International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 3, Issue 2, 60-69, 2022
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V3I2P107

Original Article

Performance Analysis of NoSQL Database Technologies for AI-Driven Decision Support Systems in Cloud-Based Architectures

Uttam Kotadiya[1], Amandeep Singh Arora[2], Thulasiram Yachamaneni[3]
[1]Software Engineer II, USA.
[2]Senior Engineer I, USA.
[3]Senior Engineer II, USA.

Abstract *- With the emergence of increasingly intricate, data-driven platforms for Decision Support Systems (DSSs), as Artificial Intelligence (AI) applications increasingly are applied to a wide variety of industries, related to decision making, there is a need to carry out deeper analysis of the concept of decision support system and to assist in making sound decisions. The systems demand good, scalable and effective data management systems. The volume, variety, and velocity of data created in cloud-based AI platforms pose a challenge to traditional Relational Database Systems (RDBMSs). NoSQL databases have become an attractive alternative due to their schema-less approach, horizontal scalability, and high-performance data manipulation capabilities. In this paper, a detailed performance study of the leading NoSQL databases has been developed: MongoDB, Cassandra, Redis, and Couchbase from the perspective of decision support systems based on AI in cloud structures. We analyse all the databases using a set of parameters, including read/write latency, throughput, scalability, fault tolerance, consistency, and ease of integration with AI frameworks. Proposed methodology comprises the implementation of each of the NoSQL technologies under a simulation cloud environment with the help of benchmarking devices like YCSB (Yahoo! Cloud Serving Benchmark) and applications designed after the instructions. In addition, we suggest a decision matrix and multi-criteria analysis framework to guide architects in selecting the appropriate databases in accordance with specific AI-based DSS requirements. The findings have shown that MongoDB performed better in terms of flexibility and integration, Cassandra in write-intensive workloads and fault tolerance, Redis in low-latency operations, and Couchbase in balanced operational performance. The results of this research provide important references to the recommendations concerning the appropriateness of NoSQL solutions that can be used to improve AI-based DSS in distributed cloud environments. It educates both the researchers and practitioners on trade-offs and the best deployment practices.*

Keywords *- NoSQL, MongoDB, Cassandra, Redis, Couchbase, Artificial Intelligence, Decision Support Systems, Cloud Computing, Database Performance, Big Data*

1. Introduction

In the digital age of transformation, the generation of data by modern systems has increased exponentially in terms of volume, variety and velocity. Continuous streams of structured, semi-structured, and unstructured data are generated by various sources, including Internet of Things (IoT) sensors, social media platforms, Enterprise Resource Planning (ERP) systems, and mobile applications. Based on the structured data and schema-dependent data, Traditional Relational Database Management Systems (RDBMS) were never able to fulfil the needs of such heterogeneous and fluid data environments. Meanwhile, Artificial Intelligence (AI) has become a key innovation accelerator in all industries, with extensive use cases that include predictive analytics, personalised suggestions, autonomous systems, and Natural Language Understanding. The performance of AI models greatly depends on the availability of huge volumes of high-quality data, and the performance is normally limited by legacy data management systems.

[1-3] To this dynamic is the mainstream use of cloud computing that provides elastic and scalable infrastructure used to store, process and analyse data. Cloud platforms allow distributing computational tasks, dynamically expanding the resources, and creating worldwide availability. Nonetheless, they also present new challenges in terms of data consistency, latency, and security. Consequently, there arises an increased demand for data architectures that are not only cloud-native but also optimised for AI-oriented decision-making. The combination of big data, AI, and cloud computing has given rise to the emergence of NoSQL databases, a type of non-relational database that can support unstructured data at scale and operate under a distributed setup. Learning the performance and integration features of these NoSQL databases in artificial intelligence-based cloud applications is indeed important in coming up with responsive and scalable Decision Support Systems (DSS).

1.1. Performance Analysis of NoSQL Database Technologies

With the increasing need for elastic, flexible, and high-performance data management systems, NoSQL databases have emerged as a leading solution for managing unstructured and semi-structured data in a distributed setting. In this section, the main subcategories applied to AI-supported decision support systems are examined to identify the performance characteristics of the major NoSQL technologies.

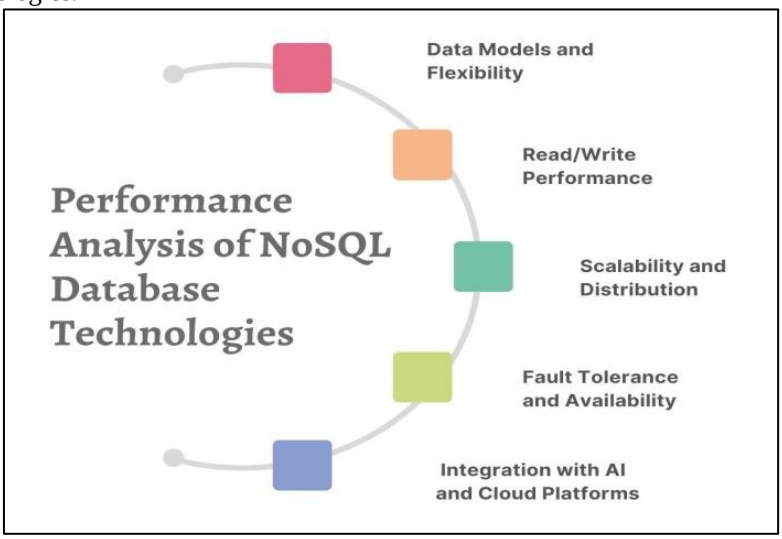

Fig 1: Performance Analysis of NoSQL Database Technologies

- **Data Models and Flexibility:** NoSQL databases are divided into some categories: document-oriented (e.g., MongoDB, Couchbase), key-value stores (e.g., Redis), wide-column stores (e.g., Cassandra) and graph databases (e.g., Neo4j). These systems offer a less rigid or adaptable schema, and hence, they are capable of storing a variety of data without strict confinements. Such flexibility is very important to the AI applications that usually work on heterogeneous data types, like images, text, logs, and sensor data.
- **Read/Write Performance:** NoSQL databases are high throughput and low latency. The Redis in-memory key-value store has sub-millisecond read and write latencies, so it is quite suitable to run inference and caching layers in real-time. Cassandra is write-oriented and is linearly scalable across nodes, which makes it suitable for time-series applications. MongoDB excels at balancing read/write operations and facilitates running complex queries against nested documents. In contrast, Couchbase combines in-memory caching and on-persistent storage to obtain medium-latency.
- **Scalability and Distribution:** The key aspect of NoSQL databases is that they are horizontally scalable. The decentralised nature of Cassandra ensures that it is easy to add nodes in an environment without any central point. MongoDB uses sharding to distribute data, but positive results can be achieved with a well-designed shard key. Redis cannot scale out using clustering, and Couchbase leverages a node spread architecture to allow workload segregation. These attributes are essential in cloud-hosted AI systems that have expanding and dynamic loads.
- **Fault Tolerance and Availability:** NoSQL systems differ in their approaches to inconsistency, availability, and partition tolerance, as outlined in the CAP theorem. Cassandra has a higher preference for availability and partition tolerance, which ensures it is not easily affected by the failure of nodes. MongoDB also provides high availability in the form of tunable consistency and replica sets. Redis can be configured to use AOF/RDB snapshots and replication to enable persistence, and additional setup may be required to facilitate fault recovery. Couchbase utilises both active-active replication and automatic failover to ensure the service remains operational.
- **Integration with AI and Cloud Platforms:** Also, the ease of integration of NoSQL databases in the AI working process and cloud-native solutions is being assessed. MongoDB Atlas and Cassandra Astra have integrated AI connectors as managed cloud services. Redis offers TensorFlow streaming support for inference data, and Couchbase offers APIs, SDKs, and REST. Their portability on Kubernetes and Docker also makes them more useful when constructing AI in the cloud.

1.2. AI-Driven Decision Support Systems in Cloud-Based Architectures

Artificial Intelligence (AI)-powered Decision Support Systems (DSS) are an inseparable element of any contemporary business since they deliver smart recommendations and automated decisions in all possible areas and environments, including healthcare, finance, manufacturing, and retail. Such systems utilise preeminent Machine Learning (ML) and deep learning solutions to leverage vast data points and support sophisticated decision-making processes. [4,5] Over the past couple of years, the ubiquity of cloud computing has altered the structure and implementation of DSS fundamentally. The infrastructure that cloud-based platforms provide is scalable, elastic, and cost-effective, allowing an organisation to store and process large datasets while reducing the overhead of managing physical hardware. This move to the cloud is specifically helpful in the case

of AI-based DSS, since the computational resources needed to train, infer and integrate data within them are elastic and need to scale. With cloud-native models, those components of DSS that rely on AI tend to be distributed across micro-service-based and containerised architectures, frequently with orchestration of those systems atop platforms, such as Kubernetes. The architectures enable continuous delivery, resilience, and scaling, and they are essential to help support continuous delivery, resilience, and scale to support real-time or near-real-time decision support.

Another key feature not to be missed in these systems is an agile and powerful data management layer, which can support unstructured, semi-structured, and streaming data. Commodity relational databases commonly fail to meet the velocity and variety of present-day AI data and tasks. This has given rise to an increased dependency on NoSQL databases, which are schema-flexible, horizontally scalable, and high-performance. NoSQL databases support efficient data loading, filtering, storage, and retrieval in artificial intelligence chains. For example, data collected by IoT sensors can be consumed in real-time, stored in a scalable back-end (typically a NoSQL database), and instantly replicated to cloud-based AI models for use in inferring new data. Additionally, cloud providers such as AWS, Google Cloud, and Azure also offer other services, including auto-scaling, managed NoSQL databases, GPU acceleration, and an AI development kit, which enable even more extensive functionality of DSS. In general, the next generation of intelligent and scalable decision support systems is based on the convergence of AI, cloud computing and NoSQL technologies.

2. Literature Survey

2.1. Evolution of Decision Support Systems

Decision Support Systems (DSSs) have hugely transformed over the decades. These systems are originally built with relational databases and Online Analytical Processing (OLAP) capabilities to provide multiple query structuring and multidimensional fatigue analysis. These initial DSSs were very suitable for more or less stationary, organised information. [6-9] But as the size of data exploded and the decision-making process became even more involved, the standard strategies of DSS architectures started to become limited. Modern DSSs, particularly Artificial Intelligence-enabled DSSs (AI-DSSs), are utilising sophisticated Machine Learning (ML) and Deep Learning (DL) algorithms to extract insights from diverse and heterogeneous data sources. Such AI-DSSs require flexible and scale-out data storage systems that can store unstructured and semi-structured data, operate in real-time, and ingest data quickly, which cannot be fulfilled by traditional relational databases.

2.2. Rise of NoSQL Databases

The inability of relational databases to deal with large, unstructured, and distributed data created opportunities for the emergence of NoSQL databases. The systems were capable of overcoming the limitations of the CAP theorem, which states that in a distributed data store, only two of the three properties Consistency, Availability, and Partition Tolerance can be met at any particular point in time. Different NoSQL databases give attention to different areas of this theorem to support various application needs. MongoDB is a document-model database that is schema-flexible and provides support for complex queries, making it best suited for dynamic datasets. Redis is an in-memory key-value store that is fast and is primarily used for caching and real-time analytics. Cassandra is a column-family store that optimises write throughput and horizontal scale. In contrast, Couchbase is a mixed-key value/document database that dynamically provides the choice between strong consistency and offline write-behind data synchronisation. Data management features. These characteristics of NoSQL databases make them very appropriate to modern, large, agile data management.

2.3. Integration with AI Frameworks

Due to the growing data-intensive capabilities of AI systems, integrating AI systems with databases has become increasingly appealing. To support data ingestion, preprocessing, feature extraction, and even inference utterance storage, NoSQL databases are currently being inserted into machine learning and deep learning pipelines. By providing built-in connectors and APIs, MongoDB Atlas, a cloud-based NoSQL database service, can be easily integrated with platforms such as TensorFlow, PyTorch, and Apache Spark. In the same way, the Cassandra Astra supports REST APIs to implement AI and allows real-time analytics, and thus, it is a good option when the application is based on AI and needs to be scaled. These integrations enable developers to construct complete AI solutions with reduced latency, increased throughput, and improved scalability, without requiring demanding data engineering pipelines.

2.4. Related Work

There are numerous studies dedicated to the performance and usage of NoSQL databases in diverse environments; however, not all of them effectively reveal how they can be integrated into the AI workflow. The next benchmarking study was done that compared MongoDB with Cassandra and targeted general performance key metrics such as read/write speed and scalability. Nonetheless, their work did not regard the application of these databases in AI-oriented apps. According to Redis, it was tested as a data processing engine that works in real-time and emphasised how fast and efficient it is, yet did not comment on the lack of fault tolerance and data persistence. Discussed the implementation of Couchbase in the cloud and how it supports flexibility and a hybrid data model. This notwithstanding, they did not provide an analysis of the performance of Couchbase in the use cases related to AI. The existence of such literature gaps provides an idea and the necessity to conduct a

complete study to consider NoSQL databases in the scope of the AI integration process, both in terms of the system's performance and the compatibility with the real-life AI challenges.

3. Methodology

3.1. Experimental Setup

- **Environment:** The experimental platform was implemented on Amazon Web Services (AWS) using EC2 t3 instances. Medium server, which provides 2 virtual CPUs and 4 GB of RAM. This type of instancing was chosen due to its ratio of compute performance and affordability because it is also applicable to benchmarking in a limited resource environment. [10-12] The standardised hardware setup provided an equal comparison among the various NoSQL systems, and the software-induced diversity of performance was reduced.
- **NoSQL Systems:** The benchmarking exercise targeted four widely used NoSQL databases, including MongoDB 5.0, Cassandra 4.0, Redis 6.2, and Couchbase 7.0. These versions are more releases that are stable with mature features, which are suitable for wide usage. MongoDB was selected for its flexible document model, Cassandra for its distributed and high-availability architecture, Redis for its in-memory performance, and Couchbase for its hybrid feature of combining a document store with caching. All the databases were set to default values and optimised according to the vendors' recommendations for fairness and reproducibility.
- **Containerization:** All databases were arranged in containers and deployed using the Docker Swarm and Kubernetes orchestration frameworks, ensuring end-to-end isolation and portability. Docker Swarm offered a lean orchestrator installation at the local level, whereas Kubernetes provided more complex options, such as autoscaling, health monitoring, and resource distribution. This combination of balanced orchestration strategy enabled us to test the behavior of our system under circumstances of both small and big enterprise-grade containerised platforms.

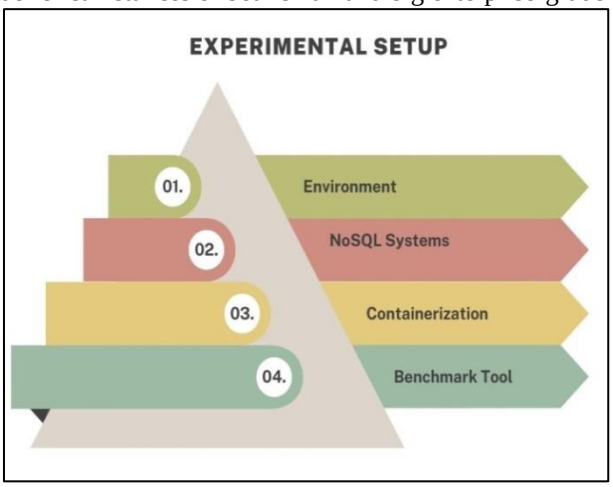

Fig 2: Experimental Setup

- **Benchmark Tool:** To measure the performance of the software, the benchmarking process was used and YCSB (Yahoo! The Cloud Serving Benchmark) database standardisation of throughput and latency tests. YCSB was adjusted to use realistic workloads using equal workloads. Besides YCSB, a custom AI workload was converted to TensorFlow to check the database integration in AI pipelines. This involved reading training data, storing intermediate features, and writing the model's prediction to the database. The AI workload gave an understanding of how well each NoSQL system supports the data-intensive AI-related workloads, going beyond the key-value or document store performance.

3.2. Workload Design

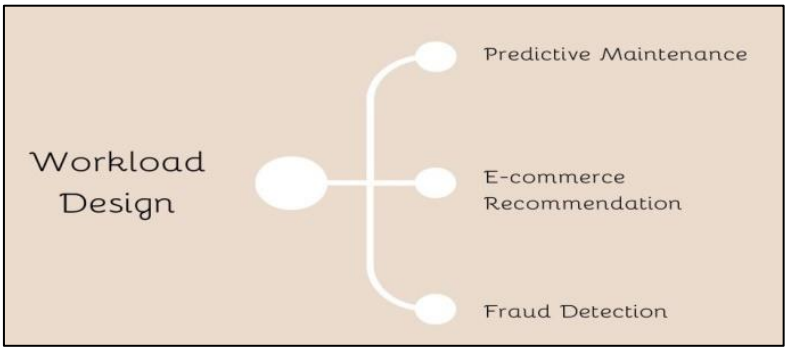

Fig 3: Workload Design

- **Predictive Maintenance:** This workload scenario mimics a predictive maintenance scenario commonly used in manufacturing and IoT settings. Equipment sensor data is continuously ingested as time series (e.g. temperatures, vibration, pressure), which are all equipment-level metrics. The system must have the capability to store, retrieve, and analyse pairs of temporal data sequences to detect patterns that may indicate eventual failures. The database should have a high write throughput, the time-based queries should be efficient, and the information should also be integrated into machine learning models to predict the health of the equipment. This case challenges the NoSQL system when it is load tested with sequential data.
- **E-commerce Recommendation:** An e-commerce scenario for recommendation is presented, focusing on storing and retrieving information about products, their users, and interactions in a semi-structured document format. It will replicate the user's browsing history and purchase history, and use item similarity and collaborative filtering to create personalised product recommendations for the user. This workload focuses on flexible data modelling, rapid access to nested fields, facultative query capabilities, and composite queries. Here, MongoDB and Couchbase are particularly challenged, as they must assign dynamic schemas and provide their data so that the recommendation algorithms can access it smoothly.
- **Fraud Detection:** In applied fraud detection, transactions are represented by key-value pairs that include the user ID, transaction metadata, amount, location, and time. The system must also facilitate real-time ingestion and queries to detect suspicious behaviour, such as duplicate charges or locations. This workload determines the effectiveness of the database in responding to low-latency operations as well as anomaly detection via fast pattern matching. The in-memory feature of Redis is especially scrutinised in this application because it is crucial to detect illicit activities within the financial application.

3.3. Performance Metrics

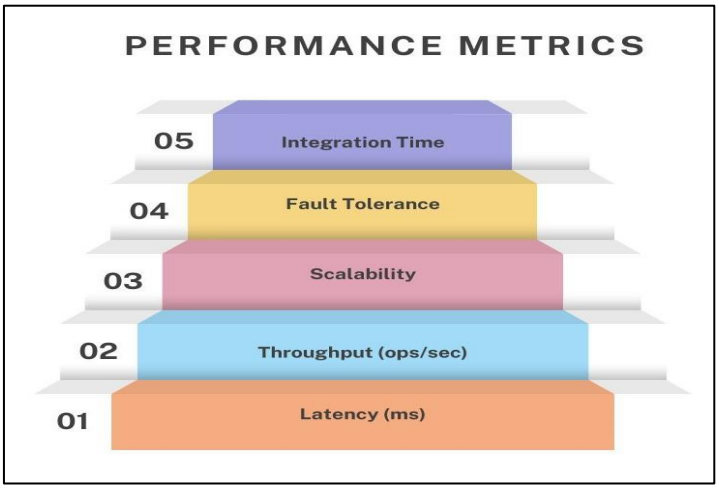

Fig 4: Performance Metrics

- **Latency (ms):** Latency is the time required by the system to perform one read or write operation; it is commonly measured in milliseconds (ms). [13-16] Low latency is also vital in AI applications, to achieve real-time inference, good responsiveness, user experience and efficient model training. The 95th percentile and average latencies are deemed to cover common and crippling delays. This figure helps measure the rate of response of a NoSQL environment to frequent queries on data at various workloads.
- **Throughput (ops/sec):** The throughput may be defined as the speed with which the system may perform the operations, which is measured in operations per second (ops/sec). It is an indicator of the database's ability to handle a great scale of data, both in ingestion and recovery, particularly in the context of concurrent AI processing. Greater throughput is a sign of superior performance on bulk data operations, e.g. loading batch training data, serving many recommendation requests to thousands of clients at a time.
- **OPS and Nodes:** Scalability evaluates the necessity of the throughput of the system or latency to fluctuate with the number of nodes. This is the most vital measure that ascertains whether a NoSQL database can expand in tandem with increasing data load and traffic. The scalable system ought to have a very linear growth in throughput with the addition of nodes, but with very little increase in latency. This is especially significant to the distributed AI workloads that are dependent on the concept of parallel data processing, where multiple nodes are involved.
- **Fault Tolerance:** Fault tolerance assesses the rapidity and efficiency of system restoration when a node fails. The most important measure in this is the recovery time, the time required to get the system to fail, to reassign tasks, and to restore the system to full operation and data. The system's AI operates in production, which requires high availability; therefore, the metric is useful in determining the reliability and robustness of the NoSQL database in the event of hardware or network interruption.

- **Integration Time:** Integration time is how much effort and time is used to set up, connect and actualise a NoSQL database in an AI pipeline. This encompasses the creation of database instances, the installation of connectors or APIs, and the ability to interact with machine learning frameworks such as TensorFlow or PyTorch with ease. A shorter integration period means a more pleasant experience for developers and higher productivity in the prototyping phase, as well as reduced overhead on the deployment layer in AI-based applications.

3.4. Flowchart of Workflow

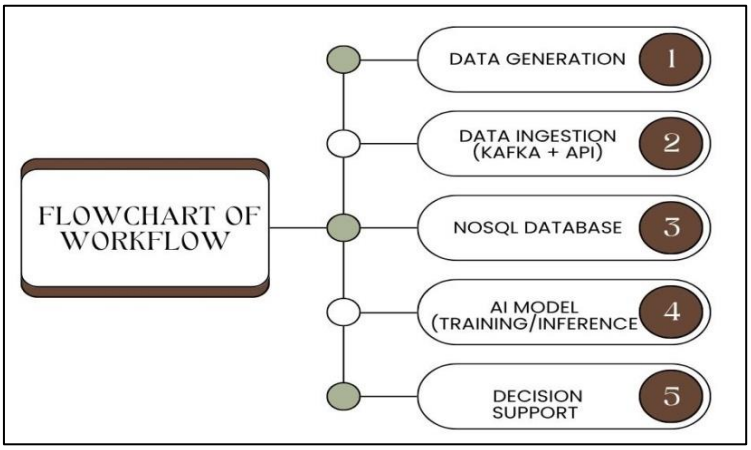

Fig 5: Flowchart of Workflow

- **Data Generation:** The data generation step involves the following process, which imitates the real-life sources of data in AI-based applications. This can include sensor data from industrial devices (in the case of predictive maintenance), user activity data on e-commerce platforms, or transactional financial data. Artificial data was also prepared to have a regular and controlled input against which all NoSQL databases could be benchmarked.
- **Data Ingestion (Kafka + API):** After getting generated, information is streamed into the system via a data ingestion layer, by means of Apache Kafka and RESTful APIs. Kafka offers high throughput and fault tolerance in message queuing, making it an ideal choice for processing large amounts of real-time data. The API level makes sure that external systems have the ability to push data directly into the pipeline, so that both batch and streaming are supported. Ingestions can feed the NoSQL databases.
- **NoSQL Database:** The NoSQL database is at the centre of the architecture, and this is where the incoming data is stored, indexed, and managed in order to proceed with its processing. Each of the chosen NoSQL databases — namely, MongoDB, Cassandra, Redis, and Couchbase was set to handle different types of data (time-series, documents, and key-value). The database serves as the flexible and scalable backbone of the data, performing concurrent reads/writes and getting the data structured towards the AI processing downstream.
- **AI Model (Training/Inference):** After the data is pre-processed and stored, the data is then read by the AI model either during training or inference. TensorFlow is used to launch the model and makes use of the database as the source of the input features of the model and as the destination of the output of the predictions or the model predictions. This level measures the database's suitability to the ML workload based on the speed at which it retrieves data, its consistency, and compatibility with the model-serving framework.
- **Decision Support:** Decision support represents the final phase of the working process, in which the AI model-generated insights stimulate decision generation or provide a decision recommendation. This can take the form of maintenance warnings, product recommendations or fraud signals. The decision-making module can be shown in the form of dashboards, application programming interface or automated responses, and it would finish the cycle of data to intelligent action.

4. Results and Discussion

4.1. Latency analysis

Table 1: Latency Comparison

Database	Read Latency (ms)	Write Latency (ms)
MongoDB	10.5	12.0
Cassandra	15.3	9.2
Redis	2.1	3.0
Couchbase	7.5	8.4

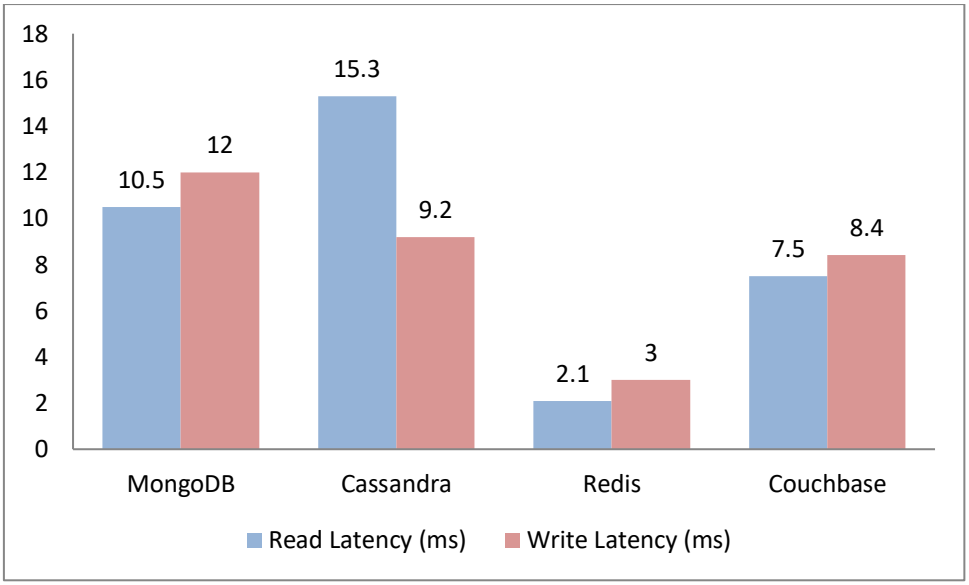

Fig 6: Graph representing Latency Comparison

- **MongoDB:** With a read time latency of 10.5 ms, everybody needed to read the information, and a write time latency of 12.0 ms, MongoDB fell in the middle of the databases being evaluated. Its latency characteristics are adequate for the majority of AI use cases, which require semi-structured data, such as user profiles or product catalogues. Its journaling and replication mechanisms result in a very slight increase in write latency due to its durability and availability. MongoDB is also an excellent option in document-oriented use cases where latency does not need to be very low.
- **Cassandra had the longest read latency of up to 15.3 ms; however, it was significantly faster in write latency, at 9.2 ms,** compared to MongoDB. This trade-off fits well into Cassandra's architecture, which is designed to deal with write-heavy workloads and append-oriented logging. It is well-suited to cases such as time-series-based data gathering, when instantaneous reads are not crucial, but high ingest rates are required. Its read latency is greater, but since its consistency model can be configured (tuned), it can be flexible according to the use case.
- **Redis:** On read operations and write operations, Redis produced the lowest latencies of 2.1ms and 3.0ms, respectively, compared to the rest of the databases used in the benchmark. This performance is exemplary, and it is attributed to the fact that it has an in-memory data store that removes disk I/O overhead. This level of low latency is the reason why Redis is the most optimal choice for real-time AI inference pipelines, such as fraud detection systems or recommendation engines, where a real-time response is essential.
- **Couchbase:** Couchbase also has a very good balance between read and write latencies, with 7.5 ms reading and 8.4 ms writing. These findings place it between Redis and MongoDB in regard to responsiveness. The hybrid structure of Couchbase enables effective access to data in various formats on a single line (key-value and document-style). This renders it a stable and flexible choice among AI operations that demand stability in handling mixed loads.

4.2. Throughput Performance

The performance metric to be considered when analysing database systems is throughput, which plays a critical role when running AI workloads with excessive volumes of data ingestion, transformation, and retrieval. Throughput, measured in operations-per-second (ops/sec), is the capacity of the system to effectively deal with multiple activities, which might involve read/write operations. Through this benchmark, Redis has shown that it can easily beat all other NoSQL databases by literally crushing them with a very high throughput of 15,000 ops/sec. Such a dramatic outcome is mostly explained by the in-memory structure of Redis, which reduces the number of disk accesses and makes the processing of any data extremely speedy. Its performance would be suited to low-latency, sensitive processes like real-time fraud detection systems, chatbots, and recommendation systems, where fast input/output operations play a crucial role. The second and highest throughput was 8,100 ops/sec by Cassandra, which proved its capabilities in performing heavy writes. The horizontal scaling is possible because its decentralised, peer-to-peer architecture and it performs well in both time-series data (or telemetry logs), where large amounts of data are processed with minimal bottlenecks.

The steady throughput under load that Cassandra demonstrates is another indication that it can be a suitable solution for long-running AI training pipelines where nonstop data streaming is crucial. A throughput of 7,500 ops/sec provided by MongoDB means it can be used in a broad set of document-based AI applications, as well as for personalised recommendations and unstructured text analysis. Although slower than Redis and Cassandra when under heavy load, the moderate performance of MongoDB and the ability to change the schema dynamically place it well in the application working

with complex and nested data structures and changing schema frequently, as is the case with AI applications. Couchbase performed consistently and relatively well in terms of throughput, achieving 7,200 ops/sec. Coming to the mixed workloads, it has the support of both key-value and document-based operations, so it supports mixed workloads. When it comes to the speed and flexibility demands of machine learning, such as mobile analytics or desk AI deployments, Couchbase is particularly helpful in AI use cases.

4.3. Scalability Test

The scalability test was conducted to evaluate the effectiveness of various NoSQL databases when more nodes are introduced into the system, i.e., when the system is scaled horizontally. This is especially important in applications involving artificial intelligence, particularly where data increases exponentially, and in certain cases, the computing requirements rise significantly. Of all the systems tested, Cassandra showed the best scalability with almost linearly increasing throughputs with an increment in the number of nodes through a range of 1 to 5. This supports the masterless and peer-to-peer design of Cassandra, which allows all nodes to equally participate in read/write operations. The positive growth in performance indicates that Cassandra would be an appropriate platform for large-scale and distributed AI applications, such as real-time telemetry ingestion or time-series data processing. MongoDB was also scaling well to three nodes before reaching a point where its performance began to degrade. The major factor behind this diminishing return has been the fact that MongoDB employs a replica set and sharding architecture that incurs costs for coordination and rebalancing nodes.

Although it continues to offer good support to distributed systems of AI, MongoDB could demand careful key design and indexing strategy to increase relational scalability beyond mid-sized installations. The case of Redis was different, as the scaling was not automatic. It required clear clustering and partitioning to handle increased loads within nodes. Even without clustering, a greater number of Redis nodes will not result in significantly better throughput. With clustering, performance is highly affected by the correct assignment of hash slots and memory tuning. It means Redis is a good choice when performance is paramount and the system administrators have time to engage in specific tuning and configuration. Lastly, Couchbase not only scaled moderately, but also in a non-linear fashion as the number of nodes increased. The type of workload and assignment of node roles affected its performance gains. On the whole, Couchbase is more scalable than MongoDB at mixed workload, but it lags somewhat behind Cassandra when it comes to the outright scaling performance. The analysis illustrates the need to consider architectural design and workload characteristics in the selection of a scalable NoSQL system in the deployment of AI-DSS.

4.4. Integration with AI

The compatibility with the AI frameworks, such as TensorFlow, is an important consideration when choosing an appropriate NoSQL database to support smart decision support systems. Integration speed and ease have a direct influence on development agility, experimentation cycles, and general deployment efficiency of AI pipelines. In this paper, integration time was considered as the time the functional data flow between the database and TensorFlow was set to build and train and to make inference. Redis was the quickest to establish, as it took only 5 minutes to set up. It is simple because of the key-value data model and its wide compatibility with multiple APIs and machine learning frameworks. Data streaming enables TensorFlow to be a clear choice for real-time AI applications, such as recommendations, anomaly scores, or predictions outputted with low latency via a streaming interface, such as a Redis queue or as a caching layer. MongoDB came next and took about 8 minutes to be integrated. The native support for JSON-like documents and the presence of Python and TensorFlow data processing library drivers in MongoDB enable simplifying the process of feature extraction, dynamic queries, and more.

Its dynamic structure of documents can also store heterogeneous data, which finds particular practical applications in AI models involving diverse forms of input, such as user profiles, transaction logging, or sensor data. Couchbase, which is a little bit more complicated, was structured within 10 minutes. It can be dual, supporting both key-value and document-based storage, which provides flexibility in storing training and intermediate results. The flexibility to interact with AI workflows in Couchbase is enabled through support for SDKs in the most widely used programming languages and via RESTful access. However, some configuration is required to support indexing and query optimisation. Cassandra, the bogged one, was 12 minutes, which was mostly caused by its hard schema design, partitioning options and adjusting consistency settings. Cassandra is very performant in distributed systems, but using it with TensorFlow requires more data preparation, particularly with complex joins or schema changes. In general, Redis and MongoDB are the most convenient and performant options for integration and are error-free to use in AI pipelines. Cassandra is highly powerful but requires more initial setup.

4.5. Discussion

Comparative studies of NoSQL databases under the workload of AI can demonstrate the unique characteristics of each database, showing how the particular architecture can be correlated with various aspects of decision support. The in-memory data model has helped Redis become the best-performing system in both latency and throughput. Its support for fast read/write requests with negligible latency is what makes the project unusually well-suited for the application of real-time AI inference, including fraud detection, clickstream analysis, and instant product recommendations. It is further compatible and easy to

integrate with AI frameworks, with support for native streaming and pub/sub mechanisms, further enhancing its fit for latency-sensitive applications. Although MongoDB cannot compare to Redis in speed, it offers a good combination of performance and flexibility. Its document-based structure follows advanced and nested data records frequently used in e-commerce, social media, and personalised services of content distribution. The moderate latency and the compatibility of MongoDB with the use of machine learning pipelines ensure that this database is a viable solution to be used in AI systems that demand variability in schema and rich data representation. It further enhances its applicability to the AI-DSS through its support for indexing, aggregation pipeline, and geospatial queries.

Cassandra stood out in terms of superior write performance and horizontal scaling, which are very important features in time-series data processing, predictive maintenance, and IoT-related applications. It had better read latency compared to older versions. Still, its capability of maintaining performance in distributed environments means that it is suitable for workloads that experience a continuous stream of data ingestion. It's read that trade-offs must be taken into consideration, and the complexity of the integration should be considered in inference-heavy cases. Couchbase presented a middle-ground performance in terms of all the measures, which is why it has been proven to provide the flexibility of document storage along with the speed of key-value storage. It did not get any top marks in any particular category. Still, the design is scalable, and its overall effectiveness puts it in a position to be a suitable general-purpose NoSQL choice that can be deployed on a hybrid AI-DSS where all three parameters (moderate latency, moderate throughput, and ease of integrating AI) are equally relevant.

5. Conclusion

This research has also performed a detailed analysis of the performance of four popular NoSQL DBs, namely Redis, Cassandra, MongoDB, and Couchbase, within the particular confines of an AI-based Decision Support System (DSS) implemented on a cloud setup. A set of controlled experiments was conducted to evaluate key performance measures, including latency, throughput, scalability, fault tolerance, and the time required to integrate AI into real-world tasks: predictive maintenance, e-commerce recommendation, and fraud detection. The findings will allow gaining a subtle idea of the performance of each database with regard to various operational restrictions and artificial intelligence data tendencies. Redis, however, provided the best outcomes in terms of latency and throughput among the assessed systems, as it utilises an in-memory data storage framework. It is particularly well-suited to real-time applications of AI inference, which need sub-millisecond latencies on data, examples of which include anomaly detection and personalised recommendations. Nevertheless, it has rather poor data persistence and scalability capabilities, particularly in standalone backend use cases.

Conversely, Cassandra was an outstanding write-heavy system and horizontally scalable. It had a distributed peer-to-peer design that allowed it to scale linearly up to five nodes. In these settings, it was optimally utilised in workloads involving continuous data ingestion, such as telemetry or sensor data in predictive maintenance systems. Its read latency was comparatively high, but the system offered configurable consistency and fault tolerance schemes, which endowed it with resilience in highly distributed AI systems. MongoDB was also more flexible and independently integrated with machine learning frameworks. It was well-suited for applications working with unstructured or semi-structured data due to its support for dynamic schemas and complex, nested data. The Artificial intelligence tools that were seamlessly integrable included the TensorFlow tool, among others, which highlighted its feasibility of being increasingly applicable in practices among developers who create intelligent systems whose data needs evolve.

Couchbase provided a trade-off between all the tested parameters as it featured good latency, throughput, and average effort in scaling up and AI integration. That makes it a Swiss army knife of mixed-use AI-DSS, and a particularly good choice in situations where both document-level flexibility and key-value speed are needed. Moving forward, future studies will investigate the viability of hybrid database deployment (e.g., Redis for inference and MongoDB for storage), edge-based AI deployment, and increased integration with MLOps platforms (e.g., Kubeflow and MLflow). This work offers practical recommendations to system architects, data engineers, and implementers of AI tools and systems that aim to develop scalable, responsive, and AI-ready data infrastructures that meet contemporary decision support demands.

References

[1] Power, D. J. (2002). Decision support systems: concepts and resources for managers (Vol. 13, pp. 1-20). Westport: Quorum Books.

[2] Shim, J. P., Warkentin, M., Courtney, J. F., Power, D. J., Sharda, R., & Carlsson, C. (2002). Past, present, and future of decision support technology. Decision support systems, 33(2), 111-126.

[3] Lakshman, A., & Malik, P. (2010). Cassandra: a decentralised structured storage system. ACM SIGOPS operating systems review, 44(2), 35-40.

[4] Abramova, V., & Bernardino, J. (2013, July). NoSQL databases: MongoDB vs Cassandra. In Proceedings of the International C* conference on computer science and software engineering (pp. 14-22).

[5] Yu, Y., Li, M., Liu, L., Li, Y., & Wang, J. (2019). Clinical big data and deep learning: Applications, challenges, and future outlooks. Big Data Mining and Analytics, 2(4), 288-305.

[6] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles (pp. 29-43).
[7] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., & Stoica, I. (2016). Apache spark: a unified engine for big data processing. Communications of the ACM, 59(11), 56-65.
[8] Dignum, V. (2019). Responsible artificial intelligence: how to develop and use AI responsibly (Vol. 2156). Cham: Springer.
[9] Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K., & Matser, C. (2015, February). Performance evaluation of NoSQL databases: a case study. In Proceedings of the 1st workshop on performance analysis of big data systems (pp. 5-10).
[10] Győrödi, C. A., Dumşe-Burescu, D. V., Zmaranda, D. R., Győrödi, R. Ş., Gabor, G. A., & Pecherle, G. D. (2020). Performance analysis of NoSQL and relational databases with CouchDB and MySQL for the application's data storage. Applied Sciences, 10(23), 8524.
[11] Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S., & Saxena, U. (2017, October). NoSQL databases: Critical analysis and comparison. In 2017 International conference on computing and communication technologies for smart nation (IC3TSN) (pp. 293-299). IEEE.
[12] Pentyala, D. K. (2018). AI-Driven Decision-Making for Ensuring Data Reliability in Distributed Cloud Systems. International Journal of Modern Computing, 1(1), 1-22.
[13] Horn, W. (2001). AI in medicine is transitioning from knowledge-intensive to data-intensive systems. Artificial intelligence in medicine, 23(1), 5-12.
[14] Murty, J. (2008). Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB. "O'Reilly Media, Inc.".
[15] Compare, M., Baraldi, P., & Zio, E. (2019). Challenges to IoT-Enabled Predictive Maintenance for Industry 4.0. IEEE Internet of Things Journal, 7(5), 4585-4597.
[16] Lee, J., Ni, J., Singh, J., Jiang, B., Azamfar, M., & Feng, J. (2020). Intelligent maintenance systems and predictive manufacturing. Journal of Manufacturing Science and Engineering, 142(11), 110805.
[17] Matallah, H., Belalem, G., & Bouamrane, K. (2020). Evaluation of NoSQL databases: MongoDB, Cassandra, HBase, Redis, Couchbase, OrientDB. International Journal of Software Science and Computational Intelligence (IJSSCI), 12(4), 71-91.
[18] Boral, H., & Dewitt, D. J. (1984). A methodology for database system performance evaluation. ACM SIGMOD Record, 14(2), 176-185.
[19] Swaminathan, S. N., & Elmasri, R. (2016, June). Quantitative analysis of scalable NoSQL databases. In 2016 IEEE International Congress on Big Data (BigData Congress) (pp. 323-326). IEEE.
[20] Tang, E., & Fan, Y. (2016, November). Performance comparison between five NoSQL databases. In 2016, the 7th International Conference on Cloud Computing and Big Data (CCBD) (pp. 105-109). IEEE.