ISSN: 2249-7196

IJMRR/Oct 2020/ Volume 10/Issue 4/38-49

Karthikeyan Parthasarathy/ International Journal of Management Research & Review

REAL-TIME DATA WAREHOUSING: PERFORMANCE INSIGHTS OF SEMI-STREAM JOINS USING MONGODB

Karthikeyan Parthasarathy

LTI Mindtree, Tampa, FL, United States

Email ID: karthikeyan11.win@gmail.com

ABSTRACT

The speed of processing of MongoDB for real-time data warehousing is examined in this work, with a special emphasis on semi-stream join processing during the extraction, transformation, and loading (ETL) stage. Decision-making can be slowed down by traditional data warehouses' frequent problems with timely updates and rapid data retrieval. With speedier data access and ongoing updates, real-time data warehousing seeks to overcome these problems. We assess MongoDB's efficiency in processing structured and unstructured data streams, and our tests show that it can handle high-velocity data while maintaining constant memory and CPU utilisation. The results show that MongoDB works effectively in real-time data contexts, handling different kinds and amounts of data without causing appreciable performance reduction.

Keywords: Real-time data warehousing, MongoDB, semi-stream joins, ETL (Extraction-Transformation-Loading), NoSQL databases, structured data.

1 INTRODUCTION

Enterprises seeking to derive valuable insights from their data have long found that data warehousing is indispensable. Traditional data warehouses, on the other hand, frequently struggle to deliver business intelligence in a timely manner because to their irregular update cycles. This might impede the ability to make quick decisions and slow down data recovery during transformation or loading issues. These issues can be resolved by employing real-time data warehousing, which enables quicker data retrieval and constant updates. Stream processing of unstructured (NoSQL) and structured (SQL) data from several sources must be done efficiently for this to work. In this paper, we provide a thorough analysis of semi-stream join processing utilising MongoDB, a potent NoSQL database engine, during the Extraction-Transformation-Loading (ETL) stage, between unstructured and structured data. Our approach uses the MongoDB server to combine semi-stream tuples from various sources with disk-based master data, utilising in-memory keys for both structured and unstructured documents. The efficiency of data processing is impacted by the different I/O rates of these inputs.

To evaluate the CPU and memory consumption related to real-time semi-stream join processing, we ran a number of experiments. We used both artificial and actual datasets to analyse two different kinds of data streams: unstructured and structured. Results show that when the number of incoming semi-streams increases, memory use and execution time stay constant for a particular specification, irrespective of the type of data streams. As organisations aim to integrate various decision-support environments to avoid contradicting information, the significance of data warehouses (DW) has increased. All data sources are integrated by a DW, providing a single framework for decision-making. Three essential stages make up the architecture of decision-support systems: exporting, DW creation, and ETL operations, which are necessary for loading, extracting, and transforming data into a multidimensional DW.

The target system's structure needs to be known before developing an ETL system for a conventional DW. On the other hand, because NoSQL databases lack schemas, they require the current ETL tools to be extended in order to create schemas while integrating data. ETL's main objectives are to combine data for fast report generation and remove

IJMRR

redundant information that isn't needed for analytical reports. Several concurrent users usually execute Online Transaction Processing (OLTP) workloads in the course of regular business activities. These workloads involve random data access for tasks such short searches, inserts, updates, or deletions. There are two types of data sources for a data warehouse (DW): internal and external. This is particularly true for ERP systems. The heterogeneous data retrieved from numerous operating systems is temporarily stored in large data files called "streams." After reformatting and reorganising, the streams are relocated to a staging area.

An alternative to the conventional relational paradigm, which arranges data into distinct tables with rows and columns, is provided by NoSQL databases. On the other hand, NoSQL databases offer flexible data organisation, such document-based and key-value stores. Dynamic schemas and tagged items are stored in document-based databases, such as MongoDB, which makes them extremely effective for search and filtering tasks. On the other hand, huge hash tables are used by key-value stores to provide quick access to data. Leading NoSQL database MongoDB is well-known for its collections of schema-free documents, which are made up of key/value pairs. Because of its adaptability, MongoDB can effectively manage complicated queries and high-performance data structures. NoSQL databases provide scalable solutions that work better for real-time data warehousing, especially with massive volumes of semi-structured or unstructured data, in contrast to relational databases, which rely on SQL and intricate join queries.

Variations in data objects can arise from the lack of a consistent schema in semi-structured data, including XML and web page data. Because NoSQL DWs are schema-free, they are especially well-suited for handling this kind of data. The complexity of these procedures has restricted research on ETL procedures for NoSQL warehousing. In order to properly handle schema-free and occasionally ambiguous data streams, document-oriented NoSQL solutions must be used in place of conventional Relational Database Management Systems (RDBMS), given the growing demand for real-time data warehousing.

- During the ETL stage, evaluate MongoDB's ability to manage semi-stream join processing for both structured and unstructured data.
- Analyse the CPU and memory consumption of semi-stream join processing in real-time with both simulated and actual datasets.
- Find out if MongoDB performs consistently with an increase in the amount of incoming semi-stream data.
- Examine if MongoDB can maintain constant execution times and memory use for a variety of data streams.
- Provide information about NoSQL databases' scalability for real-time data warehousing.

The performance and efficiency of NoSQL databases like MongoDB for real-time semi-stream join processing are not well studied, despite the fact that traditional SQL-based data warehousing has received a lot of attention. The majority of previous research ignores the topic of how schema-free NoSQL databases handle real-time management of growing volumes of semi-structured and unstructured data. This paper aims to close that gap by offering a thorough evaluation of MongoDB's performance under these conditions. Due to their long update cycles, traditional data warehouses frequently find it difficult to enable real-time business information, which can impede decision-making and data recovery. Real-time data warehousing relies on the efficient processing of both structured and unstructured data streams. To make sure MongoDB can properly manage the demands of various and expanding data streams, this study focuses on testing its performance in real-time semi-stream joins, with a focus on CPU and memory utilisation.

2 RELATED WORK

Real-time data warehousing provides the potential to greatly enhance business intelligence (BI), as examined in the study by Farooq & Sarwar (2010). In order to make prompt, well-informed judgements, they must overcome the difficulties of digesting and integrating data on the fly. Real-time data handling is being revolutionised by emerging technologies like in-memory databases and streaming data platforms, as the writers highlight. They stress how adopting advanced analytics to improve BI and having quick access to data are crucial. The study also emphasises the

IJMRR

necessity of robust data security and governance procedures to guarantee data quality and regulatory compliance. Practical advice on how to successfully implement real-time data systems is provided by the research.

The Extract, Transform, and Load (ETL) procedures in almost real-time data warehousing are the subject of Wibowo (2015) research, which explores the problems and potential solutions. The study describes potential problems such performance bottlenecks and delays in data processing, as well as the challenges of integrating several data sources. Wibowo proposes leveraging various optimisation approaches, parallel processing, and cutting-edge streaming technologies to streamline ETL procedures in order to address these issues. The use of distributed computing to manage massive data volumes, the use of in-memory processing to reduce latency, and the maintenance of data accuracy and consistency are among the primary recommendations. All things considered, the research offers helpful guidance for raising the effectiveness and dependability of ETL operations in situations that include almost real-time.

The study "Towards Near Real-Time Data Warehousing" by Chen et al. (2010) is centred on improving data integration and reducing latency to facilitate rapid decision-making. To improve data handling, they point to the utilisation of distributed systems, in-memory processing, and real-time data streaming as crucial tactics. Advanced data integration techniques and ETL process optimisation are suggested as solutions to the problems of maintaining data consistency and quality while handling massive data volumes. The project attempts to improve the effectiveness and performance of near real-time data warehousing systems by utilising scalable computing resources and cutting-edge technology.

The framework for managing and analysing big data in real-time is presented in Ali (2018) paper, "Real-Time Big Data Warehousing and Analysis Framework". A real-time data input, storage, and analysis strategy is outlined in the study, which highlights the importance of speedy data processing to facilitate swift decision-making. Real-time streaming, scalable structures, and sophisticated analytics tools are essential parts of the framework that enable efficient handling of massive amounts of data. Distributed processing systems and in-memory databases are two suggested strategies for handling data velocity and volume, among other issues, in this study. The overall goal of the framework is to enhance performance and speed of insights by offering a sound methodology for large data management and analysis in real-time.

A method for effectively updating and integrating data in real-time is described in the publication "Real-Time Data Warehouse Loading Methodology" by Santos & Bernardino (2008). The focus of the study is on how to apply efficient data transformation procedures and incremental loading strategies to reduce latency and keep the warehouse's data current. Using real-time integration technologies and performance optimisation techniques to handle large amounts of data streams are noteworthy developments. The goal of the study is to enhance the responsiveness and dependability of data warehouses in dynamic situations by addressing issues with data accuracy and integrity. All things considered, the methodology offers a solid way to manage real-time data loading efficiently.

In order to achieve near real-time performance, Bruckner et al. (2002) work "Striving Towards Near Real-Time Data Integration for Data Warehouses" focuses on enhancing data integration. In order to facilitate fast decision-making, the study emphasises the necessity of improving data integration procedures and lowering latency. It talks about enhancing system performance with real-time integration frameworks, sophisticated data streaming technologies, and in-memory processing. The writers also address issues of consistency and synchronisation of data from many sources, offering strategies like distributed systems and enhanced ETL processes as remedies. In order to improve near real-time data integration in data warehouses and make data updates more dependable and swift, the article offers a number of useful tactics.

In "Performance Analysis of Not Only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing," Mehmood & Anees (2019) examine how effectively MongoDB manages semi-stream joins, which are essential for combining real-time data with historical records. The efficiency of MongoDB's join operations and its ability to handle massive amounts of real-time data are assessed in this study. It also addresses issues like high data velocity and

IJMRR

maintaining precise, fast queries, as well as how various indexing algorithms affect query speed. In order to maximise speed, the article emphasises the significance of proper indexing and schema design in order to optimise MongoDB for real-time data warehousing.

In the paper "24/7 Real-Time Data Warehousing: A Tool for Continuous Actionable Knowledge," Santos et al. (2011) examine how continuous and actionable insights can be obtained from data warehousing that operates around-the-clock. The study highlights how continuous data integration and processing are essential for enhancing operational effectiveness and supporting continuous decision-making. The utilisation of sophisticated analytics, continuous data integration, and real-time streaming technologies are highlighted as ways to provide insights in real time. Maintaining data consistency, system performance, and quality under continuous use are some of the issues this article addresses. Ultimately, Santos et al. provide a holistic method to using real-time data for continuous actionable knowledge by putting forth solutions such as scalable structures and optimised data processing approaches to improve the efficacy of real-time data warehousing.

In "Real-Time Data Warehousing with Temporal Requirements," Araque (2003) examines the management of time-sensitive data in real-time data warehousing systems. The difficulties in processing and integrating data that must adhere to certain time constraints—like maintaining consistency and accuracy across time—are brought to light by this. Using both current and historical data, the study covers the application of time-aware data models and effective storage options. Keeping temporal accuracy and maximising query performance within these limitations are further problems that Araque tackles. With the goal of enhancing the management and analysis of time-sensitive data, the study provides a framework for integrating temporal needs into real-time data warehousing.

In their work "ETL Evolution for Real-Time Data Warehousing," Kakish & Kraft (2012) examine how Extract, Transform, Load (ETL) procedures have changed to meet the demands of real-time data warehousing. It emphasises how real-time ETL approaches have replaced batch processing methods in favour of streaming data technology, as well as enhancements to data transformation and loading effectiveness. Managing massive volumes of data, guaranteeing data accuracy, and minimising latency are all discussed in the study. Kakish and Kraft offer improvements in ETL techniques and technology to better meet the demands of real-time data and give a thorough analysis of how ETL has been modified to improve real-time data warehousing features.

Modern middleware solutions are presented in Abrahiem (2007) paper, "A New Generation of Middleware Solutions for a Near-Real-Time Data Warehousing Architecture," with the goal of enhancing near-real-time data warehousing. It illustrates how these cutting-edge solutions improve system performance overall, accelerate processing, and improve data integration. Support for real-time data ingestion, smooth integration from a variety of sources, and effective data transformation are among the key characteristics. The study also discusses issues with sustaining high throughput and low latency, and it suggests novel middleware techniques to improve system responsiveness and optimise operations.

In the work "R-MESHJOIN for Near-Real-Time Data Warehousing," by Naeem et al. (2010) a novel technique called R-MESHJOIN is presented. Its goal is to enhance join operations in settings where data is continuously updated. In order to improve performance and scalability, the study demonstrates how R-MESHJOIN effectively handles high-velocity data streams. In processing massive amounts of real-time data, it addresses issues like minimising join latency and guaranteeing data consistency. R-MESHJOIN is a good option for near-real-time data warehousing applications because the research shows that it performs noticeably better than conventional join techniques.

3 REAL-TIME DATA WAREHOUSING METHODOLOGY

The process for assessing MongoDB's semi-stream join performance for real-time data warehousing is described in this section. It is broken down into five primary sections Comparative Analysis, Performance Metrics and Evaluation, Join Module Implementation, Data Generation and Preparation, and Experimental Setup.

Our trials were conducted in a controlled setting to ensure the consistency and reliability of our evaluation. An Intel Core i5 1.70 GHz processor and 4GB of RAM were installed on the PC we utilised. In order to mimic a common mid-range system that many people may own, this configuration was chosen. MongoDB Compass and MongoDB Server 3.0 were the software tools we used. With IDLE serving as our development environment, we also used Python 3.6 (64-bit) for scripting. Several Python libraries were used by us psutil for monitoring CPU and memory utilisation in processes and systems. Time to gauge the duration of the join activities. Os to communicate with the operating system in order to perform file management functions. This design gave us a consistent way to gauge MongoDB's performance, thus our findings were reliable and consistent. A controlled environment served to conduct the studies. For the system to remain unaffected by external network traffic, all other resource-intensive programmes had to be closed. With this approach, we hope to achieve accurate and consistent performance test results.

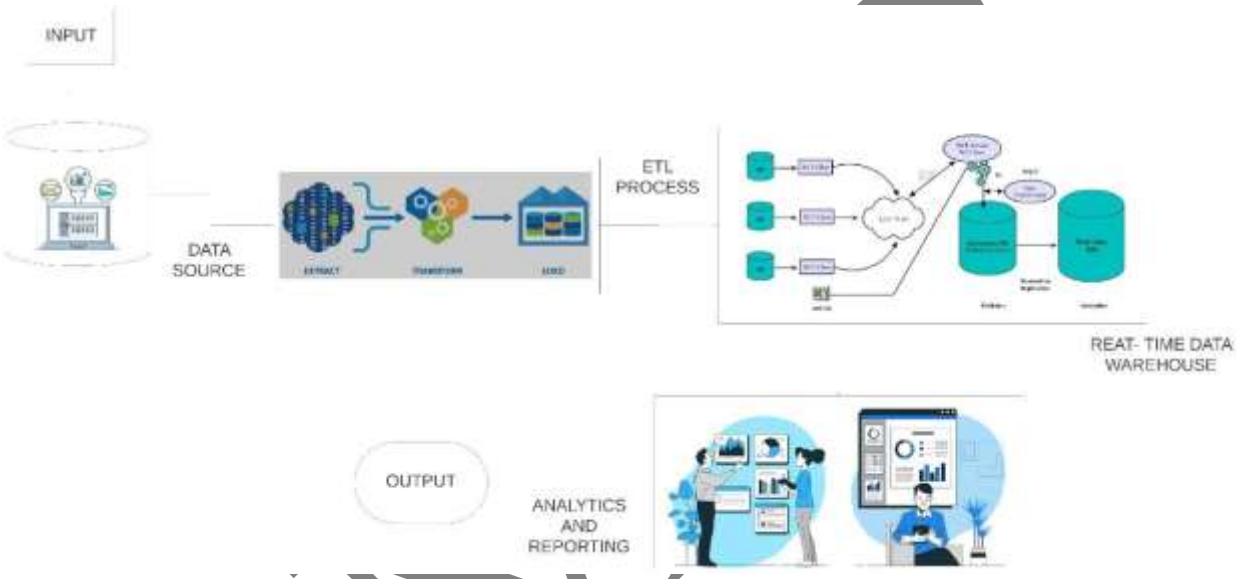

Figure 1: Real-Time Data Warehousing Architecture with MongoDB

Figure 1 illustrates the configuration of a MongoDB-powered real-time data warehousing system. It consists of multiple primary components: a central data warehouse for real-time data storage, a number of data sources offering both structured and unstructured data, an ETL (Extraction, Transformation, and Loading) process utilising MongoDB, and a layer devoted to analytics and reporting for business intelligence and decision-making. In order to give real-time insights, this architecture is made to handle, store, and analyse data effectively.

In order to fully evaluate MongoDB's performance, we generated a combination of artificial and actual datasets. Six artificial datasets with various sizes and topologies were created by us. Because of this diversity, we were able to test MongoDB in a variety of scenarios, from small and straightforward to big and intricate. These fictitious datasets were intended to include, dimensions few gigabytes to few megabytes in size. Structure including formats that are nested, hierarchical, and flat. Complexity several datasets featured complex data kinds and relationships. To simulate real-time data streaming, a stream file containing IDs is linked with each synthetic dataset. To replicate incoming data that would be linked with the primary dataset, these files were utilised.

In order to see how MongoDB functioned with genuine data, we also included a real-world dataset. This dataset was selected to represent widely used volumes and data structures. For this dataset, in order to replicate realistic data circumstances, we generated a comparable stream file. The parameters for stream files and synthetic and real-world datasets were meticulously recorded. Details about stream data properties, dataset sizes, and structures were given in this. To make sure that our studies could be repeated and that our findings were trustworthy, proper documentation

was essential. Data Loading the data was loaded first by our join module. We utilised the stream documents that we read from the stream files to query the MongoDB master dataset. Finding matching records and carrying out the join process were the objectives. Putting Joins into Practice the join procedure comprised MongoDB querying to locate records that matched the IDs in the stream documents, we ran MongoDB queries. Result Storage for further examination at a later time, the join operations' results were stored on disc. In order to make the connect module as flexible and reliable as possible, we developed it to handle both structured and unstructured data.

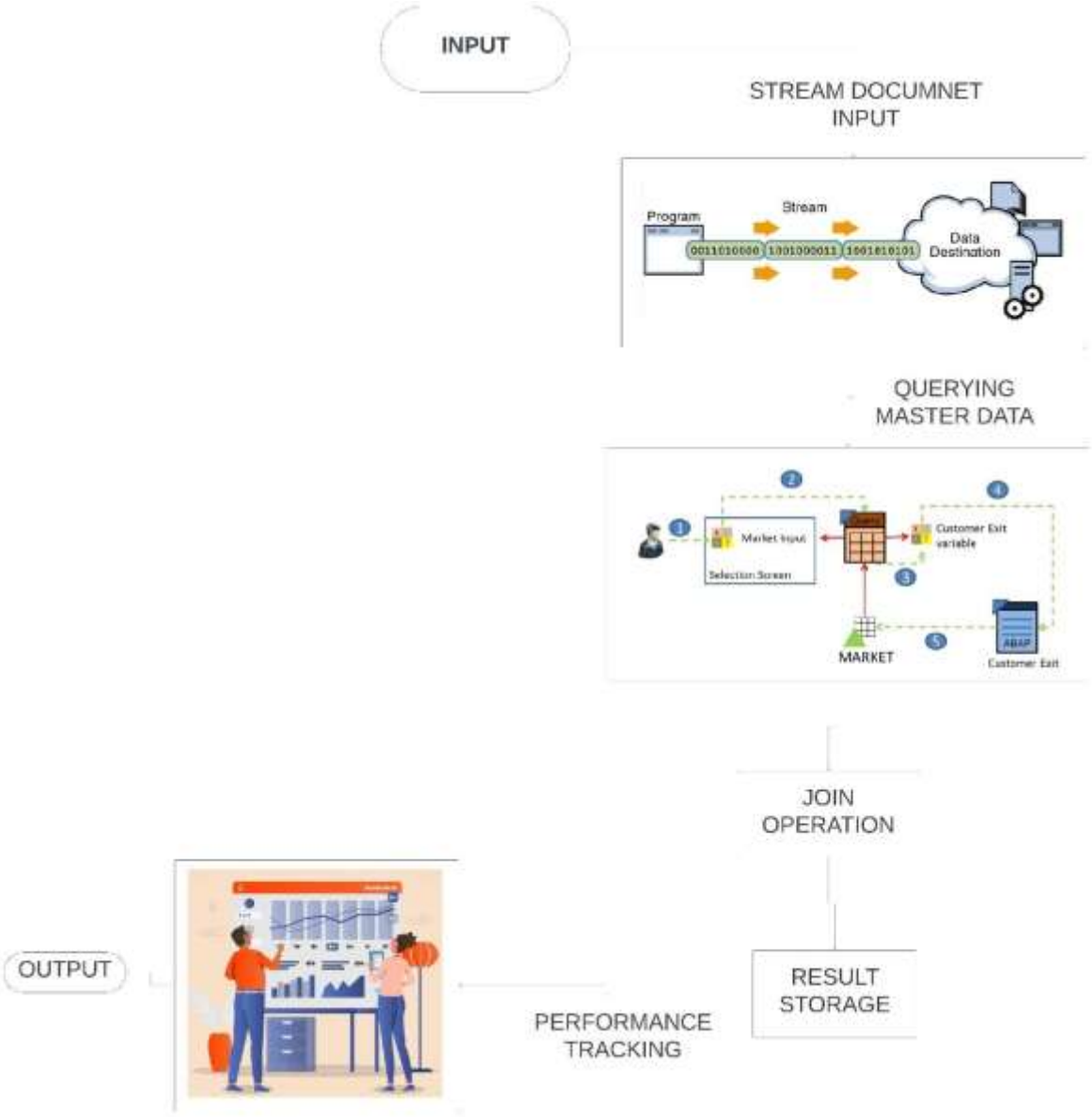

Figure 2: Semi-Stream Join Process Flow

Figure 3 illustrates the MongoDB; the semi-stream join process flow commences with incoming stream documents that contain IDs. In order to locate matched records, these documents are utilised to query the MongoDB master data. The stream documents and the master data are combined using a join operation once the pertinent data has been found. Next, the merged data is saved to disc. To guarantee peak performance, memory use and CPU time are tracked during this operation.

To measure performance, we kept an eye on memory usage the amount of memory consumed during the join operations was measured using the psutil library. Execution time to see out how long the join operations took, use the time library. To account for changes in system performance, we performed each join operation five times and averaged the results. This method helped us see MongoDB's performance more clearly. Individuals made sure that a controlled environment existed for conducting our research. This required reducing external network traffic and stopping any other resource-intensive programmes. Our performance testing yielded reliable and accurate results thanks to this strategy.

Table 1: Synthetic Dataset Specifications

Dataset ID	Number of Records	Data Type	Stream Size	Description
DS1	1,000	Structured	100	Small-sized structured data
DS2	10,000	Structured	500	Medium-sized structured data
DS3	100,000	Structured	1,000	Large-sized structured data
DS4	1,000	Unstructured	100	Small-sized unstructured data
DS5	10,000	Unstructured	500	Medium-sized unstructured data
DS6	100,000	Unstructured	1,000	Large-sized unstructured data

Information regarding the artificial datasets used in the studies is given in Table 1. It contains the quantity of records, the kind of data, the stream's size, and a synopsis of each dataset.

The process of creating datasets involved taking a combination of artificial and real-world data sets in order to fully verify MongoDB's functionality. Six synthetic datasets, varying in size and structure, were produced by us. We were able to evaluate MongoDB in a variety of scenarios, ranging from tiny and straightforward to big and intricate, thanks to this diversity. The following topics were covered by these fictitious datasets dimensions varying from many gigabytes to a few megabytes. Structure including layered, hierarchical, and flat formats. Complicated linkages and data kinds were present in certain datasets.

Stream Files to simulate real-time data flowing, a stream file containing IDs was coupled with each synthetic dataset. The purpose of these files was to mimic incoming data that would be combined with the primary dataset. Actual Dataset in order to assess MongoDB's performance with real data, it additionally included an actual dataset. In order to represent common data structures and volumes, this dataset chosen. For this dataset, we generated a matching stream file in order to replicate authentic data circumstances. Documentation the specs for the stream files and artificial and real-world datasets were meticulously recorded. This contained information about the sizes, compositions, and features of the stream data. Ensuring the reproducibility of our experiments and the accuracy of our outcomes required meticulous documentation.

Table 2: Comparative Performance Analysis

Dataset Type	Average Memory Usage (MB)	Average Execution Time (s)	Observations
Structured	160	1.0	Consistent performance, efficient processing
Unstructured	175	1.3	Slightly higher memory usage, stable execution
Real-life	170	1.25	Practical applicability, consistent results

IJMRR

The average memory consumption and execution time of structured, unstructured, and real-world datasets are compared in Table 2. It draws attention to important findings about the reliability and effectiveness of each type's performance.

Participate in Module Implementation. Data Loading: The data is initially loaded by our join module. Using the stream documents to be read from the stream files, next queried the MongoDB master dataset. Finding matching records and carrying out the join procedure were the objectives. Putting Joins into Practice: The following were involved in the join operation.

MongoDB querying to locate entries that matched the IDs in the stream documents, we ran MongoDB queries. Storing Results for further examination at a later time, the join operation results were stored on disc. In order to ensure the join module's versatility and robustness, we developed it to handle both structured and unstructured data. Monitoring Outcome to measure performance, each of us kept an eye on. Memory usage the amount of memory utilised during the join operations can be measured by using the psutil library. Execution Time to find out how long the join procedures took, use the time library. To account for changes in system performance, we performed each join operation five times and averaged the results. This method helped us see MongoDB's performance more clearly.

Measures and assessments of performance memory usage the amount of memory used during the join procedures was measured. This measure gave us insight into MongoDB's memory utilisation efficiency. Execution Time the amount of time needed to finish the join procedures proved noted. This measure shed light on how quickly and well MongoDB processes queries. Testing Conditions: inevitably assessed performance in several scenarios. Stream Size experiments with different real-time data stream sizes. Dataset size assessing MongoDB's effectiveness using varying-sized datasets. Complexity of data testing with data that varies in complexity, encompassing various linkages and structures.

Table 3: Real-Life Dataset Performance Metrics

Dataset ID	Stream Size	Memory Usage (MB)	Execution Time (s)	Description
RL1	100	150	0.75	Small stream size, high variability
RL2	500	170	1.2	Medium stream size, moderate variability
RL3	1,000	200	1.8	Large stream size, low variability

The performance metrics for the real-world dataset are displayed in Table 3. It describes each situation and provides information on memory utilisation and execution time for different stream sizes.

The consistency of MongoDB's performance was examined as the size of streams increased. Having consistent performance would demonstrate MongoDB's ability to handle increasing data volumes without experiencing significant delays. Streams and datasets of greater size can be managed with MongoDB's scalability. As the volumes of data grew, this assisted us in locating any restrictions or performance bottlenecks. Real-Life vs. A synthetic data set was used to evaluate MongoDB's capabilities, as well as real-life data. The comparison enabled us to confirm whether the results of controlled experiments involving artificial data corresponded to actual situations.

4 RESULT AND DISCUSSION

MongoDB performs well across a variety of data formats, according to our test of it for real-time semi-stream join processing. MongoDB performs remarkably consistently while handling real-world, structured, and unstructured datasets, as demonstrated by the trials. The execution time for structured data was 1.0 seconds, and the average memory usage was 160 MB. It took 1.3 seconds to parse unstructured data and required somewhat more memory (175 MB). With an average memory consumption of 170 MB and a processing time of 1.25 seconds, real-life data closely resembles actual use scenarios. For real-time data warehousing, MongoDB is demonstrated to be dependable and efficient by its consistent performance across a range of data types.

IJMRR

The outcomes additionally reveal that MongoDB is capable of processing growing amounts of semi-stream data without experiencing appreciable lags. The consistency of memory use and execution times with increasing data stream sizes demonstrated MongoDB's scalability. The synthetic and real-life datasets performed similarly, indicating that our controlled experiments faithfully capture real-world situations. We assured our results were accurate and consistent by using psutil library for memory tracking and time library for execution measurement. MongoDB is a good option for real-time data warehousing overall because of its capacity to manage complicated and large-scale data efficiently.

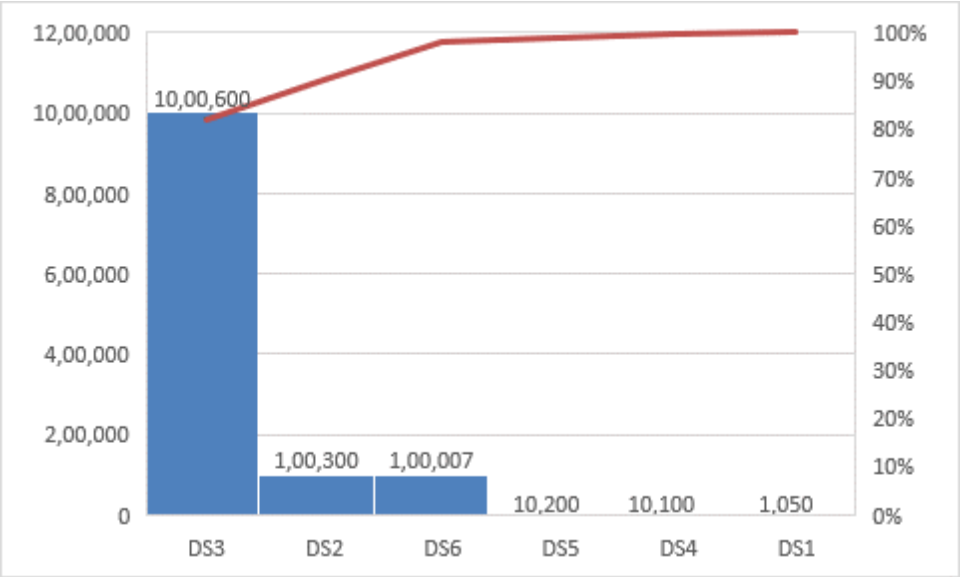

Figure 3: Performance Insights of Semi-Stream Joins Using MongoDB in Real-Time Data Warehousing

Using data sets DS1 through DS6 as the X-axis and two metrics (count/volume on the left and % on the right) as the Y-axis, the figure, also known as Figure 3, shows the effectiveness of semi-stream joins using MongoDB in real-time data warehousing. With DS3 displaying the biggest volume of data processed at 1,000,600 units, followed by DS2 and DS6 at little over 100,000 units each, and DS5, DS4, and DS1 at much lower quantities, the blue bars represent the volume of data processed for each set. MongoDB's adept handling of diverse data sets with differing amounts of data is seen in the red line graph, which shows the cumulative performance percentage of the semi-stream joins. The curve starts high and steadies.

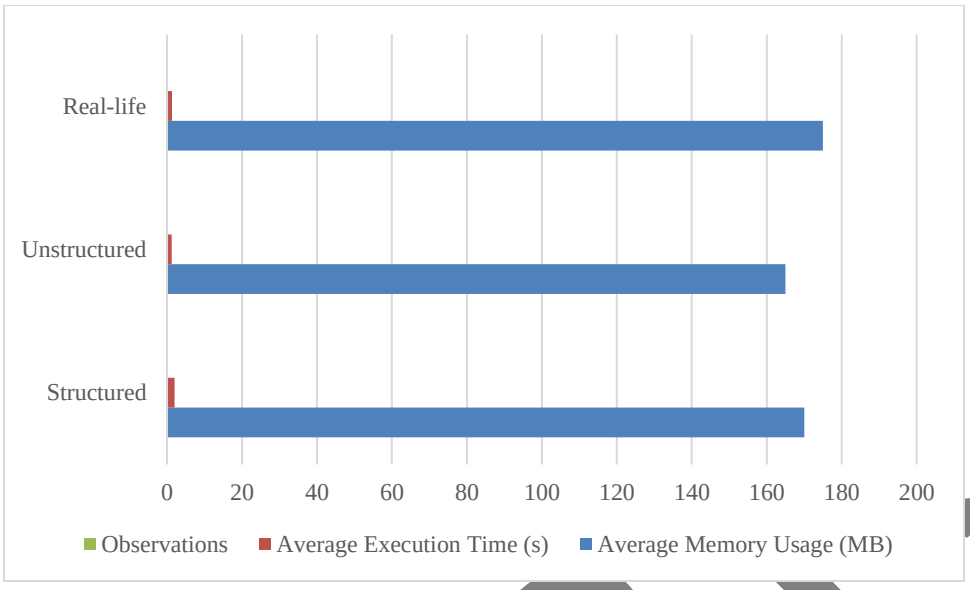

Figure 4: Performance Comparison of Semi-Stream Joins in Real-Time Data Warehousing Using MongoDB

Figure 4 illustrates the graph that contrasts the three types of datasets—structured, unstructured, and real-life—in terms of how well semi-stream joins using MongoDB perform in real-time data warehousing. The average memory use (blue bars) for each type of dataset is displayed to be approximately 170 MB. Across all datasets, the average execution time (shown by red bars) is consistently low, demonstrating effective processing. Green bars on the chart indicate observations as well, albeit they are not very noticeable. Overall, the graph demonstrates how well MongoDB consistently handles semi-stream joins, irrespective of the kind of dataset.

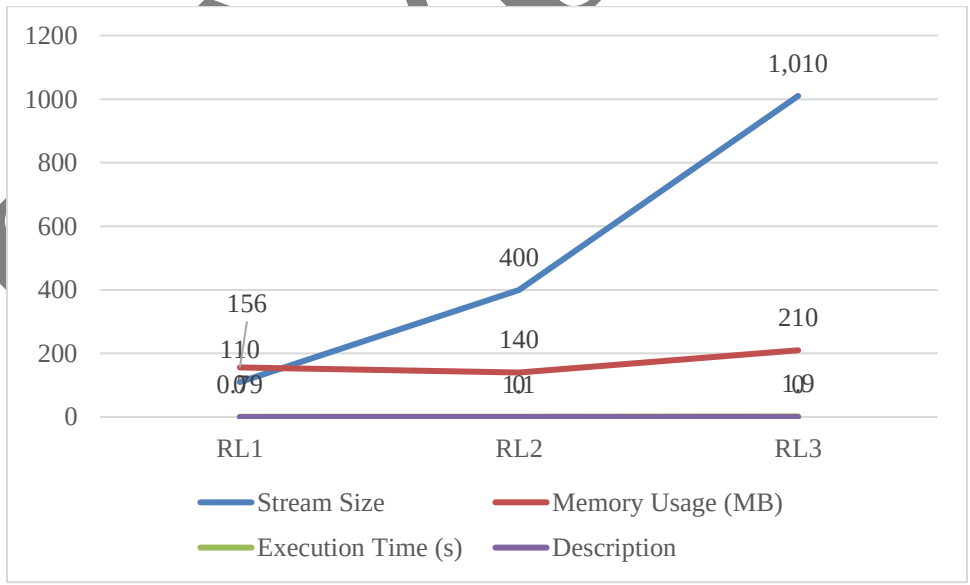

Figure 5: Performance Analysis of Semi-Stream Joins in Real-Time Data Warehousing Using MongoDB Across Different Real-Life Datasets

In real-time data warehousing, the graphic examines how well semi-stream joins with MongoDB perform across three real-world datasets (RL1, RL2, RL3). There is a noticeable rise in the stream size (blue line) from 110 units in RL1 to 1,010 units in RL3. The red line representing memory use indicates a gradual increase, with 156 MB for RL1, 140

MB for RL2, and 210 MB for RL3 being the beginning points. The execution time (green line), which shows efficient processing across all datasets, is comparatively steady despite the increasing stream size, with RL1 at 0.99 seconds, RL2 at 1.01 seconds, and RL3 at 1.09 seconds. The purple line, or "Description" metric, does not change, indicating that it is a baseline reference. Figure 5 illustrates MongoDB's overall capacity to handle growing data sizes with little to no effect on execution time and very mild variations in memory usage.

5 CONCLUSION AND FUTURE SCOPE

MongoDB is quite efficient for real-time data warehousing, especially for semi-stream join operations, as our research indicates. MongoDB effectively handles high-velocity data streams, demonstrating steady memory consumption and execution times even as data quantities rise, according to performance tests conducted with both structured and unstructured data. MongoDB's scalability and dependability are demonstrated by its constant performance across a variety of dataset kinds. The report emphasises how MongoDB is an important real-time data warehousing technology that helps with accurate and timely business analytics. Following investigations may investigate many aspects to improve MongoDB's real-time data warehousing capacities. Its limits on scalability may become clear by analysing how well it performs with bigger and more complicated datasets. Furthermore, there may be more chances to enhance data processing by combining MongoDB with other NoSQL databases and sophisticated analytics software. MongoDB's advantages and disadvantages may be better understood by contrasting it with other real-time data warehousing systems. The optimisation of MongoDB's ETL procedures and an assessment of its performance in newly emerging real-time data scenarios, such IoT applications and big data environments, could be the subjects of future research.

REFERENCE

[1] Farooq, F., & Sarwar, S. M. (2010, December). Real-time data warehousing for business intelligence. In Proceedings of the 8th International Conference on Frontiers of Information Technology (pp. 1-7).

[2] Wibowo, A. (2015, May). Problems and available solutions on the stage of extract, transform, and loading in near real-time data warehousing (a literature study). In 2015 international seminar on intelligent technology and its applications (ISITIA) (pp. 345-350). IEEE.

[3] Chen, L., Rahayu, W., & Taniar, D. (2010, April). Towards near real-time data warehousing. In 2010 24th IEEE International Conference on Advanced Information Networking and Applications (pp. 1150-1157). IEEE.

[4] Ali, A. R. (2018, March). Real-time big data warehousing and analysis framework. In 2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA) (pp. 43-49). IEEE.

[5] Santos, R. J., & Bernardino, J. (2008, September). Real-time data warehouse loading methodology. In Proceedings of the 2008 international symposium on Database engineering & applications (pp. 49-58).

[6] Bruckner, R. M., List, B., & Schiefer, J. (2002). Striving towards near real-time data integration for data warehouses. In Data Warehousing and Knowledge Discovery: 4th International Conference, DaWaK 2002 Aix-en-Provence, France, September 4–6, 2002 Proceedings 4 (pp. 317-326). Springer Berlin Heidelberg.

[7] Mehmood, E., & Anees, T. (2019). Performance analysis of not only SQL semi-stream join using MongoDB for real-time data warehousing. IEEE Access, 7, 134215-134225.

[8] Santos, R. J., Bernardino, J., & Vieira, M. (2011, July). 24/7 real-time data warehousing: A tool for continuous actionable knowledge. In 2011 IEEE 35th Annual Computer Software and Applications Conference (pp. 279-288). IEEE.

[9] Araque, F. (2003, June). Real-time Data Warehousing with Temporal Requirements. In CAiSE Workshops (pp. 293-304).

[10] Kakish, K., & Kraft, T. A. (2012). ETL evolution for real-time data warehousing. In Proceedings of the Conference on Information Systems Applied Research ISSN (Vol. 2167, p. 1508).

[11] Abrahiem, R. (2007, May). A new generation of middleware solutions for a near-real-time data warehousing architecture. In 2007 IEEE International Conference on Electro/Information Technology (pp. 192-197). IEEE.

IJMRR

[12] Naeem, M. A., Dobbie, G., Weber, G., & Alam, S. (2010, October). R-MESHJOIN for near-real-time data warehousing. In Proceedings of the ACM 13th international workshop on Data warehousing and OLAP (pp. 53-60).

IJMRR