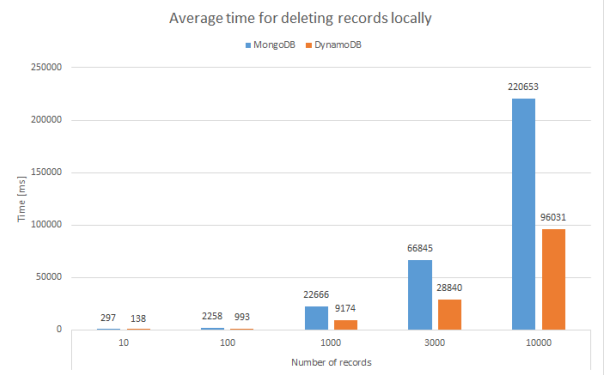

Figure 3: Chart of the average time for deleting records locally from the *Reviews* collection

The data from Figure 3 show a clear performance advantage of DynamoDB over MongoDB in the local environment. When deleting 10 records, MongoDB required more than twice as much time as DynamoDB. A similar difference was observed for 100, 1000, 3000, and 10000 records.

Figure 4 presents the average time required to delete records from the MongoDB and DynamoDB databases hosted in the cloud.

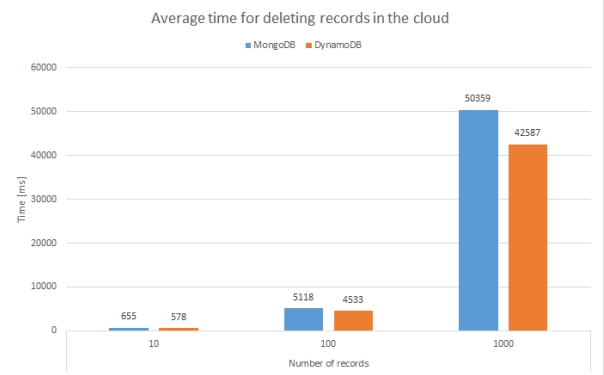

Figure 4: Chart of the average time for deleting records in the cloud from the *Reviews* collection

The data from Figure 4 indicate a slight performance advantage of DynamoDB over MongoDB in the cloud environment. When deleting 10 and 100 records, the difference between the database systems was minimal. However, for 1000 records, the operation time for MongoDB was approximately 18% longer.

5.3. Results for scenario 3

Figure 5 presents the average time required to read records with a limit and without conditions from the MongoDB and DynamoDB databases running locally on the computer.

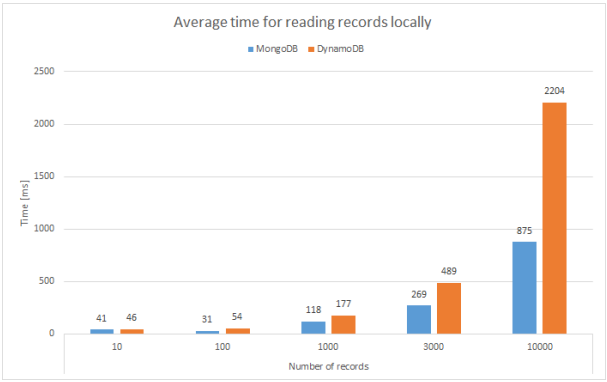

Figure 5: Chart of the average time for reading records locally from the *Movies* collection

The data from Figure 5 indicate a performance advantage of MongoDB over DynamoDB in the local environment. When reading 10 records, the difference was marginal, but as the number of retrieved records increased, the difference also grew. The largest discrepancy was observed for 10000 records, where the average data retrieval time in DynamoDB was approximately 2.5 times longer than in MongoDB.

Figure 6 presents the average time required to read records with a limit and without conditions from the MongoDB and DynamoDB databases hosted in the cloud.

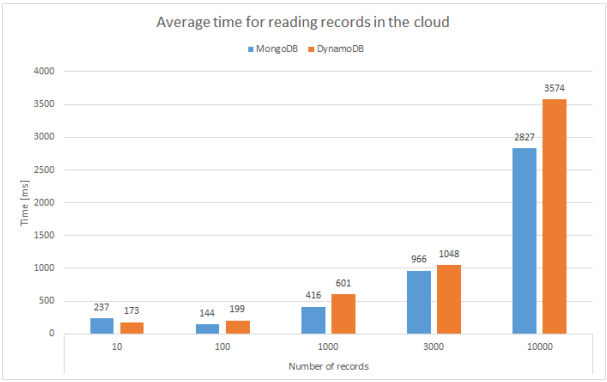

Figure 6: Chart of the average time for reading records in the cloud from the *Movies* collection

The data from Figure 6 reveal different performance patterns compared to the local environment. When retrieving 10 records, the operation in MongoDB took about 40% longer than in DynamoDB. However, as the number of records increased to 100, MongoDB's performance improved. For datasets containing 100, 1000, 3000, and 10000 records, MongoDB demonstrated better time efficiency than DynamoDB.

5.4. Results for scenario 4

Figure 7 presents the average time required to read records with conditions from the MongoDB and DynamoDB databases running locally on the computer.

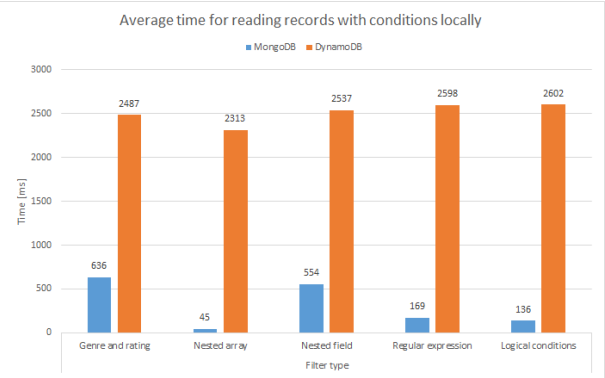

Figure 7: Chart of the average time for reading records with conditions locally from the *Movies* collection

The data from Figure 7 indicate a significant performance advantage of MongoDB over DynamoDB in all conditional read operations. For filtering by genre combined with sorting by rating, DynamoDB took about four times longer than MongoDB. The largest difference occurred in filtering by a nested array, where MongoDB retrieved records approximately 51 times faster than DynamoDB. For filtering by a nested field, the difference was about 4.6 times, while for filtering using regular expressions, it was about 15 times. In the case of filtering with logical conditions, MongoDB performed about 19 times faster than DynamoDB.

Figure 8 presents the average time required to read records with conditions from the MongoDB and DynamoDB databases hosted in the cloud.

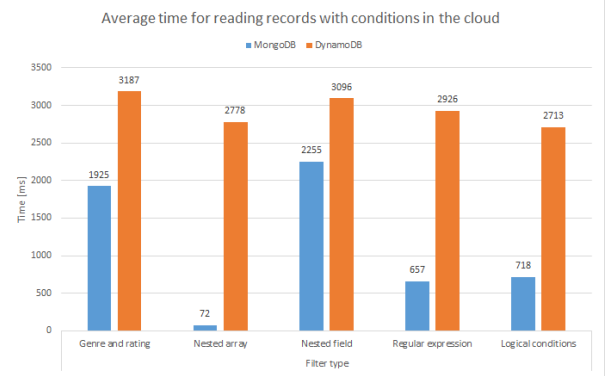

Figure 8: Chart of the average time for reading records with conditions in the cloud from the *Movies* collection

The data from Figure 8 also demonstrate MongoDB's superior performance over DynamoDB in all conditional read operations. For filtering by genre combined with sorting by rating, DynamoDB's execution time was about 1.7 times longer than MongoDB's. The largest performance gap was observed in filtering by a nested array, where MongoDB retrieved records approximately 39 times faster than DynamoDB. For filtering by a nested field, the difference was around 1.4 times, for filtering using regular expressions about 4.5 times, and for filtering with logical conditions about 3.8 times.

5.5. Results for scenario 5

Figure 9 presents the average time required to update records in the MongoDB and DynamoDB databases running locally on the computer.

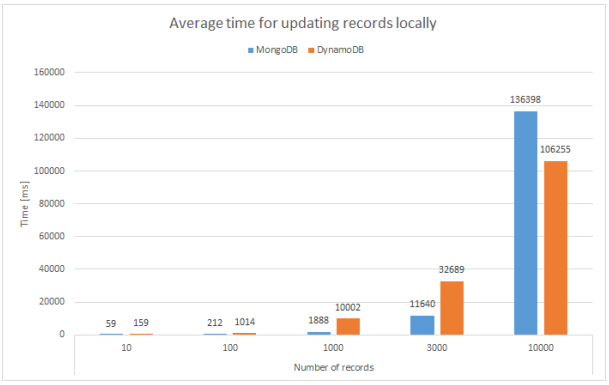

Figure 9: Chart of the average time for updating records locally in the *Users* collection

The data from Figure 9 show a significant performance advantage of MongoDB over DynamoDB in update operations for smaller datasets in the local environment. However, as the dataset size increased to 10000 records, DynamoDB became faster.

Figure 10 presents the average time required to update records in the MongoDB and DynamoDB databases hosted in the cloud.

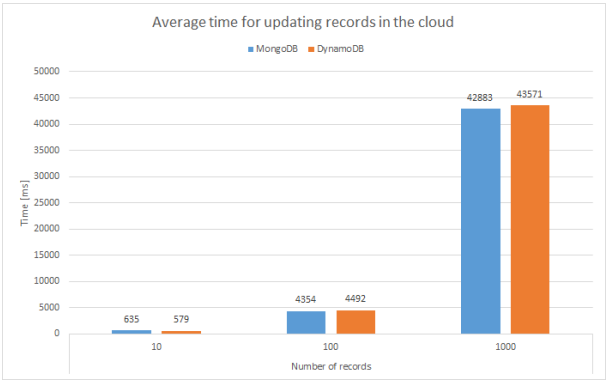

Figure 10: Chart of the average time for updating records in the cloud in the *Users* collection

The data from Figure 10 indicate a similar performance level for both database systems in the cloud environment. When updating 10 records, DynamoDB was 9% faster than MongoDB. For 100 and 1000 records, the operations were marginally faster in MongoDB.

5.6. Summary of results

This is the summary of the research conducted to compare the performance of Amazon DynamoDB and MongoDB. Table 4 summarizes the performance comparison results for each operation.

Table 4: Summary of database performance comparison for individual operations in local and cloud environments

Operation	Environment	More efficient database
Record insertion	Local	MongoDB
Record deletion	Local	DynamoDB
Read without conditions	Local	MongoDB
Read with conditions	Local	MongoDB
Record update	Local	MongoDB
Record insertion	Cloud	DynamoDB
Record deletion	Cloud	DynamoDB
Read without conditions	Cloud	MongoDB
Read with conditions	Cloud	MongoDB
Record update	Cloud	MongoDB

6. Conclusions

Based on the conducted performance tests of MongoDB and DynamoDB, the following conclusions can be drawn. The environment in which the test scenarios were executed had a significant impact on the final results. In the local environment, MongoDB demonstrated a clear performance advantage, except for the operations involving record deletion.

In the cloud environment, the performance differences were smaller. The results were still generally better for MongoDB, but only in the cases of data reading (both with and without conditions) and record updates. For data insertion and deletion, there was only a marginal difference between the examined environments, with DynamoDB performing slightly faster. Additionally, the research results revealed a significant advantage of MongoDB over DynamoDB in terms of query optimization. In both the local and cloud environments, conditional data retrieval was completed considerably faster in MongoDB.

In conclusion, the hypothesis stated at the beginning of the article, “MongoDB is a more efficient database than DynamoDB in terms of the time required for inserting, retrieving, updating, and deleting data.” has been confirmed in most cases.

References

[1] DB-Engines Ranking, https://db-engines.com/en/ranking, [09.06.2025].

[2] I. Carvalho, F. Sá, J. Bernardino, Performance Evaluation of NoSQL Document Databases: Couchbase, CouchDB, and MongoDB, Algorithms 16 (2023) 1-17, https://doi.org/10.3390/a16020078.

[3] M. Diogo, B. Cabral, J. Bernardino, Consistency Models of NoSQL Databases, Future Internet 11 (2019) 1-19, https://doi.org/10.3390/fi11020043.

[4] S. Gilbert, N. Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, SIGACT News Volume 33(2) (2002) 51-59, https://doi.org/10.1145/564585.564601.

[5] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, B. Luo, SQL and NoSQL Database Software Architecture Performance Analysis and Assessments - A Systematic Literature Review, Big Data and Cognitive Computing 7 (2023) 1-44, https://doi.org/10.3390/bdcc7020097.

[6] P. Atzeni, F. Bugiotti, L. Cabibbo, R. Torlone, Data modeling in the NoSQL world, Computer Standards & Interfaces Volume 67 (2020) 1-36, https://doi.org/10.1016/j.csi.2016.10.003.

[7] J. Kalajdjieski, M. Raikwar, N. Arsov, G. Velinov, D. Gligoroski, Databases fit for blockchain technology: A complete overview, Blockchain: Research and Applications Volume 4 (2023) 1-18, https://doi.org/10.1016/j.bcra.2022.100116.

[8] What is Amazon DynamoDB? – Amazon DynamoDB documentation, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html, [02.06.2025].

[9] NoSQL Workbench for DynamoDB - Amazon, https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/workbench.html, [02.06.2025].

[10] What is MongoDB? – MongoDB documentation, https://www.mongodb.com/docs/manual/, [02.06.2025].

[11] What is MongoDB Atlas? – Atlas – MongoDB, https://www.mongodb.com/docs/atlas, [02.06.2025].

[12] What is MongoDB Compass – Compass, https://www.mongodb.com/docs/compass, [02.06.2025].

[13] Mockaroo, https://www.mockaroo.com, [03.06.2025].

[14] Apache JMeter: Narzędzie do zaawansowanego testowania wydajności, https://boringowl.io/blog/apache-jmeter-narzedzie-do-zaawansowanego-testowania-wydajnosci, [03.06.2025].