# Evaluation of SQL and NoSQL Databases on Parallel Processing

**Iyas Qaddara**
Department of Computer Science, Faculty of Information Technology, Al-Ahliyya Amman University, Jordan
i.qaddara@ammanu.edu.jo (corresponding author)

**Yousef Alraba'nah**
Department of Software Engineering, Faculty of Information Technology, Al-Ahliyya Amman University, Jordan
y.alrabanah@ammanu.edu.jo

**Mohammad O. Hiari**
Department of Networks and Cybersecurity, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan
m.hyari@ammanu.edu.jo

*Received: 23 February 2025 | Revised: 12 April 2025 | Accepted: 22 April 2025*

 *DOI: https://doi.org/10.48084/etasr.10620*

**ABSTRACT**

**This study performs an independent examination of the performance exhibited by NoSQL and SQL databases regarding key-value stores in parallel processing using a simulation model containing 15 nodes. Specifically, the focus is on comparing the efficiency of read, write, delete, and instantiate operations within the context of key-value stores implemented with both NoSQL (MongoDB, RavenDB, and Cassandra) and SQL (MS-SQL) databases within the average time of three runs. The results show that SQL achieved better read operation times, while NoSQL databases, especially RavenDB, had better results in the other operations.**

***Keywords-SQL; NoSQL; simulation model; parallel processing***

## I. INTRODUCTION

Databases play a crucial role in managing and organizing vast amounts of data in various applications. Two major types of databases are SQL (Structured Query Language) databases and NoSQL. These databases can benefit significantly from parallel processing, a computing paradigm in which multiple processors work simultaneously to execute a task. This parallelism improves performance, scalability, and responsiveness when handling large datasets [1]. Relational databases, commonly known as SQL databases, employ SQL to define and manipulate data and are well-suited for applications requiring complex transactions and queries. Notable examples of SQL databases include Oracle Database, Microsoft SQL Server, PostgreSQL, and MySQL [2]. Within SQL databases, parallel processing involves the segmentation of extensive queries or transactions into smaller tasks that can be executed simultaneously. Through the utilization of multiple processors or cores, parallel execution improves query performance and accelerates response times. An integral aspect of modern SQL database management systems is the optimization of parallel queries [3].

NoSQL databases are specifically crafted to manage substantial amounts of unstructured or semistructured data. In contrast to SQL databases, NoSQL databases do not rely on a rigid schema, allowing flexibility in how data are represented. There are various categories of NoSQL databases, such as document stores, key-value stores, graph databases, and column-family stores. Well-known NoSQL databases are MongoDB, Cassandra, CouchDB, and Neo4j [4]. Effective parallel processing is imperative in NoSQL databases to efficiently distribute and oversee extensive data volumes across multiple nodes or servers. This horizontal scaling strategy significantly improves the overall performance and fault tolerance of NoSQL databases, making them suitable for applications dealing with large datasets and high transaction rates [4].

Contemporary data management relies significantly on SQL and NoSQL databases, each with distinctive strengths and weaknesses [5]. SQL databases, commonly affiliated with Relational Database Management Systems (RDBMS), exhibit characteristics such as a structured schema, adherence to ACID properties (Atomicity, Consistency, Isolation, Durability), and

a robust query language. These attributes make them well-suited for applications demanding intricate relationships and transactional capabilities, establishing them as a prevalent choice for conventional business applications. Conversely, NoSQL databases are crafted to manage substantial volumes of unstructured or semi-structured data, providing adaptability in data modeling [6]. Notably proficient in facilitating horizontal scalability, they prove instrumental for rapidly expanding distributed datasets [6]. NoSQL databases find applications in dynamic and evolving data structures, prevalent in domains such as big data, real-time applications, and content management systems. Parallel processing, which involves the simultaneous execution of multiple tasks, significantly influences the performance of both SQL and NoSQL databases [7]. This approach facilitates concurrent operations, enhancing query response times and overall system efficiency. Many contemporary database systems strategically employ parallel processing techniques to distribute workloads across multiple nodes or processors, increasing scalability and overall system performance [8].

This study employed an experimental setup to evaluate the performance of SQL and NoSQL databases in parallel processing scenarios. A simulation environment with 15 nodes and a mix of periodic and nonperiodic tasks was used to emulate a simplified parallel system capable of executing concurrent database operations. The chosen databases differ in terms of schema design, consistency models, and scalability strategies, all of which influence performance under concurrent loads. The focus was on basic interactions that are common in distributed systems, measuring key-value operations such as instantiate, read, write, and delete. This approach enables a direct comparison of how database architectures impact efficiency and responsiveness in parallel environments.

In [9], the crucial role of big data and analytics in the Industry 4.0 (I4.0) transformation was highlighted, focusing on the development of smart factories with Intelligent Manufacturing Systems (IMS). Although many system architecture proposals for IMS are data-driven, the importance of selecting an appropriate data model, specifically relational (SQL) or non-relational (NoSQL) models, is often overlooked. This study evaluates the advantages and disadvantages of these models in the context of I4.0, considering data characteristics based on the five dimensions of big data and the asset administration shell format. By examining transactional properties and logical data models, this study aimed to guide the choice between SQL and NoSQL databases, considering factors such as data volume, variety, velocity, veracity, and value for different scenarios within I4.0.

The Internet of Things (IoT) comprises a network of sensors that rapidly collect extensive data. In [10], the efficacy of three databases, SQL (MySQL), NoSQL (MongoDB), and NewSQL (VoltDB), was evaluated in managing sensor readings, performing tests involving single write, single read, single delete, and multi-write operations on extensive sensor data. This study assessed the efficiency of these database systems in handling diverse and substantial amounts of IoT data. VoltDB was selected for sensor data analysis due to its commendable performance in write and read-intensive operations. Although MongoDB demonstrated competitive performance, its efficacy decreased with increasing data size and client counts. VoltDB emerged as the faster and superior choice in many scenarios, although MongoDB could potentially enhance its performance by adopting a flexible schema-less data model. This study concluded that the selection of a database should be based on the specific properties and requirements of the system in question.

In [11], the dynamic challenges posed by the rapid growth of the digital world were investigated, focusing on the convergence of NoSQL databases and big data analytics. Integrating these elements can significantly enhance organizations' ability to make timely decisions based on diverse and complex datasets. The study explored various NoSQL data models to provide insight into future implications. The results showed that NoSQL databases outperformed traditional SQL in big data analytics, offering superior performance, adaptability, simplicity, and scalability to handle large and distributed datasets. The conclusion advocates for the widespread adoption of NoSQL in various markets.

In the era of digitalization, social media platforms generate vast amounts of unstructured data daily, prompting the increasing use of NoSQL databases, particularly MongoDB, for storage. In [12], MongoDB (NoSQL) and MySQL (SQL) were compared in terms of indexing, performance tuning, and record support, specifically for handling large volumes of social media data. The study employed four performance measures, insert, select, update, and delete, on up to 1 million Twitter records. Although MongoDB was faster in all four operations, MySQL offered more flexibility in performance optimization due to a wider range of tuning options. The findings provide valuable insights into how these databases support semistructured social media data, considering the nuances of performance tuning.

In [13], the focus was on NoSQL databases, categorizing them as key value stores, graph databases, wide column stores, and document stores. This study compared MongoDB, Cassandra, Redis, and Neo4j based on functional and nonfunctional features. Document stores, such as MongoDB, are suitable for high performance, while wide column stores, such as Cassandra, excel for semi-structured data. Redis performs well for single-shared operations. This study also discussed the functionality of distributed environments.

In [6], various NoSQL systems were explored, providing a comparative analysis based on multiple criteria. NoSQL databases were introduced to offer high performance and availability at the expense of sacrificing the ACID traits found in traditional databases. Instead, NoSQL maintains a weaker BASE (Basic Availability, Soft state, Eventual consistency) feature. This study noted the emergence of more than 120 NoSQL solutions since the concept's official definition in 1998, with the question of which solutions truly deliver higher performance remaining open. The conclusions emphasized that although the SQL and NoSQL databases share some features, their behaviors differ in specific instances. This study suggested that they are not interchangeable for solving any type of problem, and that the choice between them should be made based on the specific requirements of a given instance.

In [1], the effectiveness of SQL and NoSQL databases was examined, focusing on their ability to handle data loading, response times, and retrieval. Various types of NoSQL databases, such as wide-column stores, documents, graph databases, and key-value pairs, provide alternatives to the standardized structure of SQL. The experiments used PHP for database connectivity to compare the performance of SQL (MySQL) and NoSQL (MongoDB) databases. The results showed that SQL databases outperformed NoSQL databases in terms of loading, responding, and retrieval times, displaying their efficiency and speed.

In [14], the evolution of databases from traditional relational to NoSQL was discussed, highlighting their advantages and disadvantages. Transitioning from relational to NoSQL is challenging due to their lower user base and lack of a standard query language. The choice between relational and NoSQL databases depends on application requirements, including scalability, flexibility, performance, and security. Relational databases are recommended for their ACID properties, while NoSQL databases offer flexibility for handling large datasets. Despite security concerns, NoSQL technologies show promising prospects. Combining both databases could provide a more effective solution.

In [15], the performance of NoSQL and SQL databases was investigated in terms of key-value stores, comparing read, write, delete, and instantiate operations. An abstract key-value pair framework was designed and implemented using all tested databases. The results showed that not all NoSQL databases perform better than SQL databases, with performance varying with each operation. There is little correlation between performance and the data model each database uses. In [16], the convergence of big data analytics and NoSQL databases was discussed, highlighting their complementary development and potential benefits for real-time decision-making. This study compared four main NoSQL data models, finding that NoSQL databases were better for scenarios that require simplicity, adaptability, high-performance analytics, and distributed scalability on large data. This study concluded that NoSQL can coexist with traditional SQL databases for big data analytics and offers flexible data modeling suitable for dynamic scalability and improved performance. Finally, this study suggested that NoSQL can be used as a new type of data architecture to coexist with traditional SQL databases.

## II. METHODOLOGY

This study aimed to compare the performance of SQL and NoSQL databases in parallel processing environments using key-value store operations. Figure 1 provides an overview of the method used. The NS-3 simulation was used to model task execution in a parallel environment. This workflow visually reinforces the experimental pipeline and highlights the logical structure of the evaluation process.

### *A. Research Design*

Table I provides a detailed overview of the databases tested in this study. For each database management system, the selected operation (e.g., read or write) was executed three times and the average time was recorded. The dataset for the experiment was made up of autogenerated key-value pairs in the form $(kN, vN)$ where $N$ is a sequence number up to twenty million.

TABLE I. DATABASES AND VERSIONS

| Database | Version |
|---|---|
| MongoDB | 5.0 |
| RavenDB | 5.2 |
| Cassandra | 4.0 |
| Microsoft SQL Server | 2019 |

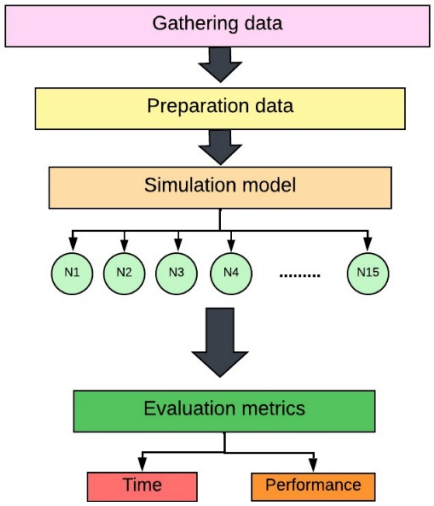


Fig. 1. Workflow of this study.

MongoDB is a NoSQL document-oriented database that stores data in flexible JSON-like BSON documents. It is designed for scalability and flexibility, allowing developers to work with varying data structures within the same database. MongoDB is known for its ease of use, horizontal scalability through sharding, and rich query language. It is suitable for a variety of applications, ranging from content management systems to real-time analytics, where flexibility and scalability are crucial.

RavenDB is a NoSQL document-oriented database that focuses on providing ACID transactions in a NoSQL environment. It supports a flexible data model like MongoDB but emphasizes strong consistency and transactional capabilities. RavenDB is often chosen for applications that require a balance between the benefits of NoSQL databases and the transactional guarantees associated with traditional relational databases. It is particularly suitable for scenarios where data consistency is a top priority.

Cassandra is a NoSQL-distributed database designed for high availability, fault tolerance, and scalability. It follows a column-family data model and is optimized for write-intensive workloads. Cassandra's peer-to-peer architecture allows it to scale horizontally by adding more nodes to the cluster. It is commonly used in scenarios that require high write and read throughput, such as time-series data, sensor data, and real-time analytics. Cassandra sacrifices some features of traditional relational databases for high scalability and fault tolerance [17].

Microsoft SQL Server is an RDBMS that follows the traditional relational database model [18]. It supports ACID

transactions, providing strong consistency and data integrity. SQL Server is widely used in enterprise environments and offers a comprehensive set of features, including support for complex queries, indexing, stored procedures, and triggers. It is suitable for a wide range of applications, including business-critical applications where data relationships and transactional integrity are paramount. It also integrates well with other Microsoft technologies and provides robust security features.

These database systems were chosen for their architectural diversity and widespread use in real-world applications. MongoDB and RavenDB are representative of document-based NoSQL systems, Cassandra is a column-oriented NoSQL database designed for high availability and horizontal scalability, and MS SQL Server is a traditional relational system commonly used in enterprise applications. This diverse selection enables a balanced performance comparison under parallel workloads. Furthermore, the study focuses on key-value store operations (instantiate, read, write, delete) because they provide a simplified yet powerful abstraction that is heavily used in parallel and distributed systems. These operations are essential for understanding system behavior under concurrent access patterns and align with fundamental database interaction models such as CRUD.

For each operation tested (instantiate, read, write, delete), the database action was executed three times under identical simulation conditions. The reported average time is the arithmetic mean of these three measurements. Although this number provides a basic indication of typical performance, it does not fully capture variability or allow robust outlier detection. All tests were carried out in a controlled environment, minimizing noise and ensuring consistent system state. However, future experiments will consider a larger sample size and include standard deviation analysis to enhance the reliability and statistical rigor of the performance results.

### *B. Simulation Model*

The simulation employed a system consisting of 15 nodes, each containing two active and three passive resources. Due to the simulation's demanding CPU and memory requirements, a larger set of nodes and additional resources were not explored. Each node is associated with a periodic task featuring a 3000-unit period and a 600-unit computation time, utilizing all available resources. Nonperiodic tasks, unless otherwise specified, follow normal distributions for both computation time and laxity. The mean computation time and laxity for the tasks are 600 and 800 time units with standard deviations of 100 and 300, respectively.

Nonperiodic tasks are introduced through a Poisson process, leading to diverse node loads due to variations in arrival rates across nodes. The concept of the system-wide nonperiodic task arrival rate (R) is introduced, representing the cumulative arrival rates of nonperiodic tasks across all nodes. To standardize the arrival rate, the task arrival rate is defined as the average number of arrivals per 600 time units (matching the mean task computation time of 600 units). This normalization provides flexibility in adjusting the specific computation time value. Although 600 was chosen as the nominal mean laxity value, the impact of varying laxity values was also explored.

Each nonperiodic task requires at least one active resource and may utilize zero or more passive resources, with randomly determined resource requirements. Specifically, a non-periodic task has a 2/3 probability of requesting each of the two active resources and a 0.5 probability of requesting each passive resource. Although the choice of probabilities is arbitrary, it has a direct impact on the system load (L).

The simulation model assumes the execution of both the global and local schedulers on a dedicated coprocessor designed specifically for scheduling and other system support tasks. This coprocessor serves to isolate application tasks from external interrupts and overhead caused by the execution of kernel modules, ensuring predictable system behavior. Given the specialized nature of real-time systems, which often involve dedicated hardware for tasks such as processing information from sensors, relying on an additional specialized coprocessor is justifiable, especially considering the additional benefits it provides [19].

The interconnectivity between nodes is made easier by a star communication network, where each channel is tied to a certain node. Exactly one end of a channel connects to the node it is mapped to; the other ends of all channels are connected to each other. A message passed from one node to another will first pass through the sender channel and then go through the receiver channel. The star topology was selected due to its prevalence in real-world systems such as centralized cloud services, IoT platforms with hub devices, and hierarchical monitoring frameworks. This topology allows all nodes to communicate through a central point, simplifying routing logic and reducing network complexity within the simulation. Such a configuration aligns well with scenarios where a centralized scheduler or controller distributes tasks and aggregates results. Although this approach provides clarity in communication performance analysis, future work could expand the model to include more decentralized topologies (e.g., mesh or tree) to simulate alternative system architectures.

This experimental design was carefully structured to align with the research goal of comparing the performance of SQL and NoSQL databases under parallel processing conditions. The use of 15 nodes in a simulated NS-3 environment mimics distributed processing units commonly found in parallel systems. Task parameters such as periodicity, laxity, and probabilistic resource usage model dynamic workloads that stress concurrent database access. The core key-value operations tested (instantiate, read, write, delete) correspond to universal CRUD operations, allowing for a fair comparison of systems that differ in schema structure, consistency models, and data access patterns. The selected databases, MS SQL, MongoDB, RavenDB, and Cassandra, represent a range of data architectures, from strictly structured transactional systems to highly scalable NoSQL platforms. This configuration ensures that performance differences can be attributed to underlying architectural characteristics, thus supporting the comparative analysis at the heart of this research.

The simulation model was implemented using NS-3, and Figure 2 shows the initial code for building the simulation with 15 nodes. The model includes 15 processing nodes connected through a central communication hub in a star topology. The

configuration of 15 nodes was selected to represent a mid-scale parallel system, offering sufficient complexity to reflect realistic distributed behavior while ensuring that the NS-3 simulator could execute the model efficiently. Using more nodes or resources would have significantly increased memory and CPU requirements, potentially compromising repeatability and runtime feasibility. Each node was designed with 2 active and 3 passive resources to mirror typical multicore processing environments, where computation tasks (active resources) interact with shared memory, caches, or I/O buffers (passive resources). This configuration allowed us to evaluate how parallel database operations behave under controlled but realistic conditions.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"

using namespace ns3;

int main (int argc, char *argv[])
{
  LogComponentEnable ("Simple15NodeSimulation", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (15);

  PointToPointHelper p2p;
  p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));

  for (uint32_t i = 0; i < 14; ++i) {
    NetDeviceContainer devices = p2p.Install (nodes.Get(i), nodes.Get(i+1));
  }

  InternetStackHelper internet;
  internet.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  for (uint32_t i = 0; i < 15; ++i) {
    Ipv4InterfaceContainer interfaces = address.Assign (p2p.GetDevice (i));
  }

  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

Fig. 2. Simulation model with 15 nodes in NS-3.

## III. RESULTS AND DISCUSSION

The comparison involved four fundamental operations with three runs:

- Instantiate a storage bucket for the key-value pairs.
- Read: This involves initializing a value that is read from the storage in the form of a key-value pair. This is similar to the Read operation in the CRUD model, which is widely used when describing the key database operations.
- Write: In case the Storage object does not hold a certain key-value pair, then Storage will store the key-value pair. Otherwise, it sets the value for the key in the storage for a given key. Therefore, this operation involves the Create and Update operations of the CRUD model.
- Delete. This deletes the record (i.e., key-value pair) corresponding to a given key from the key-value pair storage.

The first experiment measured the time taken to instantiate a database bucket. Figure 3 summarizes the results of this experiment. RavenDB exhibited the fastest instantiation time, followed by MongoDB and Cassandra, while MS SQL showed the slowest performance in this operation. These results indicate that NoSQL systems may have efficiency advantages in setting up key-value storage under parallel processing conditions.

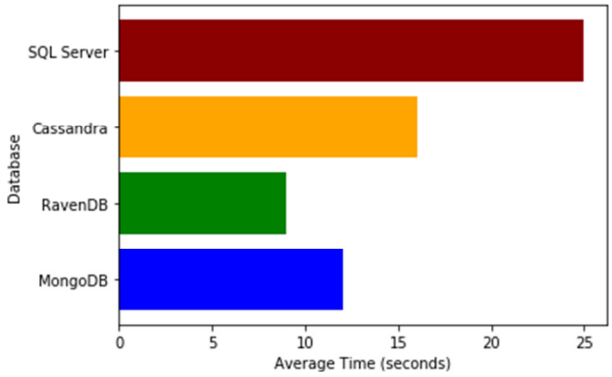


Fig. 3. Instantiate operation results.

The second experiment measured how long it took to read values associated with certain keys in the bucket. The results are summarized in Table II. The number of operations indicates how often certain operations (for example, read) were performed in the test. This is also equal to the number of records that are in the store in terms of the number of key-value pairs. The results show that MS SQL gave the best average reading time, while Raven DB gave the worst results.

TABLE II. READING RESULTS

| Database | Number of operations | | |
|---|---|---|---|
| | 100 | 1000 | 10000 |
| MongoDB | 6 | 8 | 13 |
| RavenDB | 11 | 14 | 15 |
| Cassandra | 10 | 17 | 29 |
| MS SQL | 4 | 5 | 9 |

The third experiment measured the time taken to delete specific key-value pairs in the bucket, with the results summarized in Table III. MS SQL gave the worst time, while RavenDB gave the best time.

TABLE III. RESULTS ON THE DELETE OPERATION

| Database | Number of operations | | |
|---|---|---|---|
| | 100 | 1000 | 10000 |
| MongoDB | 7 | 11 | 14 |
| RavenDB | 7 | 9 | 10 |
| Cassandra | 8 | 10 | 17 |
| MS SQL | 11 | 15 | 18 |

The fourth experiment calculated the time required to write a particular key-value pair to the bucket. If such a key-value pair already exists in the bucket, then it constitutes an update of the value. The results are summarized in Table IV, where it can be seen that MongoDB gave the best result while MS SQL gave the worst result.

TABLE IV. WRITING RESULTS

| Database | Number of operations | | |
|---|---|---|---|
| | **100** | **1000** | **10000** |
| MongoDB | 20 | 24 | 33 |
| RavenDB | 18 | 21 | 25 |
| Cassandra | 22 | 28 | 37 |
| MS SQL | 32 | 41 | 58 |

Figure 4 summarizes the comparative performance of SQL (MS SQL) and NoSQL databases across the four operations tested. The results show that while MS SQL performed best in read operations, NoSQL databases, especially RavenDB and MongoDB, tend to outperform in write, delete, and instantiate operations. This figure highlights the operation-specific strengths of each database type under parallel execution.

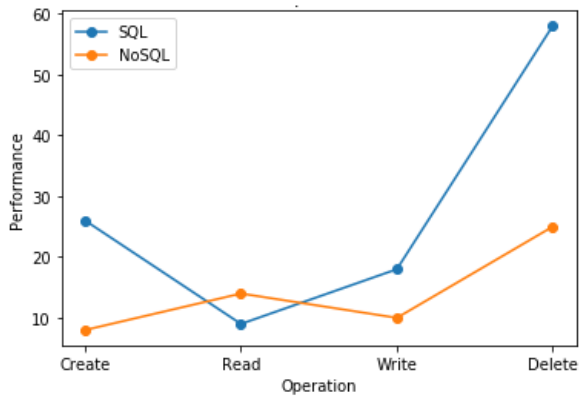


Fig. 4. Comparative performance of SQL vs NoSQL databases.

From the above results, it can be observed that not always NoSQL databases give better times, but sometimes SQL do so. The performance of each system depends on the operation.

## IV. SCIENTIFIC CONTRIBUTION

The results of this study make a substantial contribution to the discussion of SQL and NoSQL database performance in parallel processing systems. This research identifies the advantages of SQL databases in read operations while showing that NoSQL databases excel in write, delete, and instantiate operations. The findings of this study can help developers and system architects in choosing the right database technologies depending on the requirements of the application. However, this study emphasizes the need to design a distinct database optimization technique that fits a given type of operation to improve the overall performance of the system. These results open the path for more complex research to design and implement new hybrid DBMSs that consist of both SQL and NoSQL.

## V. CONCLUSION

This study evaluated the performance of key-value storage implementations across NoSQL and SQL databases. Despite the general optimization of NoSQL databases for key-value stores, SQL databases, which are not designed inherently for this purpose, exhibited competitive performance. Surprisingly, not all NoSQL databases outperformed the SQL database under examination. These observations highlight significant performance variability among NoSQL databases, indicating that the specific type of operation being performed influences the efficiency of these systems. The performance of four database management systems, MongoDB, RavenDB, Cassandra, and MS SQL Server, was evaluated in storing and retrieving synthetic key-value datasets. The datasets consisted of autogenerated pairs ($kN$, $vN$), where $kN$ is a unique key and $vN$ is its corresponding value, with N ranging up to 20 million. These data were uniformly applied across all DBMSs in a simulated parallel processing environment. The SQL database gave the best results on reading operations, but in the other operations (write, delete, create), the NoSQL databases offered better results.

## REFERENCES

[1] M. Z. Khan, F. U. Zaman, M. Adnan, A. Imroz, M. A. Rauf, and Z. Phul, "Comparative Case Study: An Evaluation of Performance Computation between SQL and NoSQL Database," *Journal of Software Engineering*, vol. 1, no. 2, pp. 14–23, Feb. 2023.

[2] R. Ramakrishnan and J. Gehrke, *Database Management Systems, 3rd Edition*, 3rd ed. McGraw-Hill, 2002.

[3] A. Silberschatz, Henry F. Korth, and S. Sudarshan, *Database system concepts*, 7th ed. McGraw-Hill, 2020.

[4] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, Dec. 2010, https://doi.org/10.1145/1773912.1773922.

[5] H. Garcia-Molina, J. D. Ullman, and J. Widom, "Stored Procedures," in *Database Systems: The Complete Book*, 2nd ed., Pearson, 2008, pp. 391–404.

[6] B. G. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments and notes," in *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research*, Iasi, Romania, Jun. 2011, pp. 1–5, https://doi.org/10.1109/RoEduNet.2011.5993686.

[7] I. I. Imran, "Enhancement of NoSQL Database Performance Using Parallel Processing," *Journal of Information Systems Engineering and Management*, vol. 9, no. 2, Apr. 2024, Art. no. 26126, https://doi.org/10.55267/iadt.07.14670.

[8] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," *SIGMOD Rec.*, vol. 22, no. 2, pp. 297–306, Mar. 1993, https://doi.org/10.1145/170036.170081.

[9] V. F. de Oliveira, M. A. de O. Pessoa, F. Junqueira, and P. E. Miyagi, "SQL and NoSQL Databases in the Context of Industry 4.0," *Machines*, vol. 10, no. 1, Jan. 2022, Art. no. 20, https://doi.org/10.3390/machines10010020.

[10] H. Fatima and K. Wasnik, "Comparison of SQL, NoSQL and NewSQL databases for internet of things," in *2016 IEEE Bombay Section Symposium (IBSS)*, Baramati, India, Dec. 2016, pp. 1–6, https://doi.org/10.1109/IBSS.2016.7940198.

[11] M. A. Kausar, A. Soosaimanickam, and M. Nasar, "Public Sentiment Analysis on Twitter Data during COVID-19 Outbreak," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 2, 2021, https://doi.org/10.14569/IJACSA.2021.0120252.

[12] M. L. E. Chang and H. N. Chua, "SQL and NoSQL Database Comparison," in *Advances in Information and Communication Networks*, 2019, pp. 294–310, https://doi.org/10.1007/978-3-030-03402-3_20.

[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)*, Gurgaon, Oct. 2017, pp. 293–299, https://doi.org/10.1109/IC3TSN.2017.8284494.

[14] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, "Comparison between relational and NOSQL databases," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, May 2018, pp. 216–221, https://doi.org/10.23919/MIPRO.2018.8400041.

[15] 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)*, Victoria, Canada, Aug. 2013, pp. 15–19, https://doi.org/10.1109/PACRIM.2013.6625441.

[16] S. Venkatraman, K. Fahd, S. Kaspi, and R. Venkatraman, "SQL Versus NoSQL Movement with Big Data Analytics," *International Journal of Information Technology and Computer Science*, vol. 8, no. 12, pp. 59–66, Dec. 2016, https://doi.org/10.5815/ijitcs.2016.12.07.

[17] M. AbuSafiya and S. Mazumdar, "Accommodating paper in document databases," in *Proceedings of the 2004 ACM symposium on Document engineering*, Jul. 2004, pp. 155–162, https://doi.org/10.1145/1030397.1030428.

[18] M. A. Almaiah, L. M. Saqr, L. A. Al-Rawwash, L. A. Altellawi, R. Al-Ali, and O. Almomani, "Classification of Cybersecurity Threats, Vulnerabilities and Countermeasures in Database Systems," *Computers, Materials & Continua*, vol. 81, no. 2, pp. 3189–3220, 2024, https://doi.org/10.32604/cmc.2024.057673.

[19] S. R. Meruva and B. Venkateswarlu, "Dynamic Association Mining Techniques for the Faster Extraction of High Utility Itemsets from Incremental Databases," *Engineering, Technology & Applied Science Research*, vol. 15, no. 1, pp. 19396–19400, Feb. 2025, https://doi.org/10.48084/etasr.9295.