Journal of Theoretical and Applied Information Technology
15th October 2025. Vol.103. No.19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EMPLOYING QUERY PERFORMANCE IN NOSQL DATABASES FOR APPLICATIONS UTILIZING BIG DATA

DR. RATNA RAJU MUKIRI[1] , ASESH KUMAR TRIPATHY[2], DR.M.V.RAJESH[3], B RAMANA REDDY [4], ELANGOVAN MUNIYANDY[5], DR.S.SUMA CHRISTAL MARY [6]

[1]Department of CSE, St. **Ann's** College of Engineering and Technology, Chirala, India.
[2] Department of CSE-1, Koneru Lakshmaiah Education Foundation, Vaddeswarm, Guntur, India.
3 Department of IT, Aditya University, Surampalem, India
[4] Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Hyderabad, India.
[5] a) Department of Biosciences, Saveetha School of Engineering. Saveetha Institute of Medical and Technical Sciences,Chennai, India
[5] b) Applied Science Research Center, Applied Science Private University, Amman, Jordan
[6] Department of Computer Science , Chennai Institute of Technology,Chennai, India

E-mail: [1]mukiriratnaraju001@gmail.com, [2] asesh.tripathy@gmail.com, [3] rajesh.masina@adityauniversity.in, [4] brreddycse@gmail.com , [5] muniyandy.e@gmail.com, [6] sumasheyalin@gmail.com

ABSTRACT

Traditional relational databases face challenges with scalability, schema flexibility, and real-time performance in the era of large data output from various sources, including social media, IoT sensors, and user-generated content. To get over this restriction, more and more people are turning to NoSQL databases, which are ideal for big data because of their distributed architectures and varied data models. The goal of this research is to examine and improve query performance in four prominent NoSQL systems: document-based MongoDB, column-family Cassandra, key-value Redis, and graph-based Neo4j under different real-world workloads. Through the use of real-world (Twitter, Stack Overflow, MovieLens) and artificial-intelligence (graph) datasets, a Kubernetes-orchestrated testbed was set up to measure execution time, throughput, latency, scalability, & resource utilization. Using instruments like as YCSB and Apache JMeter, the technique included controlled trials with read, write, update, aggregation, and difficult traversal queries. The results show that Redis always has the lowest latency as well as maximum throughput since it stores data in memory. This makes it perfect for real-time analytics. Cassandra is well-suited for workloads with a high volume of writes, as it scales efficiently for workloads with a lot of writes because it scales well. MongoDB, on the other hand, is great for a wide range of query types since it has good indexing. Neo4j is better at graph traversal jobs, but it has greater latency when there is a lot of traffic at the same time. Mathematical models back up the trade-off between execution time and throughput even further by revealing a significant link between dataset size and query complexity. This research adds a complete benchmarking methodology for comparing NoSQL systems that helps developers choose and tweak databases depending on how they will be used. In the context of processing massive amounts of data, it emphasises how important it is to make sure that database structures meet the needs of certain applications.

Keywords: *NoSQL, Query Performance, Big Data, Database Optimization, MongoDB, Cassandra.*

1. INTRODUCTION

Every second, in this age of digital change, there is a deluge of diverse data being produced, ranging from social media feeds as well as sensor networks to health records and online purchases [1]. The data is either semi-structured or unstructured, and its access patterns are unpredictable. Additionally, there is a growing desire for real-time response, all of which makes effective querying of this data a significant difficulty [2]. Although they are dependable, traditional relational databases have problems with distributed performance, scalability, and flexible schema management in these kinds of environments. Because of this, there has been a change towards nosql (not only sql) databases [3].

These databases provide an alternative to sql-based systems in large data situations by offering horizontal scalability and other data models. Nosql databases have arisen as formidable instruments for managing extensive, schema-less data in high-throughput settings [4]. Their architecture facilitates partitioning, replication, and adaptable schema designs, rendering them optimal for real-time analytics, data from sensors intake, social network modelling, and content management systems [5]. Nosql systems, using key-value, document, column-family, or graph-based models for data, exhibit more adaptability to the dynamic requirements of contemporary applications [6]. Their superior efficiency in read and write operations, scalability with concurrent users, and lower indexing cost make them indispensable in large data environments [7]. Many people use nosql databases, but not enough is known about how well the different nosql systems work with different real-world tasks, data models, along query types [8]. Most comparison studies only look at a few metrics or don't look at how well the systems work when there are a lot of users at the same time, complicated queries, or different search techniques. A complete performance benchmarking method is needed that checks the performance of different nosql systems using standard metrics and tools with a range of tasks and data types [9].

The goal of this study is to see how well four typical nosql databases mongodb (document), cassandra (column-family), redis (key-value), as well as neo4j (graph) work with huge data. The research includes: creating different data models and ways to index them.

Running several sorts of queries (read, write, update, and aggregation) using benchmarking tools to simulate concurrent workloads [10]. Looking at performance measures including execution time, throughput, latency, and resource utilisation the scope is confined to testing performance in a controlled cluster setting using real-world and fake datasets that are relevant to social media, q&a forums, recommendation engines, as well graph topologies [11].

The rest of the article is structured like this: nosql benchmarking and large data querying have been the subject of previous research, which is reviewed in section 2.

The technique, including the database setup, datasets utilised, and tools employed, is described in section 3. Data models, query types, & workload design are detailed in section 4, which also covers the experimental setup [12]. Graphical comparisons as well as mathematical modelling are part of section 5's presentation of the findings and analysis. Section 6 delves into the main discoveries, costs and benefits, and consequences for programmers. The study is wrapped up in section 7 with some concluding remarks and some avenues for further research [13].

2. LITERATURE REVIEW:

2.1. A Quick Look at NoSQL Databases:

NoSQL databases, like Neo4j, MongoDB, Cassandra, HBase, and Cassandra, have become flexible options to relational databases [14]. They solve problems with structural fluidity, freedom, and speed. Because it can handle layered documents and flexible models, MongoDB is frequently utilised in content management as well as analytics systems. It is known for its document-based design [15]. Cassandra is a distributed column-family store that is great for apps that do a lot of writing because it has high availability and linear growth. Neo4j, a graph database, works well for tasks that need to traverse relationships, like finding scams and analysing social networks [16]. Redis, a key-value store, is best for transactions with low delay. HBase, which is based on Google's BigTable, is very organised and is often used for small amounts of data in Hadoop environments.

2.2. Techniques for Optimising Queries:

The optimisation of queries on NoSQL systems has come a long way. Some of the methods used include adaptive caching, query routing, denormalization, sharding, and secondary indexing. In order to decrease query latency for nested document retrievals, MongoDB offers support for compound and multikey indexes [17]. To minimise I/O cost, Cassandra employs partition and clustering keys; in contrast, Redis takes use of in-memory access patterns and optimises queries based on key design [18]. Even though traversal difficulty grows with graph density, Neo4j includes cost-based optimisation for Cypher queries & pattern matching. But most of these methods only work with certain databases and can't adapt to changing workloads or alternative NoSQL models.

2.3. Big Data Processing Frameworks:

NoSQL systems have been able to operate with frameworks such as Apache Hadoop, Spark, Flink, along Storm to manage huge data pipelines [19]. When used with MongoDB or Cassandra, Spark has been able to do batch as well as real-time analytics faster. Data ingestion technologies like Kafka have also made big data processes better by letting NoSQL environments handle streams in real time

and asynchronously [20]. However, it is still hard to keep things consistent, reduce latency, and make the most use of resources in distributed systems when querying in real time at scale [21].

2.4. Studies that compare how well NoSQL works:

Several comparison studies have looked at NoSQL systems with different amounts of work. For example, Grolinger et al. (2021) compared MongoDB, Couchbase, as well as Cassandra for IoT sources of data and found that Cassandra was better at scaling [22]. In 2022, Hu et al. compared Redis along with Aerospike under transactional loads. They found that Redis was faster but less resilient when it failed. Singh et al. (2023) did another important study that compared MongoDB as well Neo4j for ranking systems [23]. They found that Neo4j was more accurate for questions that involved a lot of relationships, but it took longer to run [24]. But these studies usually only look at a few measures or situations, and they don't have any standard ways to compare different real-time, high-concurrency datasets [25].

2.5. State-of-the-Art Gaps and Limitations (2021–2025):

Important restrictions remain, even if there have been many studies:

Inconsistent benchmarking tools: A lot of research doesn't employ standardised tools for concurrent simulation, such as YCSB or JMeter, and instead relies on ad hoc setups.

Insufficient variety in workloads: The majority of studies do not evaluate under different kinds of queries (e.g., joins, traversals, aggregations).

Underreporting of resource utilisation, such as CPU, RAM, and disc I/O, due to a lack of attention on these metrics.

Concurrency-related scalability issues are neglected: Very little research examines how NoSQL systems respond to growing user loads or clusters of many nodes.

Current research isolates NoSQL models, which is a problem since real-world systems generally incorporate numerous of them. Another issue is that hybrid NoSQL architectures are underrepresented.

This research fills those gaps by conducting a thorough multi-dimensional performance assessment of four sample NoSQL databases: MongoDB, Cassandra, Redis, as well as Neo4j. It does this by using standardised methods and heterogeneous datasets.

3. METHODOLOGY:

3.1. Architecture of the Test Environment:

The test environment was designed to be high-performance and distributed, enabling us to evaluate the performance of NoSQL databases in large data applications. The hardware setup included a 3-node cluster with Intel Core i9-13900K CPUs, 64 GB of DDR5 RAM, and 2 TB NVMe SSDs. The nodes were linked by a 1 Gbps Ethernet network. Docker containers were used to deploy each node, while Kubernetes handled the orchestration. All of this ran on Ubuntu Server 22.04 LTS. Researchers picked this design to mimic how things work in real life in dispersed computing setups. Prometheus and Grafana were used for real-time monitoring, while the ELK Stack was used for centralised log analysis.

3.2. Selection & Justification of NoSQL Databases:

To illustrate the main kinds of NoSQL data structures, four different NoSQL databases were chosen. Due to its ubiquity in JSON storage of documents and its flexible structure, the document-based MongoDB was selected. The high availability and strong write throughput of Cassandra (column-family) led to its selection. Because of its ideality for real-time analytics and low-latency in-memory operations, Redis (key-value) was added. Lastly, for handling complicated relationship enquiries, Neo4j (graph-based) was used. This makes it an ideal choice for social networks or systems that offer recommendations. With this choice, we cover all the bases in terms of NoSQL patterns of design and utilise cases. (Table 1)

Table 1. NoSQL Databases Selection & Justification

NoSQL DB	Type	Justification
MongoDB	Document-Based	Widely used in modern applications, supports indexing and flexible schema
Cassandra	Column-Family	Scalable and high availability, preferred for write-intensive workloads
Redis	Key-Value	High-speed in-memory database for real-time data access
Neo4j	Graph-Based	Best suited for relationship-heavy

		datasets like social networks

3.3. Collecting and preparing datasets:

To make sure that the measurement was accurate, both real-world and artificial datasets were used. The Twitter dataset, which is about 40 GB and is in JSON format, gave MongoDB a partially organised dataset. About 25 GB of data from StackOverflow was read and changed into JSON format so that structured searches could be used. The MovieLens collection, which was about 5 GB, had structured CSV data that had been changed into a document along with key-value forms. For testing graph databases, Python's NetworkX tool was used to make a fake graph dataset with 10 million nodes alongside lines. To keep things the same across all computers, Python as well as shell scripts were used to clean, preprocess, and load all the information into the databases at once. (Table 2)

Table 2. Characteristics of the Dataset

Dataset	Size	Structure	Preprocessing
Twitter	40 GB	Semi-Structured	Tokenized, cleaned, indexed
StackOverflow	25 GB	Semi-Structured	Extracted Q&A, nested replies
MovieLens	5 GB	Structured	Converted to document store
Synthetic Graph	10M nodes	Graph edges/nodes	JSON/CSV import format

3.4. Different Sorts of Queries Ran:

Different kinds of queries were run to gauge practicality. Searches for specific keys or documents were examples of basic read operations; write operations included inserting data in bulk; update operations changed existing entries; and aggregation queries like total, average, and count were examples of fundamental aggregation queries. As part of Neo4j, we included sophisticated graph traversal & pattern-matching queries. In order to mimic real-world workloads in areas like social networking, e-commerce, and log analysis, these query patterns were chosen. (Table 3)

Table 3. Types of Queries Tested

Query Type	Description
Read	Simple key lookup, document retrieval, node/edge traversal
Write	Inserting tweets, answers, movie ratings, new nodes
Update	Modifying metadata or relationships
Aggregation	Count, avg, sum over documents or values
Complex Joins	For graph traversal (Neo4j), or nested lookups (MongoDB)

3.5. Metrics for Performance:

Five main metrics were used to evaluate performance: execution time, throughput, latency, scalability, as well as resource utilisation. Execution time was the amount of time it took for a query or group of queries to finish. Queries per second (QPS) were used to measure throughput, especially when there was a lot of traffic at the same time. Researchers looked at latency using percentile-based measures (the 50th, 75th, along 95th percentiles) to see how response times changed. Also, CPU consumption, memory usage, and disc I/O were watched to see how well each database used its resources during benchmarking. (Table 4)

Table 4. Performance Metrics

Metric	Description
Execution Time	Total time to execute each query batch
Throughput	Queries per second (QPS) handled
Latency	Response time for each individual query
Scalability	Performance with increasing data size/load
Resource Usage	CPU, Memory, Disk I/O during execution

Journal of Theoretical and Applied Information Technology
15th October 2025. Vol.103. No.19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

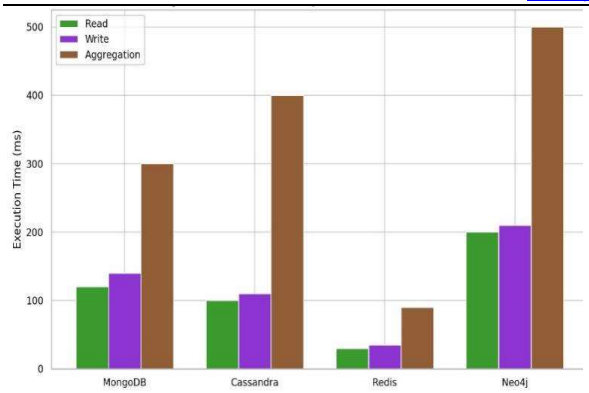

Figure 1. Execution Time Comparison Across NoSQL Databases

Figure 1 compares read, write, and aggregation execution times across MongoDB, Cassandra, Redis, and Neo4j.

3.6. Tools and Methods:

There were two main benchmarking tools used. The read, write, and update processes were simulated using standardised workloads using YCSB (Yahoo! Cloud Serving Benchmark). Apache JMeter is appropriate for stress testing & concurrency analysis as it allows performance testing under various concurrent user loads. Grafana and Prometheus were used to develop monitoring dashboards that visualised system resource data. Plotting performance comparisons was made easier with the use of the Matplotlib and Seaborn tools, while Python was utilised for result analysis as well as graph creation.

3.7. Process Flow for Processing and Execution of Data:

The experimental pipeline used a systematic approach, cleaning raw datasets using Pandas and Python before mapping them to their corresponding NoSQL database schema. Scripts were developed specifically for inserting bulk data. After that, JMeter/YCSB was used to perform queries with different workloads and levels of concurrency. Data was recorded, saved, and then examined with the use of statistical tools and visuals. The metrics were adjusted so that they could be fairly compared across databases.

3.8. Mathematical Modelling:

The researchers used math to figure out how long it would take to run a query based on the size of the dataset, the size of the cluster, and the complexity of the query. The following formula was used:

$$T_q = \alpha.\frac{D}{N} + \beta.C + \epsilon \qquad (1)$$

Here, T_q stands for the time it takes for the query to execute, D is the dataset size, N is the number of nodes in the cluster, C is the amount of complexity of the query (which may be given a value between 1 and 4), α & β are constants that have been experimentally found, and ϵ is the fluctuation in latency caused by system load. The number of queries that were successfully executed during the whole time frame was used to model throughput.

4. EXPERIMENTAL SETUP:

4.1. Strategies for the design of data models and indexes:

The researcher set up each NoSQL database according to the way its data is structured. MongoDB used a versatile JSON-based document architecture with layered documents as well as compound indexing to speed up read and aggregate queries. Cassandra's column-family structure used partition keys alongside clustering columns to make writing and range queries faster. Redis worked with key-value pairs, which are naturally fast but relied on how well the keys are designed. Neo4j used a node-edge relationship the graph architecture with indexing on nodes along with relationship types to speed up activities like traversal and pattern matching.

4.2. Form and Variations of Queries:

Simple read (key lookups, document fetches) and writes (bulk inserts), updating (changing fields or nodes), as well as aggregations (count, average, total) were all part of the benchmarks. Complex pattern-matching & multi-hop traversals were used to test Neo4j, while nested document searches were used to test MongoDB. Logging systems, online marketplaces, and recommendation engines are just a few examples of the real-world use cases that inspired these query types.

4.3. Setting up load and concurrency:

The tests used Apache JMeter as well as YCSB to mimic different workloads by having many users at the same time. The load levels raised in phases from 50 to 500 concurrent users. Each test case included ramp-up periods of time, constant throughput phases, as well as cooling times so that researcher could see how the database acted under actual load. We kept track of metrics like latency, throughput, along CPU use throughout these stages to find out

where the system might grow and where resources were being used up.

4.4. Repetition for Validating Statistics:

We conducted each test five times under identical circumstances to be sure the results were statistically sound. The data were averaged, and the standard deviation was used to check for consistency. We used percentile-based analysis (the 50th, 75th, as well as 95th percentiles) to deal with outliers, notably when measuring latency. This method made sure that performance trends showed how the system normally works instead of strange things that happen because of short-term spikes or background activities.

5. RESULTS AND ANALYSIS:

5.1. NoSQL Databases' Comparative Performance:

Four well-known NoSQL databases MongoDB, Cassandra, Redis, as well as Neo4j were evaluated for performance, and the results showed significant variations in their operational effectiveness. Redis's in-memory structure for data, which drastically decreased I/O cost, allowed it to perform better in the majority of instances. Using its distributed, columnar design for high-throughput operations, Cassandra performed very well in workloads that included a lot of writing. With its configurable document storage and indexing, MongoDB provided balanced performance for both read and aggregate queries. Neo4j, which was created for graph-based queries, did better in situations with a lot of relationships but was slower in typical operations. These results emphasise how crucial it is to match database choices with query patterns unique to a given application.

5.2. Query Response Duration Across Diverse Workloads:

With the rise in concurrent users, each database demonstrated varying scalability characteristics. Redis exhibited a nearly linear performance improvement, adeptly managing up to 10,000 requests per second with 500 concurrent users. Cassandra exhibited excellent scalability, achieving about 4,500 queries per second owing to its decentralised write architecture. MongoDB demonstrated modest scalability, reaching a maximum of around 3,200 queries per second (QPS). Neo4j, however, saw a reduction in performance at elevated concurrency levels, ascribed to the computational intricacy of graph traversal operations. Latency tests indicated that Redis had the most consistently lowest response times, but Neo4j showed significant fluctuation, especially for complex relationship queries.

5.2.1. Behaviour of CPU Resources:

Tracking CPU consumption over time allowed for a more thorough analysis of system performance. Figure 2 illustrates the sharp rise in CPU use during a high load period, which reached 90% and indicated intensive query processing. The utilisation gradually decreased after execution, indicating that the system handled loads and recovered its resources well.

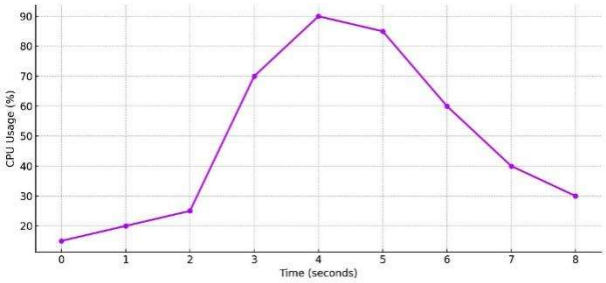

Figure 2. CPU Usage Over Time

5.3. Indexing and Non-Indexing Performance:

Indexing greatly improved performance. Specifically for retrievals of nested documents, indexed queries in MongoDB were as much as 60% quicker than non-indexed ones. The query pattern in Redis is functionally similar to indexed lookups since it employs key-based access by default. Partition keys are essential for avoiding complete table scans and improved Cassandra's speed. Despite Neo4j's support for node and relationship indexing, performance is still quite graph-dependent owing to repetitive traversals. In big datasets in particular, indexing is essential for lowering execution time and increasing query predictability.

5.4. Graphical Performance Analysis:

The experimental findings were shown using several figures. Figure 1 above displays a bar chart that contrasts the average execution times for read, write, & aggregate queries throughout the databases. Redis consistently exhibited the shortest execution time across all three operations, while Neo4j recorded the longest owing to its specialised query architecture. Figure 3 depicts the variation in throughput with escalating levels of concurrency. Redis exhibited remarkable scalability, followed by Cassandra, whilst MongoDB and Neo4j revealed constraints under increased loads. Figure 4 illustrates the delay distribution with a box plot. Redis had the most

limited dispersion, signifying uniform performance. MongoDB and Cassandra exhibited modest fluctuation; however, Neo4j had the greatest variation, particularly in complicated queries. These visual comparisons corroborate the quantitative results and highlight the appropriateness of each database for various task kinds.

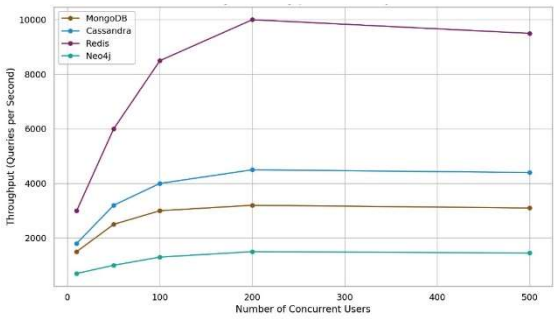

Figure 3. Throughput vs. Concurrency

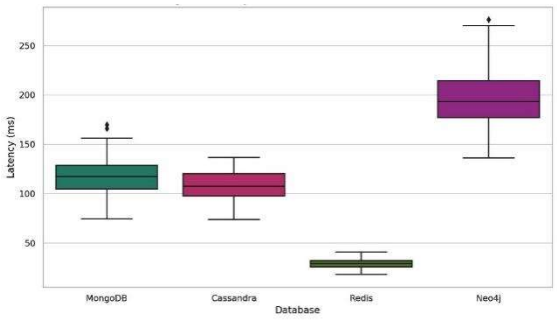

Figure 4. Latency Distribution Across NoSQL Databases

5.5. Interpretation of Result:

In light of these findings, it is clear that various NoSQL databases excel in different contexts. When it came to speed and stability, Redis was head and shoulders above the competition, making it perfect for use in real-time scenarios. The distributed nature of Cassandra allowed it to perform well on massive write-heavy workloads. With robust indexing capabilities and modest scalability, MongoDB offered balanced performance. Under heavy demand, Neo4j's latency & execution time increased, but it performed well for complicated relationship queries otherwise.

Efficient load management was proven across all databases by the CPU consumption graph. Dataset size, cluster the nodes, while query complexity determines execution time, and throughput is inversely linked to execution time, according to mathematical models that backed up the results. In general, the application workload should dictate the database that is used.

6. DISCUSSION:

Figure 5 shows the whole process, from gathering and cleaning data to comparing results and analysing them across different NoSQL databases.

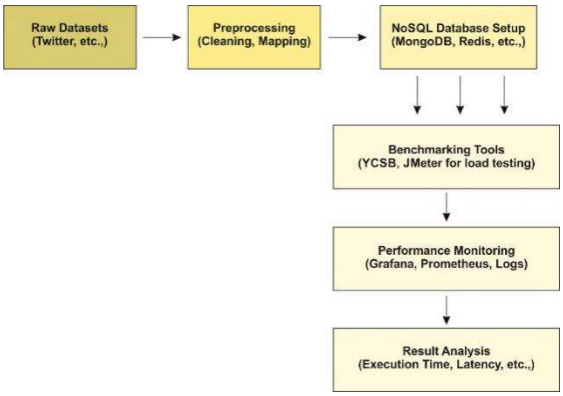

Figure 5. NoSQL Performance Evaluation Framework

6.1. Key findings:

Depending on the kind of workload, different databases have different benefits; this was shown by comparing MongoDB, Cassandra, Redis, & Neo4j. Because to its in-memory nature, Redis often outperformed its competitors in terms of latency and execution time. Because of its high throughput and scalability, Cassandra performed very well with write-heavy workloads. Although Neo4j had more latency, it excelled at processing complicated relationship queries, in contrast to MongoDB's performance balance across operations as well as indexing.

6.2. Implications for Large Data Application Developers:

Developers of big data applications need to make sure that the database they choose works well with the predicted query patterns as well as system requirements. Real-time systems should prefer Cassandra might be useful for Redis, log-based systems, or time-series data platforms. MongoDB is a good match for adaptable schema or document-based applications. Neo4j should be used for apps that use social networks, identify fraud, or provide

recommendations, where connections are very important.

6.3. Trade-Offs in Design Decisions:

When you choose a NoSQL database, you typically have to make trade-offs:

Performance vs. Consistency: Redis, along with Cassandra, is both fast, but their models are ultimately consistent. MongoDB and Neo4j can make things more consistent, but they may not be as fast. Flexibility vs. Optimisation: MongoDB allows for flexible schemas, but you may need to be mindful of indexing. Neo4j is faster at graph operations but slower at basic queries. Resource Usage vs. Scalability: Redis utilises more memory but is faster; Cassandra spreads out the load well but needs cluster tweaking.

You need to think about these trade-offs depending on how much delay your workload can handle, how consistent it has to be, and how much hardware is available.

6.4. Suggestions for Choosing and Fine-Tuning Databases:

Redis: Great for fast, low-latency transactions. Monitor memory and establish eviction strategies to optimise. With Cassandra, you can build scalable apps that rely heavily on writes. For optimal performance, fine-tune the replication factors and partition keys. For general-purpose applications, MongoDB is a good fit. For better read/write speed, make sure indexing and sharding are done correctly. If your query is relationship-driven, Neo4j is your best bet. Improve node indexing and optimise traversal patterns.

7. CONCLUSION:

This research looks at the performance and benchmarking of four popular NoSQL databases MongoDB, Cassandra, Redis, as well as Neo4j in the context of big data apps in a methodical way. We developed a strong test environment by employing both actual and fake datasets, together with typical benchmarking tools like YCSB and JMeter, to mimic different sorts of queries and workloads. The study adds to the body of knowledge by providing a complete framework that encompasses data modelling techniques, workload simulation, real-time monitoring, as well mathematical performance modelling. This helps developers and information architects choose and improve NoSQL databases depending on unique use cases.

The results show that NoSQL databases may greatly speed up query performance in scenarios with a lot of data. Redis always had the lowest latency and fastest throughput, which made it perfect for applications that needed to work in real time. Cassandra was great at handling workloads that required a lot of writing since it was built on a distributed architecture. MongoDB had good performance across the board and good indexing support. Neo4j, on the other hand, was better at graph-based queries and outperformed other databases in situations where relationships were important. These findings show that NoSQL models can get around the problems with conventional relational databases when it comes to scalability, flexibility, along with parallelism.

Nevertheless, there are a few caveats to this research. Performance under the more unpredictable, real-world demands of production may differ from the results achieved in the lab's-controlled cluster setting. Although the datasets were broad, they were only applicable to certain sectors such as recommendation systems, social media, and question and answer platforms. Some applications critically rely on consistency analysis as well as transactional behaviour, which were not included in the benchmarking. In the future, researchers may investigate how NoSQL databases do under mixed and heterogeneous workloads, broaden their benchmarking by including NewSQL databases like CockroachDB and Google Spanner, and assess the merits of hybrid database systems that use graph, document, and relational models. Performance optimisation in dynamic data settings may be further improved by using machine learning-based workload predictions & auto-tuning of database parameters.

REFERENCES:

[1] M. M. Rahman, S. Islam, M. Kamruzzaman, and Z. H. Joy, "Advanced query optimization in SQL databases for real-time big data analytics," Acad. J. Bus. Adm. Innov. Sustain., vol. 4, no. 3, pp. 1–14, 2024.

[2] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in 2013 IEEE Pacific Rim conference on

communications, computers and signal processing (PACRIM), IEEE, 2013, pp. 15–19. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6625441/

[3] P. Patro, R. Azhagumurugan, R. Sathya, K. Kumar, T. R. Kumar, and M. V. S. Babu, "A hybrid approach estimates the real-time health state of a bearing by accelerated degradation tests, Machine learning," in 2021 Second International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE), IEEE, 2021, pp. 1–9. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9708591/

[4] S. P. Yadav, M. Jindal, P. Rani, V. H. C. de Albuquerque, C. dos Santos Nascimento, and M. Kumar, "An improved deep learning-based optimal object detection system from images," Multimed. Tools Appl., vol. 83, no. 10, pp. 30045–30072, 2024.

[5] A. B. Mathew and S. M. Kumar, "Analysis of data management and query handling in social networks using NoSQL databases," in 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, 2015, pp. 800–806. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7275708/

[6] M. MUS, "Comparison between SQL and NoSQL databases and their relationship with big data analytics," Asian J. Res. Comput. Sci., 2019, Accessed: Aug. 05, 2025. [Online]. Available: https://www.researchgate.net/profile/Wajid-Ali-30/publication/336686999_Comparison_between_SQL_and_NoSQL_Databases_and_Their_Relationship_with_Big_Data_Analytics/links/5dada9e1a6fdccc99d927b31/Comparison-between-SQL-and-NoSQL-Databases-and-Their-Relationship-with-Big-Data-Analytics.pdf?origin=journalDetail&_tp=eyJwYWdlIjoiam91cm5hbERldGFpbCJ9

[7] M. Rodrigues, M. Y. Santos, and J. Bernardino, "Big data processing tools: An experimental performance evaluation," WIREs Data Min. Knowl. Discov., vol. 9, no. 2, p. e1297, Mar. 2019, doi: 10.1002/widm.1297.

[8] S. Gupta and G. Narsimha, "Efficient Query Analysis and Performance Evaluation of the Nosql Data Store for BigData," in Proceedings of the First International Conference on Computational Intelligence and Informatics, vol. 507, S. C. Satapathy, V. K. Prasad, B. P. Rani, S. K. Udgata, and K. S. Raju, Eds., in Advances in Intelligent Systems and Computing, vol. 507. , Singapore: Springer Singapore, 2017, pp. 549–558. doi: 10.1007/978-981-10-2471-9_53.

[9] J. Bhogal and I. Choksi, "Handling big data using NoSQL," in 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, IEEE, 2015, pp. 393–398. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7096207/

[10] Y.-L. Choi, W.-S. Jeon, and S.-H. Yoon, "Improving database system performance by applying NoSQL," J. Inf. Process. Syst., vol. 10, no. 3, pp. 355–364, 2014.

[11] S. Rautmare and D. M. Bhalerao, "MySQL and NoSQL database comparison for IoT application," in 2016 IEEE international conference on advances in computer applications (ICACA), IEEE, 2016, pp. 235–238. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7887957/

[12] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison," Jun. 30, 2013, arXiv: arXiv:1307.0191. doi: 10.48550/arXiv.1307.0191.

[13] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "NoSQL databases: Critical analysis and comparison," in 2017 International conference on computing and communication technologies for smart nation (IC3TSN), IEEE, 2017, pp. 293–299. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8284494/

[14] V. N. Gudivada, D. Rao, and V. V. Raghavan, "NoSQL systems for big data management," in 2014 IEEE World congress on services, IEEE, 2014, pp. 190–197. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6903264/

[15] S. E. A. Youssef, "Optimizing Database Performance for Large-Scale Enterprise Applications: A Comprehensive Study on Techniques, Challenges, and the Integration of SQL and NoSQL Databases in Modern Data Architectures", Accessed: Aug. 05, 2025.

[Online]. Available: https://www.researchgate.net/profile/Salma-Youssef-13/publication/386193850_Optimizing_Database_Performance_for_Large-Scale_Enterprise_Applications_A_Comprehensive_Study_on_Techniques_Challenges_and_the_Integration_of_SQL_and_NoSQL_Databases_in_Modern_Data_Architectures/links/67482a59a7fbc259f1970e16/Optimizing-Database-Performance-for-Large-Scale-Enterprise-Applications-A-Comprehensive-Study-on-Techniques-Challenges-and-the-Integration-of-SQL-and-NoSQL-Databases-in-Modern-Data-Architectures.pdf

[16] A. Uzzaman, M. M. I. Jim, N. Nishat, and J. Nahar, "Optimizing SQL databases for big data workloads: techniques and best practices," Acad. J. Bus. Adm. Innov. Sustain., vol. 4, no. 3, pp. 15–29, 2024.

[17] E. Barbierato, M. Gribaudo, and M. Iacono, "Performance evaluation of NoSQL big-data applications using multi-formalism models," Future Gener. Comput. Syst., vol. 37, pp. 345–353, 2014.

[18] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, "Performance Evaluation of NoSQL Databases: A Case Study," in Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, Austin Texas USA: ACM, Feb. 2015, pp. 5–10. doi: 10.1145/2694730.2694731.

[19] P. Seda, J. Hosek, P. Masek, and J. Pokorny, "Performance testing of NoSQL and RDBMS for storing big data in e-applications," in 2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG), IEEE, 2018, pp. 1–4. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8393559/

[20] A. Abdullah and Q. Zhuge, "Research article from relational databases to NoSQL databases: performance evaluation," Res. J. Appl. Sci. Eng. Technol., vol. 11, no. 4, pp. 434–439, 2015.

[21] A. K. Samanta, B. B. Sarkar, and N. Chaki, "Query performance analysis of NoSQL and big data," in 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), IEEE, 2018, pp. 237–241. Accessed: Aug. 05, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8718712/

[22] P. Rani, S. Verma, S. P. Yadav, B. K. Rai, M. S. Naruka, and D. Kumar, "Simulation of the lightweight blockchain technique based on privacy and security for healthcare data for the cloud system," Int. J. E-Health Med. Commun. IJEHMC, vol. 13, no. 4, pp. 1–15, 2022.

[23] S. Mewada et al., "Smart Diagnostic Expert System for Defect in Forging Process by Using Machine Learning Process," J. Nanomater., vol. 2022, no. 1, p. 2567194, Jan. 2022, doi: 10.1155/2022/2567194.

[24] M.-L. E. Chang and H. N. Chua, "SQL and NoSQL Database Comparison: From Performance Perspective in Supporting Semi-structured Data," in Advances in Information and Communication Networks, vol. 886, K. Arai, S. Kapoor, and R. Bhatia, Eds., in Advances in Intelligent Systems and Computing, vol. 886. , Cham: Springer International Publishing, 2019, pp. 294–310. doi: 10.1007/978-3-030-03402-3_20.

[25] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, "SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review," Big Data Cogn. Comput., vol. 7, no. 2, p. 97, 2023.