A Comparative Evaluation of PostgreSQL, MongoDB, and MySQL for Ethiopian Migrant Data Management

Tariq Emad Ali[1], Faten Imad Ali[2], Saker Hekmat[4], Farid Eyvazov[4], Pavle Dakić[3] and Alwahab Dhulfiqar Zoltan[4,5]

[1]Department of Information and Communication Engineering, Al Khwarizmi College of Engineering, University of Baghdad, Baghdad, Iraq
[2]Department of Biomedical Engineering, College of Engineering, AL-Nahrain University, Baghdad, Iraq
[3]Faculty of Informatics and Computing, Singidunum University, Danijelova 32, 11000 Belgrade, Serbia
[4]Faculty of Informatics, Eotvos Lorand University, Pázmány Péter stny. 1/C, 1117 Budapest, Hungary
[5]John von Neumann Faculty of Informatics, Óbuda University, Bécsi út 96/B, 1034 Budapest, Hungary

Abstract

In the face of rising international migration, dealing with large-scale, established migrant data is a monumental task. This article discusses the Ethiopian Migrant Database Management System (EMDMS) as a real-world software that necessitates effective, scalable, and dependable statistics garage solutions. The goal of this study is to undertake a comprehensive comparison of three database control structures (PostgreSQL, MongoDB, and MySQL) in the EMDMS environment to verify their suitability for dealing with migration-related data. Using an AWS cloud-based deployment and containerized microservices architecture, the test links each DBMS with real-time APIs via Hasura and collects performance data via Prometheus and Grafana. Insert, pick out, and delete are benchmark procedures that are measured on datasets ranging from 100 to 100,000 rows in size. PostgreSQL outperformed other databases in terms of managing large amounts of data, particularly in insert, select, and delete operations. MongoDB offered schema flexibility but confirmed decreased performance at higher volumes. MySQL performed well with smaller datasets but lagged in scalability. PostgreSQL emerged as the most suitable DBMS for dealing with existing migration records within EMDMS, providing both overall performance and operational reliability. MongoDB is ideal for flexible information models, but MySQL is better suited to simple, lightweight applications.

Keywords

PostgreSQL, MongoDB, MySQL, Migration Data, Database Benchmarking, GraphQL, EMDMS, Docker, Prometheus, Grafana

1. Introduction

Efficient management of migrant records is a critical need in an increasingly globalized world and having good communication channels [1, 2]. This particularly for countries like Ethiopia, where migration patterns are shaped by complex economic, social, and political factors [3, 4]. Both governmental and non-governmental organizations require robust database systems capable of handling complex, large-scale, and structured datasets to support effective policy-making and service delivery [5, 6].

To address this challenge, the Ethiopian Migrant Database Management System (EMDMS) was developed as a real-world platform for documenting, analyzing, and managing migration data. The system integrates advanced database technologies to support critical functions such as data collection, processing, patterns, CI/CD, and reporting [7, 8, 9]. Given the wide range of available database man-

SQAMIA 2025: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 10–12, 2025, Maribor, Slovenia

*Corresponding author.
†These authors contributed equally.
tariqemad@kecbu.uobaghdad.edu.iq (T. E. Ali); fatenemadali@gmail.com (F. I. Ali); enghekmatsaker2018@gmail.com (S. Hekmat); farid.eyvazov@inf.elte.hu (F. Eyvazov); pdakic@singidunum.ac.rs (P. Dakić); alwahab.zoltan@nik.uni-obuda.hu (A. D. Zoltan)
0000-0003-0505-8669 (T. E. Ali); 0000-0002-1767-8825 (F. I. Ali); 0000-0003-3538-6284 (P. Dakić); 0000-0002-7893-6250 (A. D. Zoltan)

agement systems (DBMSs), selecting the most suitable option for such a mission-critical application requires a systematic and comprehensive performance evaluation [10, 11].

In this context, software performance metrics play a vital role in evaluating and comparing database systems. By collecting and analyzing quantitative indicators—such as throughput, latency, resource utilization, and error rates—this study enables objective benchmarking. The goal is to provide practical, data-driven insights into the performance, scalability, and reliability of different DBMSs under realistic deployment conditions, ultimately guiding the selection of optimal data management solutions for migration-related applications.

The main contribution of this paper is a unique, comprehensive empirical benchmarking and evaluation of PostgreSQL, MongoDB, and MySQL for large-scale Ethiopian migrant data management within a real-world, cloud-native environment. This work provides actionable, evidence-based insights for database selection in governmental and institutional contexts, distinguishing itself by embedding each DBMS into a fully deployed system with real-time data flows, user interfaces, and monitoring tools—unlike prior studies that focus on isolated or synthetic environments.

The paper is organized as follows: Section 3 describes the materials and methods, including the experimental setup and benchmarking approach. Section 4 presents the results of the performance evaluation. Section 5 discusses the findings in relation to the research questions. Section 6 concludes the paper and outlines directions for future work.

2. Literature Review

Numerous studies had been carried out on the performance assessment of database control structures. Makris et al. [12] compared MongoDB and PostgreSQL in the context of spatial queries and concluded PostgreSQL had superior indexing and performance for established queries. Additionally, the adoption of DataOps practices is increasingly important for ensuring efficient, automated, and reliable data workflows in modern database management environments.

Similarly, Li & Manoharan [13] evaluated SQL and NoSQL databases under various workloads and found PostgreSQL superior for ACID-compliant tasks, while MongoDB offered flexibility in handling unstructured data. Keshavarz [14] provided a detailed comparison between MySQL and MongoDB, showing MongoDB's performance benefits in insert-heavy operations. Taipalus [15] emphasized the significance of evaluating DBMS overall performance in real-global, in preference to synthetic, environments. Battle et al. [16] and Jansson & Johansson [17] explored application-based benchmarking, arguing for tests that mimic production-level workloads.

Bernstein and Elmore et al. [18, 19] investigated scalability challenges in distributed databases, while Abadi and Stonebraker [20, 11] discussed data modeling techniques for both relational and NoSQL systems. Haerder & Reuter [21] and Özsu & Valduriez [22] provided foundational insights into transaction management and consistency. Further, Cattell [23] reviewed scalable data store architectures.

3. Material and methods

This study employs an empirical benchmarking methodology, focusing on quantitative performance evaluation of PostgreSQL, MongoDB, and MySQL within the EMDMS context. The approach is experimental and data-driven, using controlled deployments and standardized workloads to ensure objectivity and reproducibility.

Two research questions guide this work:

- **RQ1:** Which database management system (PostgreSQL, MongoDB, or MySQL) delivers the best operational performance (insert, select, delete) for large-scale Ethiopian migrant data in a cloud-native environment?
- **RQ2:** How do these DBMSs compare in terms of scalability and resource efficiency as dataset size increases?

All three DBMSs were deployed in isolated Docker containers on AWS EC2 (t2.medium, Ubuntu 22.04 LTS) to guarantee consistent environments. The architecture included Hasura GraphQL Engine for real-time APIs [24], Node.js (Express.js) for business logic, and Prometheus with Grafana for monitoring.

A synthetic dataset, simulating Ethiopian migrant records, was generated and scaled from 100 to 100,000 rows. Identical data structures were used across all DBMSs. Benchmarking focused on bulk insert, select, and delete operations, executed via standardized scripts and Hasura APIs to ensure uniformity and minimize bias.

Performance metrics—operation throughput, CPU, memory, and system load—were collected using Prometheus exporters and visualized in Grafana. Each test was repeated multiple times, reporting average values. All scripts and configurations were containerized and orchestrated with Docker Compose, and the environment was reset between tests to ensure validity and reproducibility. This approach enables transparent, repeatable results and supports future research extensions.

4. Results

This section presents the benchmarking results for PostgreSQL, MongoDB, and MySQL within the Ethiopian Migrant Database Management System (EMDMS). Performance was evaluated based on three primary database operations: insert, select, and delete, across four dataset sizes—100, 1,000, 10,000, and 100,000 rows. All metrics were measured in operations per second (ops/sec) and visualized using Prometheus and Grafana.

4.1. System Architecture

The EMDMS architecture, illustrated in Figure 1, is composed of four key areas with the following key components:

- **Hasura GraphQL Engine:** Automatically generates real-time GraphQL APIs for PostgreSQL, MongoDB, and MySQL, enabling efficient data access.
- **Node.js (Express.js):** Handles business logic, including user authentication, authorization, and custom API endpoints.
- **Databases:** The system integrates three database platforms under identical schema and workload conditions to ensure fair benchmarking:
  - **PostgreSQL** is a relational, ACID-compliant database ideal for structured data and complex queries. Integrated with Hasura, it provides real-time GraphQL functionality and advanced indexing features (e.g., PostGIS for spatial data).
  - **MongoDB** is a document-oriented NoSQL database known for schema flexibility and horizontal scalability. While typically optimized for unstructured data, it is configured here to manage structured datasets for performance evaluation.
  - **MySQL** is a lightweight relational DBMS widely used for web applications. Although not as feature-rich as PostgreSQL, it performs well under lighter workloads and benefits from broad community support and compatibility.
- **Docker Compose:** Orchestrates deployment of all services to ensure isolated, repeatable environments for each test case.
- **Prometheus and Grafana:** Used for real-time collection and visualization of performance metrics, including CPU, memory, and operation throughput.

Each DBMS is deployed in an isolated Docker container and instrumented with exporters to ensure accurate collection of performance metrics. This configuration allows for consistent benchmarking of insert, select, and delete operations across increasing dataset sizes (from 100 to 100,000 rows). The architecture promotes reproducibility and scalability and reflects a real-world application environment hosted on AWS infrastructure.

Wireless patient monitoring systems, such as those described by Ali et al. [25], highlight the importance of real-time data management in critical applications—further motivating the need for performance and reliable database solutions such as those evaluated in this study.

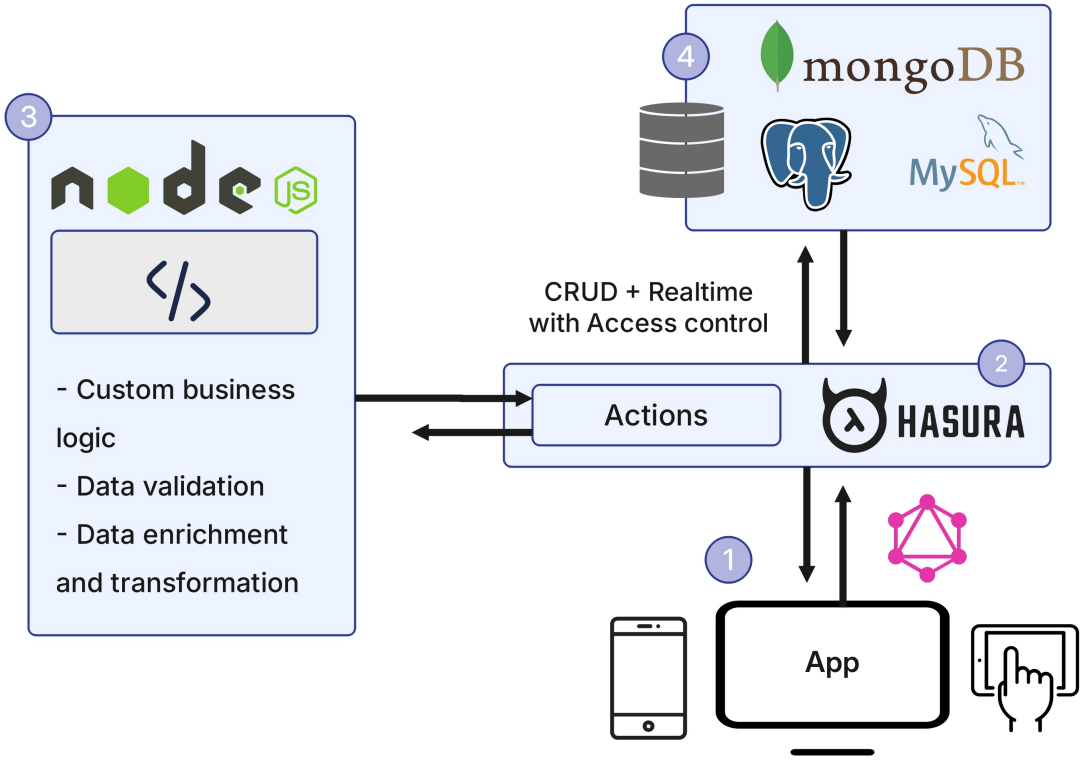

Figure 1: EMDMS System Architecture. Source: author's contribution.

4.2. Experimental Setup

To ensure a realistic and reproducible environment for benchmarking, the whole device was deployed on an Amazon Web Services (AWS) EC2 example of type t2. Medium, going for walks Ubuntu 22.04 LTS. This configuration presents a balance of computational electricity and scalability suitable for evaluating cloud-native packages. Structured take a look at facts representing migrant statistics changed into generated and uniformly applied across all three DBMSs—PostgreSQL, MongoDB, and MySQL. Each database was hosted in a remoted Docker container, ensuring resource separation and consistency in the course of benchmarking. Operational workloads consisted of trendy CRUD operations, specifically bulk insertions, choice queries, and deletion operations. These operations were completed on datasets of increasing sizes: a hundred, 1,000, 10,000, and 100,000 rows. This gradient allowed for distinct commentary of scalability and machine behavior under load.

Performance metrics were collected (Table 1) using Prometheus Node Exporter and visualized through Grafana dashboards (Figure 2).

Metrics such as operation throughput (operations per second), system load, memory usage, and CPU utilization were monitored in real time. The use of containerized environments, combined with consistent deployment practices via Docker Compose, ensured the repeatability and validity of the performance tests.

Table 1
Monitored Performance Metrics and Their Units

Metric	Unit of Measure
Operation Throughput	Operations per second (ops/sec)
Response Time	Milliseconds (ms)
System Load	Load average (no units)
Memory Usage	Megabytes (MB)
CPU Utilization	Percentage (%)

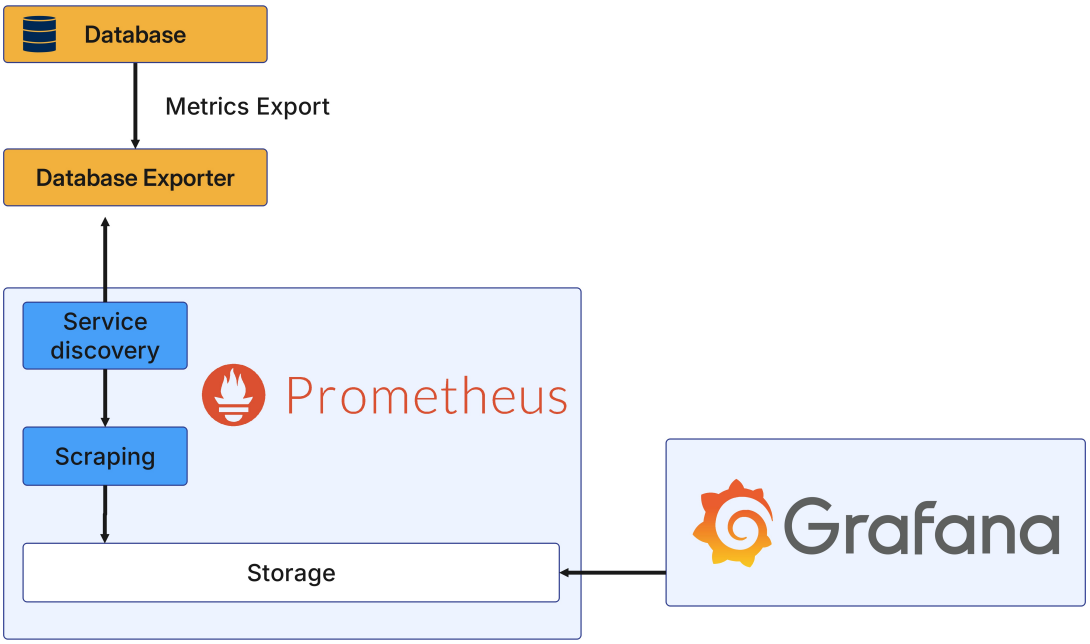

Figure 2: Metrics collection and visualization workflow using Prometheus and Grafana integrated with database exporters. Source: author's contribution.

4.3. Dataset Structure

To ensure consistency across the tested database systems, a synthetic dataset representing migrant information was generated with a uniform schema. This dataset simulates realistic data attributes relevant to migration management, including personal, demographic, and migration-related fields.

Table 2 presents the structure of the dataset used in benchmarking. Each database (PostgreSQL, MongoDB, and MySQL) was populated with the same records and field types, adjusted as needed for each system's schema requirements.

Table 2
Schema of the Synthetic Migrant Dataset

Field Name	Data Type	Description
`id`	Integer (Primary Key)	Unique identifier for each migrant
`first_name`	String (VARCHAR/UTF-8)	Migrant's first name
`last_name`	String (VARCHAR/UTF-8)	Migrant's last name
`gender`	String (ENUM)	Gender identity (e.g., Male, Female, Other)
`age`	Integer	Age in years
`origin_country`	String	Country of origin
`destination`	String	Intended destination country or region
`migration_date`	Date (ISO 8601)	Date of migration
`status`	String (ENUM)	Migration status (e.g., Registered, In Transit, Returned)
`created_at`	Timestamp	Record creation timestamp
`updated_at`	Timestamp	Record last update timestamp

Example Record:

```
{
  "id": 1023,
  "first_name": "Amanuel",
  "last_name": "Bekele",
  "gender": "Male",
  "age": 29,
  "origin_country": "Ethiopia",
  "destination": "Saudi Arabia",
  "migration_date": "2022-05-10",
  "status": "Registered",
  "created_at": "2022-05-10T12:30:45Z",
  "updated_at": "2022-05-10T12:30:45Z"
}
```

Each benchmark test was performed using dataset sizes of 100, 1,000, 10,000, and 100,000 rows, with randomly generated values for all fields to simulate production-scale usage while ensuring consistency across tests.

4.4. System Environment Control and Test Consistency

To ensure that performance measurements accurately reflect the behavior of each database system, several measures were taken to eliminate system-related interference:

- **Isolated Deployment:** Each DBMS was deployed in a separate Docker container on the same AWS EC2 instance (t2.medium, Ubuntu 22.04 LTS). Containers were configured with limited resource access to prevent contention, and no background processes were running on the host machine during testing.
- **Consistent Runtime Conditions:** All services were restarted before each batch of tests to clear internal state. This ensured that metrics such as buffer pool contents, system caches, and memory state were reset.
- **Cache Management:** Where applicable, database caches were cleared using DB-specific mechanisms (e.g., `DISCARD ALL` in PostgreSQL) and test scripts included a cold-start and warm-up phase. This allowed both uncached (first-run) and cached (subsequent-run) performance to be observed in a consistent manner.
- **Benchmark Repeatability:** Each test was repeated a minimum of three times under identical conditions. The average value of operation throughput (ops/sec), CPU usage, and memory consumption was recorded. Variance across runs was monitored to ensure stability, and outlier runs were discarded when significant deviation occurred.
- **Monitoring Isolation:** Prometheus and Node Exporter were configured to monitor only the Docker containers, excluding host-level metrics. This ensured that system-level noise (e.g., OS-level background jobs) did not influence the collected results.

These controls ensured that the observed performance differences among PostgreSQL, MongoDB, and MySQL were attributable to the DBMS characteristics and not external environmental factors. The approach increases the reliability and reproducibility of the benchmarking results.

4.5. Insert Performance

Insert operations measured how correctly every DBMS should cope with records ingestion at varying scales. PostgreSQL exhibited advanced scalability and throughput, performing continually throughout all dataset sizes.

MongoDB observed with aggressive overall performance on small to medium datasets but showed a tremendous drop at better volumes. MySQL's performance changed into corresponding to PostgreSQL on smaller datasets however declined sharply because the dataset grew. Table 3 illustrates the comparative insert performance across all database systems.

Table 3
Insert Operation Performance Across Dataset Sizes

Dataset Size	PostgreSQL		MongoDB		MySQL	
	Throughput (ops/sec)	Resp. Time (ms)	Throughput (ops/sec)	Resp. Time (ms)	Throughput (ops/sec)	Resp. Time (ms)
100	5	200	6	170	4	250
1,000	15	180	16	160	10	210
10,000	130	150	120	170	80	220
100,000	450	100	400	140	250	180

4.6. Select Performance

Read efficiency changed into testing the use of select queries. PostgreSQL once more led in performance, specifically on datasets exceeding 10,000 rows. MongoDB outperformed others on small datasets due to its bendy file version but struggled to keep performance at large scales. MySQL showed applicable performance to begin with but suffered widespread throughput degradation as dataset size extended. Table 4 provides the select operation throughput across dataset sizes and DBMS platforms.

Also, Table 4 compares the performance of SELECT operations in PostgreSQL, MongoDB, and MySQL with dataset sizes ranging from 100 to 100,000 records. It measures both throughput (operations per second) and response time (milliseconds). PostgreSQL has the greatest throughput of 450 ops/sec for 100,000 records, with an 80 ms response time. MongoDB is close behind with 400 ops/sec and 100 ms, while MySQL has the lowest throughput with 250 ops/sec and a response time of 130 ms. As dataset size grows, throughput increases and response times fall across all databases, with PostgreSQL continuously outperforming the others.

Table 4
Select Operation Performance Across Dataset Sizes

Dataset Size	PostgreSQL		MongoDB		MySQL	
	Throughput (ops/sec)	Resp. Time (ms)	Throughput (ops/sec)	Resp. Time (ms)	Throughput (ops/sec)	Resp. Time (ms)
100	5	180	6	160	4	210
1,000	15	140	16	130	10	170
10,000	130	100	120	110	80	150
100,000	450	80	400	100	250	130

4.7. Delete Performance

Delete operations were evaluated to test records elimination efficiency. PostgreSQL continually added excessive performance in spite of the biggest datasets, confirming its robust transaction engine. MongoDB remained aggressive with close to-parallel performance at medium scale. MySQL once more lagged at the back of, especially with datasets over 10,000 rows, reflecting obstacles in bulk delete operations. Table 5 indicates delete operation overall performance across the tested DBMS platforms.

Table 5
Delete Operation Performance Across Dataset Sizes

Dataset Size	PostgreSQL		MongoDB		MySQL	
	Throughput (ops/sec)	Resp. Time (ms)	Throughput (ops/sec)	Resp. Time (ms)	Throughput (ops/sec)	Resp. Time (ms)
100	5	160	6	140	4	190
1,000	20	120	18	130	10	160
10,000	150	100	140	110	100	140
100,000	480	70	420	90	280	110

5. Discussion

The assessment identified PostgreSQL as the most reliable and scalable database for structured data management within the Ethiopian Migrant Database Management System (EMDMS). Its strong performance in insert, select, and delete operations can be attributed to its ACID compliance, efficient indexing mechanisms, and support for complex query operations. Furthermore, PostgreSQL's integration with Hasura for real-time GraphQL APIs enhances both developer productivity and system responsiveness.

MongoDB offered the advantage of flexible schema design and ease of horizontal scaling. While it delivered competitive performance in insert and delete operations, its throughput declined significantly in select operations at higher data volumes. This makes MongoDB less suitable for read-intensive applications dealing with large-scale structured datasets.

MySQL was found to be appropriate for small datasets or applications with modest performance requirements. Its performance degraded noticeably as data size increased, particularly in select and delete operations. Nonetheless, MySQL's simplicity, widespread community support, and mature tooling make it a practical option for low-complexity systems.

Research Question Findings:

RQ1: Which database provides the best operational performance for EMDMS?

PostgreSQL demonstrated the highest overall performance in insert, select, and delete operations across all dataset sizes, making it the most suitable choice for large-scale Ethiopian migrant data management in a cloud-native context. MongoDB performed competitively for smaller datasets and provided schema flexibility; however, its efficiency declined with increasing data volume, especially in read-intensive workloads. MySQL showed acceptable performance for small-scale use but suffered significant performance drops at larger scales.

RQ2: How do the databases scale and manage resources with increasing data volume?

PostgreSQL exhibited superior scalability and resource efficiency, maintaining consistent throughput and manageable resource consumption as dataset size grew. MongoDB, while efficient in insert and delete operations at moderate data volumes, revealed several performance bottlenecks at scale:

- Higher memory usage due to data duplication and less compact document storage.
- Decreased indexing efficiency, especially with compound or nested fields.
- Increased read latency under heavy query loads, likely due to its eventual consistency model and query planning overhead.

These limitations suggest MongoDB is less suitable for structured, read-heavy workloads at scale.

Impact of Caching and Buffer Management:

Database buffer management and internal caching significantly influence performance, especially for repeated operations. To account for this, caching behavior was explicitly controlled. Before each test sequence, databases were either restarted or issued cache-clearing commands (e.g., `DISCARD ALL` in PostgreSQL, `FLUSH TABLES` in MySQL) to simulate cold-cache conditions. Additional warm-up runs were conducted to represent real-world, long-running scenarios.

While complete cache eviction could not be guaranteed—particularly in MongoDB due to background journaling and memory-mapped files—the adopted methodology provides a reasonable approximation of both cold-start and warmed-up performance states. These factors were considered when interpreting performance results, especially for select operations, where caching can significantly improve throughput.

6. Conclusions

In this take a look at, PostgreSQL, MongoDB, and MySQL were in comparison underneath practical deployment conditions to evaluate their suitability for dependent migration facts management in EMDMS.

PostgreSQL tested superior overall performance and scalability across all tested operations, making it the most suitable choice for big-scale programs. MongoDB, at the same time as imparting flexibility

in schema layout, showed performance obstacles under load, particularly with read-in depth operations. MySQL, although capable at lower volumes, didn't maintain efficient operation at scale.

This study opens several avenues for future work where the further research in the future could explore the integration of additional NoSQL and NewSQL databases, such as Cassandra, DynamoDB, or CockroachDB, to assess their suitability for large-scale, heterogeneous migration data.

Investigating hybrid storage models that combine relational and non-relational paradigms may yield performance and flexibility benefits for complex data scenarios. Moreover, extending the benchmarking framework to include real-time analytics, data security, and compliance requirements would provide a more comprehensive evaluation for governmental and institutional use cases. Finally, automating deployment and monitoring pipelines, as well as incorporating machine learning-driven workload optimization, could further enhance the scalability and adaptability of EMDMS in dynamic environments.

Declaration on Generative AI

Either:
The author(s) have not employed any Generative AI tools.

References

[1] F. Eyvazov, T. E. Ali, F. I. Ali, A. D. Zoltan, Beyond containers: Orchestrating microservices with Minikube, Kubernetes, Docker, and Compose for seamless deployment and scalability, in: 2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), IEEE, Noida, India, 2024, pp. 1–6.

[2] P. Dakić, T. Heričko, v. Kljajić, V. Todorović, The economics of ai-powered call center development using chatgpt for the needs of an automotive retail business, in: Proceedings of the International Scientific Conference - Sinteza 2025, Sinteza 2025, Singidunum University, 2025, pp. 382–388. doi:10.15308/sinteza-2025-382-388.

[3] L. Röthlisberger, Ethiopian Female Labor Migration to the Middle East: An Investigation of the System of Irregular Migration, Master's thesis, Alice Salomon Hochschule – University of Applied Sciences, Berlin, Germany, 2019. URL: https://www.researchgate.net/publication/333123456_Ethiopian_Female_Labor_Migration_to_the_Middle_East_An_Investigation_of_the_System_of_Irregular_Migration.

[4] T. E. Ali, F. I. Ali, P. Dakić, A. D. Zoltan, Trends, prospects, challenges, and security in the healthcare Internet of Things, Computing 107 (2025) 28. In press.

[5] G. Adugna, Impact of remittances on home communities in Ethiopia, Available at: https://www.academia.edu/3123456/Impact_of_remittances_on_home_communities_in_Ethiopia, 2012. URL: https://www.academia.edu/3123456/Impact_of_remittances_on_home_communities_in_Ethiopia.

[6] T. E. Ali, F. I. Ali, N. Pataki, A. D. Zoltán, Exploring attribute-based facial synthesis with generative adversarial networks for enhanced patient simulator systems, in: 2024 7th International Conference on Software and System Engineering (ICoSSE), IEEE, Istanbul, Turkey, 2024, pp. 53–60.

[7] T. Golis, P. Dakić, V. Vranić, Creating microservices and using infrastructure as code within the ci/cd for dynamic container creation, in: 2022 IEEE 16th International Scientific Conference on Informatics (Informatics), IEEE, 2022, pp. 114–119. doi:10.1109/informatics57926.2022.10083442.

[8] P. Dakić, Software compliance in various industries using ci/cd, dynamic microservices, and containers 14 (2024). doi:10.1515/comp-2024-0013.

[9] M. D. Haiderzai, P. Dakić, I. Stupavský, M. Aleksić, V. Todorović, Pattern shared vision refinement for enhancing collaboration and decision-making in government software projects 14 (2025) 334. doi:10.3390/electronics14020334.

[10] T. E. Ali, F. I. Ali, B. Kovacs, A. D. Zoltán, Comparison of IoTStack and EdgeX foundry performance for asset tracking, Procedia Computer Science 258 (2025) 3369–3380.
[11] H. Matallah, G. Belalem, K. Bouamrane, Comparative study between the MySQL relational database and the MongoDB NoSQL database, International Journal of Software Science and Computational Intelligence 13 (2021) 1–15. doi:10.4018/IJSSCI.2021070104.
[12] A. Makris, et al., MongoDB vs PostgreSQL: A comparative study on performance aspects, GeoInformatica 25 (2021) 319–343. doi:10.1007/s10707-020-00407-w.
[13] T. E. Ali, F. I. Ali, M. A. Abdala, P. Norbert, M. Tejfel, A. D. Zoltán, Exploring application deployment on edge solutions: A focus on mobile edge computing, Akraino Eliot, EdgeX, and OpenVINO for healthcare applications, in: The International Conference on Recent Innovations in Computing, Springer Nature Singapore, Singapore, 2023, pp. 851–862.
[14] S. Keshavarz, Analyzing performance differences between MySQL and MongoDB, 2021. URL: https://www.researchgate.net/publication/349264964_Analyzing_Performance_Differences_Between_MySQL_and_MongoDB.
[15] T. Taipalus, Database management system performance comparisons: A systematic literature review, Journal of Systems and Software 195 (2023) 111872. doi:10.1016/j.jss.2023.111872.
[16] L. Battle, et al., Database benchmarking for supporting real-time interactive querying of large data, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, ACM, Portland, OR, USA, 2020, pp. 1559–1576. doi:10.1145/3318464.3389732.
[17] M. Jansson, K. Johansson, Performance Benchmarking Using Real-World Applications, Master's thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2012. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A537441&dswid=1234.
[18] A. Quiña-Mera, P. Fernandez, J. M. García, A. Ruiz-Cortés, GraphQL: A systematic mapping study, ACM Computing Surveys 55 (2023) 1–35.
[19] I. Papadhopulli, H. Paci, GraphQL: A comprehensive analysis of its advantages, challenges, and best practices in modern API development, The Eurasia Proceedings of Science Technology Engineering and Mathematics 32 (2024) 230–237.
[20] M. Winand, SQL Performance Explained: Everything Developers Need to Know About SQL Performance, M. Winand, 2012. URL: https://sql-performance-explained.com/.
[21] L. Marrero, et al., Performance analysis in NoSQL databases, relational databases, and NoSQL databases as a service in the cloud, in: Communications in Computer and Information Science, volume 1375, Springer, 2021, pp. 137–150. doi:10.1007/978-3-030-75836-3_11.
[22] T. Capris, et al., Comparison of SQL and NoSQL databases with different workloads: MongoDB vs MySQL evaluation, in: 2022 International Conference on Data Analytics for Business and Industry (ICDABI), IEEE, 2022, pp. 214–218. doi:10.1109/ICDABI56818.2022.10041513.
[23] T. E. Ali, A. H. Morad, M. A. Abdala, Efficient private cloud resources platform, in: 2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), IEEE, Istanbul, Turkey, 2021, pp. 1–6.
[24] Hasura, Hasura vs Apollo: Comparing GraphQL platforms, 2024. URL: https://hasura.io/blog/hasura-vs-apollo-comparing-graphql-platforms.
[25] T. E. Ali, F. I. Ali, M. Tejfel, A. D. Zoltán, GSM-enabled wireless patient monitoring system integrating microcontroller for managing vital signs, Al-Khwarizmi Engineering Journal 20 (2024) 65–72.