Kurdistan Journal of Applied Research (KJAR)
Print-ISSN: 2411-7684 | Electronic-ISSN: 2411-7706

Website: Kjar.spu.edu.iq | Email: kjar@spu.edu.iq

Improving The Performance of Big Data Databases

Nzar Abdulqadir Ali
Computer Science Department
Cihan University Sulaimani
Sulaimani, Iraq
Nzar@mail.com

Dashne Raouf Arif
Sulaimani Technical Institute
Sulaimani Polytechnic University
Sulaimani, Iraq
Dashne.raouf@spu.edu.iq

Volume 4 – Issue 2
December 2019

DOI:
10.24017/science.2019.2.20

Received:
04 November 2019

Accepted:
22 December 2019

Abstract
Real-time monitoring systems utilize two types of database, they are: relational databases such as MySQL and non-relational databases such as MongoDB. A relational database management system (RDBMS) stores data in a structured format using rows and columns. It is relational because the values of the tables are connected. A non-relational database is a database that does not adopt the relational structure given by traditional. In recent years, this class of databases has also been referred to as Not only SQL (NoSQL). This paper, discusses many comparisons that have been conducted on the execution time performance of types of database (SQL and NoSQL). In SQL (Structured Query Language) databases different algorithms are used for inserting and updating data, such as indexing, bulk insert and multiple updating. However, in NoSQL different algorithms are used for inserting and updating operations such as default-indexing, batch insert, multiple updating and pipeline aggregation. As a result, firstly compared with related papers, this paper shows that the performance of both SQL and NoSQL can improved. Secondly, performance can be dramatically improved for inserting and updating operations in the NoSQL database compared to the SQL database. To demonstrate the performance of the different algorithms for entering and updating data in SQL and NoSQL, this paper focuses on a different number of data sets and different performance results. The SQL part of the paper is conducted on 50,000 records to 3,000,000 records, while the NoSQL part of the paper is conducted on 50,000 to 16,000,000 documents (2GB) for NoSQL. In SQL, three million records are inserted within 606.53 seconds, while in NoSQL this number of documents is inserted within 67.87 seconds. For updating data, in SQL 300,000 records are updated within 271.17 seconds, while for NoSQL this number of documents is updated within just 46.02 seconds.

Keywords: Big data, Real-time, SQL, and NoSQL

1. INTRODUCTION

Big data is a term used to explain sets of data that are of several forms or structures, achieve extremely high speeds and cannot be processed successfully by traditional database management systems [1] [2]. Zhou et al [3], argued that by the end of 2015 the overall data volume is going to surpass 7.9 Zettabytes(ZB) reaching 35ZB by 2020. Big data has five features [4] [5] that are known as the "5Vs": volume, velocity, variety, veracity and value as shown in Figure (1) [6].

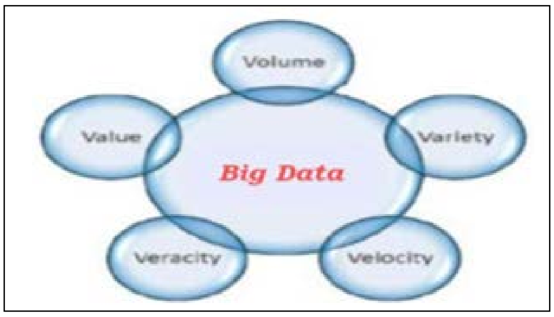


Figure 1: Big data 5 V [6].

1.Volume: Indicates large amounts of data used for various functions, for example, data for mobile devices.
2.Velocity: Indicates the speed or frequency at which data is created, updated, processed, and accessed.
3.Variety: Data collection through different device types such as videos, images, etc.
4.Value: Indicates the way in which massive data sets are used to draw useful knowledge. Value is the most important feature of any large data tool as it allows useful information to be generated
5.Veracity: Relates to the high accuracy of knowledge or informatics and value.
Two types of databases are used to store big data: relational databases such as MySQL and no n-relational databases such as MongoDB are available.
The problem in this paper is a big data management while storage have become a global challenge in recent years. In the past, relational databases were used to store and manage data; increasing the amount of data generated a new type of database as NoSQL. On the other hand, the SQL database does not stop exactly for sensitive data such as banking services. For this reason, some researchers measured the test performance of each type of database and have compared response time measurements.
The aim of this paper is to improve performance in the real-time monitoring system regarding inserting and updating big data using different types of algorithms in both SQL and NoSQL databases.

1.1. Relational Model (SQL)

The traditional database is the most well-known type of database and by far the most commonly used. It was originally developed for the long-term storage of information [6]. The entire data load is contained in tables of rows and columns in a relational database. Tables can be seen as unified relational data bodies. Rows have unique data sets and are grouped by column [7]. A unique index is required for each row or column in a table; through matching data fields various tables are linked together. One of the most popular databases used for SQL is MySQL[8] [9].

1.2. Non- relational databases (NoSQL)

Generally, the term "NoSQL" refers to database systems that do not comply with a strict relational data model. Higher availability and scalability can be achieved using NoSQL databases, which are important criteria for big data processing. NoSQL databases do not need to be finalized at an early stage of database design with a fixed scheme of predefined data structures and limitations [10] [11]. A non-relational database is a database that does not contain the rows and columns used in most conventional database systems. Instead, non-relational databases use a storage model that is optimized for the specific requirements of the type of data being stored. For example, data may be stored as simple key/value pairs, as JSON documents, or as a graph consisting of edges and vertices [12] [13].

There are four different categories of NoSQL that are classified by storage of information [14] [15].

1.Key-value: Each item in the database is stored as a name (key) attribute along with a name (value) attribute. The most common for this type are Riak and Voldemort.

2.Wide-column stores: Store data together as columns instead of rows. Cassandra and HBase are the most common.

3.Document A document database is a kind of non-relational database designed to store and search for information in JSON-like files. Document databases make it easier for developers to store and request information in a database that is used in their application software. MongoDB is the top database of this type.

4. Graph databases: Used to store network data such as social connections such as Neo4J.

One of the top databases used for NoSQL is MongoDB [16], MongoDB is a NoSQL system and is a document-based database. The data is saved in BSON form, which is a binary sequence encoding JSON documents. In MongoDB, a collection is comparable to a table and a document is equal to a record in a relational record. MongoDB currently provides official driver support for all popular programming languages like C, C++, C#, Java, Node, Perl, PHP, Python [18] [17].

MongoDB uses some kinds of aggregation; for instance, the aggregation pipeline is a data aggregation system based on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into aggregated results [19] [20].

2. LITERATURE WORKS

H. Ansari 2018 [9]. This paper conducted a comparison between MySQL and MongoDB on two factors, performance and space allocation, with different data sizes. The results showed that MongoDB was faster than MySQL at for every process in almost every test case, but MySQL allocated less storage when it held large volumes of data. For instance, when inserting 10^6 records in each database, the size of MongoDB was 270.42 Megabytes (MB) while the size of MySQL was 175.28MB.

D. Merriman 2018 [24]. The paper was focused on pipeline aggregation. This framework can be designed to optimize aggregate operations that include; data access, data retrieval, data writes, indexing, aggregate multiple operations and/or commands. This aggregation operation can be defined as a pipeline that provides the results of the first process to be forwarded to the next process input. Computations can also be performed at each stage of the process, where the calculation results of each stage aggregate until the final result is reached.

E. Andersson and Z. Berggren 2017 [8]. Illustrated comparison between MySQL and MongoDB concerning different operations such as single and multi-insert. The paper explained that MongoDB was faster in every operation; 10^6 records were inserted in MySQL within 1020 seconds while the equivalent value for MongoDB was 105 seconds concerning a single operation. For multi-operation, the same number of records were inserted in MySQL within 50 seconds and were inserted in MongoDB within 28 seconds. Update query was another comparison conducted in their paper using different data sizes; for example, updating one element for 10^6 records took 2.5 seconds in MySQL and about 0.5 seconds in MongoDB.

D. Sink 2017 [21]. This paper compared sequence and random inserting and updating for different popular types of databases such as MySQL, MongoDB and Rethink DB. The results showed that MongoDB was faster than each type of NoSQL for sequence or random insert but for an update operation, if random update was used then MySQL was faster than MongoDB and MongoDB were faster than Rethink DB.

V. Abramova et al 2014 [22]. Their paper compared between the distinguished popular types of NoSQL: Cassandra, HBase, MongoDB, Orient DB and Redis. Reading and updating on specific mechanisms and applications. The results of the comparison showed that Cassandra and HBase were faster to update than MongoDB but slower to read. In every operation, the Orient DB was the lowest performance and slower than MongoDB.

Z. Parker. et al 2013 [23]. This paper also conducted a comparison between SQL and NoSQL and indicated that NoSQL was faster for inserts and updates in a simple query; nevertheless, SQL had more speed when updating with complex querying or non-key attributes.

3. METHODS AND MATERIALS

The main focus of this paper is on testing performance on big data in real-time monitoring system. This reminder of this paper develops in two main sections. The first section to SQL attempts to increase performance by using multi-insert (bulk insert) instead of single insert. When a bulk insert is inserted, multiple records in one operation are separated by commas. It also tries to accelerate performance by finding and updating data using a multi-update operation and indexing. The second section is related to NoSQL, the aim is to increase performance using two stages: The first stage inserts data by using batch insert to insert multi documents at a time; by default, MongoDB batch inserts 101 documents. The second stage finds and updates documents using multi-update operation and pipeline aggregation.

3.1. The proposed database architecture

In this section, a design is formulated consisting all the steps needed to create the best database system for big data in a monitoring system, shown in Figure (2).

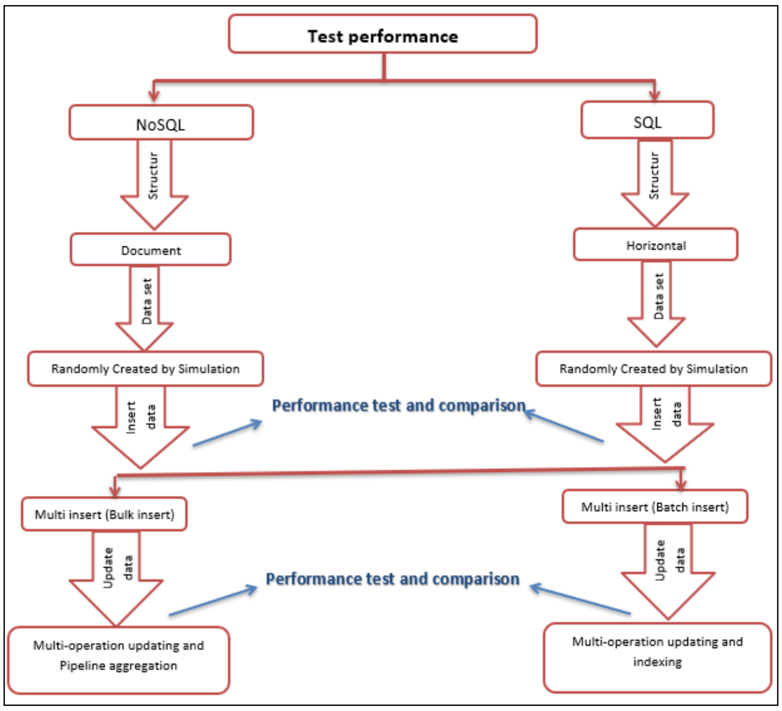

Test performance
NoSQL
SQL
Structur
Structur
Document
Horizontal
Data set
Data set
Randomly Created by Simulation
Randomly Created by Simulation
Insert data
Insert data
Performance test and comparison
Multi insert (Bulk insert)
Multi insert (Batch insert)
Update data
Update data
Performance test and comparison
Multi-operation updating and Pipeline aggregation
Multi-operation updating and indexing

Figure 2: System design of monitoring system in SQL and NoSQL.

3.2. Structured Query Language (SQL)

Structured Query Language is the typical way in which information is stored in a table. This section discusses how to enhance the performance of the database to create, add and update information.

3.2.1. Database schema

To implement our algorithms for SQL we created a database for a real-time temperature monitoring system, as shown in Figure (3). The first table is called Initial-table and the second table is called Join-table.

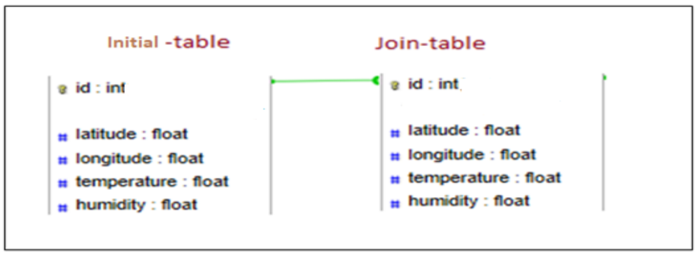
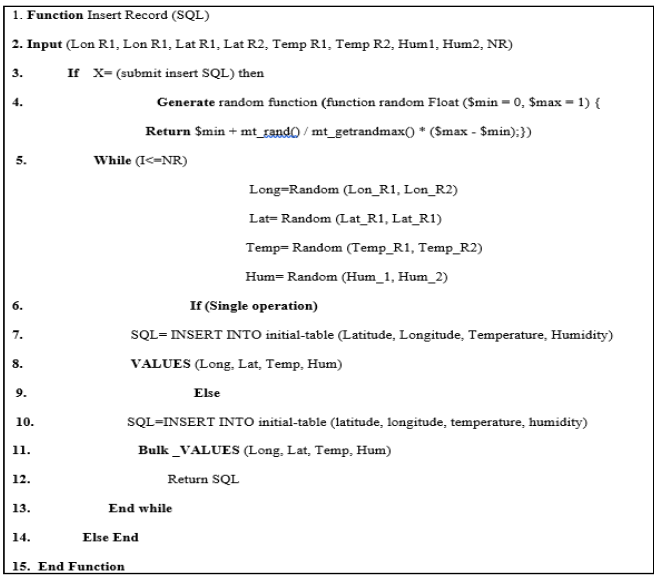


Figure 3: Database table in this system.

To create and update the dataset in SQL, algorithms have to be created; from those algorithms the following definition are made: “NR as number of records”, “Lon_R1 as minimum-value longitude”, “Lon_R2 as maximum-value longitude”, “Lat_R1 as minimum-value latitude” , “Lat_R2 as maximum-value latitude”, “Temp_R1 as minimum-value temperature”, “Temp_R2 maximum-value temperature”, “Hum_1 as minimum-value humidity”, and “Hum_2 as maximum- value humidity”.

3.2.2 Inserting data

The “insert into” statement adds a new record to a table using the simulation method. Data simulation means generating a random function between the range values. The value range for all fields is by default between 0 and 1, but the random function can return an unexpected value within the specified range. The value of each field is entered in to the table through bulk insert or multi-run because a single data insertion process takes more time and reduces performance. Figure (4) illustrates the algorithm for entering data into SQL using the simulation method.



```
1. Function Insert Record (SQL)
2. Input (Lon R1, Lon R1, Lat R1, Lat R2, Temp R1, Temp R2, Hum1, Hum2, NR)
3.      If  X= (submit insert SQL) then
4.                  Generate random function (function random Float ($min = 0, $max = 1) {
                  Return $min + mt_rand() / mt_getrandmax() * ($max - $min);})
5.         While (I<=NR)
                              Long=Random (Lon_R1, Lon_R2)
                              Lat= Random (Lat_R1, Lat_R1)
                              Temp= Random (Temp_R1, Temp_R2)
                              Hum= Random (Hum_1, Hum_2)
6.                     If (Single operation)
7.               SQL= INSERT INTO initial-table (Latitude, Longitude, Temperature, Humidity)
8.               VALUES (Long, Lat, Temp, Hum)
9.                     Else
10.              SQL=INSERT INTO initial-table (latitude, longitude, temperature, humidity)
11.               Bulk _VALUES (Long, Lat, Temp, Hum)
12.                  Return SQL
13.          End while
14.       Else End
15. End Function
```

Figure 4: Algorithm to multi-insert data in SQL using the simulation algorithm.

3.2.3 SQL updating data

This section shows how performance was tested for changing the temperature to equal or greater than 40 degrees in SQL using an updating query. Figure (5) illustrates the algorithm for randomly updating data into SQL. The database for SQL includes Initial-table and Join-table; the initial-table aims to store all the data, but the second table was created to store only updating data and to insert data from the initial-table to second table randomly. For example, if we want to update 10^3 records from 10^6 in the first table, we select 10^3records in the initial-table (first table) in random order, inserting them in to the second table (join-table) and creating a join between them depending on their ID field. The goal of the second table is increase performance. In this case, less data scanned for updating.

```
1. Function Update Record (SQL)
2. Input (NR)
3.      If   X= (Submit Update SQL) then
4.          While (I<=NR)
5.                  Generate random update
6.              While (ID NOT IN (SELECT ID FROM Join-table))
7.                      Insert values from initial table to the join table
8.                          Order by Rand () Limit 1, NR
9.                              If (Primary Index)
10.                                 Generate Cluster Index
11.                             Else
12.                                 Generate Secondary Index
13.                 End While
14.                             While (Second table. Temperature<40)
15.                         Update Second table. Temperature equal or greater than 40
16.                             End While
17.         End while
18.  End Function
```

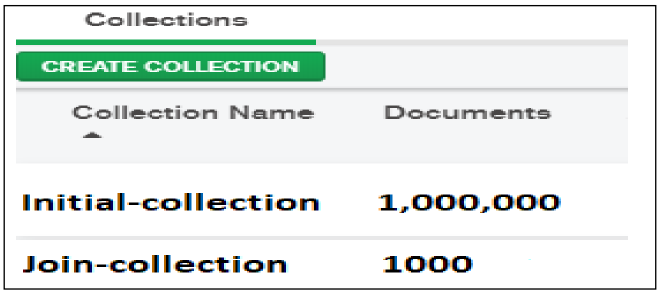
Figure 5: Algorithm to update data in SQL using the random function.

3.3. Not only SQL Language (NoSQL)

NoSQL is a new generation of database management systems that differ primarily from relational database management systems. These databases do not require column tables, avoid joins and usually support horizontal scaling. MongoDB is used as a higher NoSQL database in this paper. This section explains how to improve the performance of the database

3.3.1. Database collections

A collection in NoSQL is like a table in SQL; this section describes two collections. The name of the first collection is initial-collection and the second is called join-collection, as shown in Figure (6).



Collection Name	Documents
Initial-collection	1,000,000
Join-collection	1000

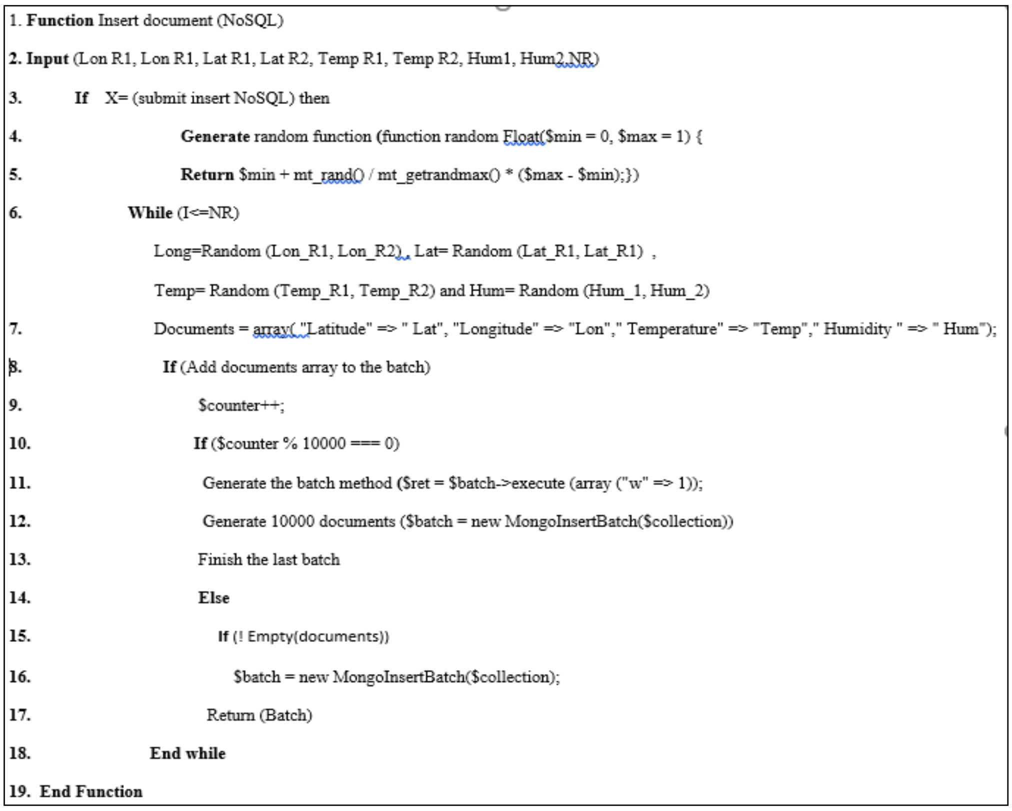
Figure 6: Database collection in this paper.

To create and update the dataset in NoSQL, algorithms have to be created; from those algorithms the following definition are made: “NR as number of documents”, “Lon_R1 as minimum-value longitude”, “Lon_R2 as maximum-value longitude”, “Lat_R1 as minimum-value latitude” , “Lat_R2 as maximum-value latitude”, “Temp_R1 as minimum-value temperature”, “Temp_R2 maximum-value temperature”, “Hum_1 as minimum-value humidity”, and “Hum_2 as maximum- value humidity”.

3.3.2 Inserting data in NoSQL

MongoDB stores documents in collections. Inserting data in to MongoDB takes place via single insert or multi-insert (batch insert) operations. In this paper, batch insert is used to restrict data insertion.

Batch input is used to insert multiple documents in one iteration, by default 101 documents. But as shown in this section, this number can be increased to 10,000 documents instead of 101 documents as shown in Figure (7), in this case we can increase the performance of the inserting operation.

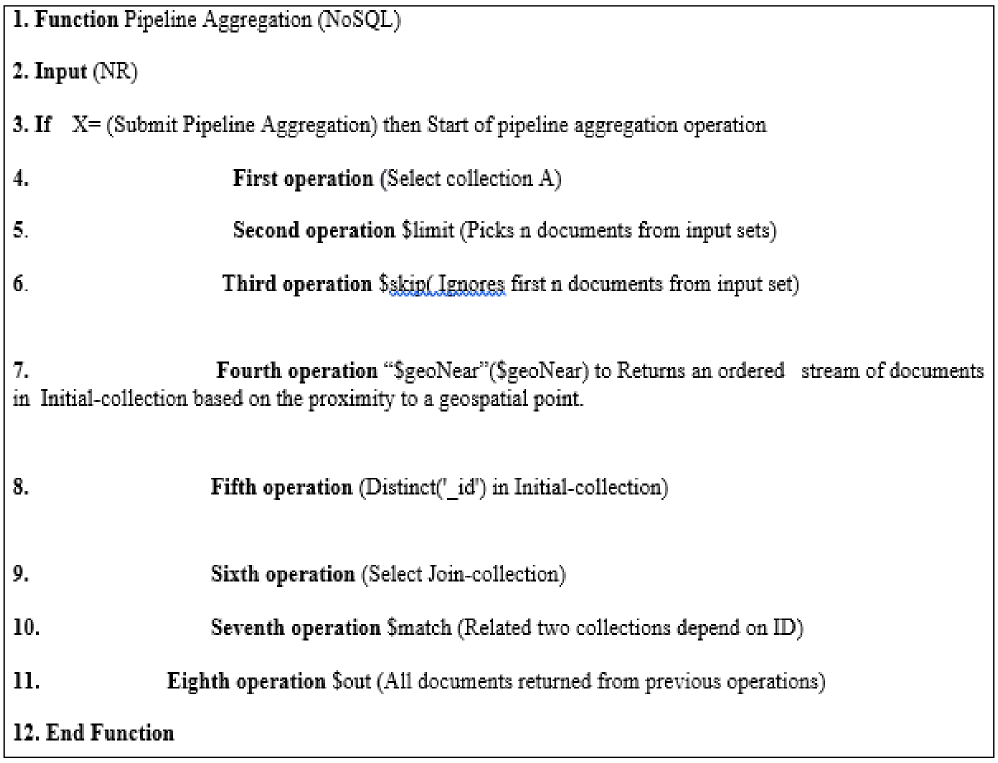


```
1. Function Insert document (NoSQL)
2. Input (Lon R1, Lon R1, Lat R1, Lat R2, Temp R1, Temp R2, Hum1, Hum2,NR)
3.    If  X= (submit insert NoSQL) then
4.            Generate random function (function random Float($min = 0, $max = 1) {
5.            Return $min + mt_rand() / mt_getrandmax() * ($max - $min);})
6.        While (I<=NR)
              Long=Random (Lon_R1, Lon_R2), Lat= Random (Lat_R1, Lat_R1) ,
              Temp= Random (Temp_R1, Temp_R2) and Hum= Random (Hum_1, Hum_2)
7.            Documents = array( "Latitude" => " Lat", "Longitude" => "Lon"," Temperature" => "Temp"," Humidity " => " Hum");
8.             If (Add documents array to the batch)
9.                 $counter++;
10.                If ($counter % 10000 === 0)
11.                 Generate the batch method ($ret = $batch->execute (array ("w" => 1));
12.                 Generate 10000 documents ($batch = new MongoInsertBatch($collection))
13.                Finish the last batch
14.                Else
15.                  If (! Empty(documents))
16.                    $batch = new MongoInsertBatch($collection);
17.                 Return (Batch)
18.           End while
19. End Function
```

Figure 7: Algorithm code for multi-insert operation in NoSQL.

3.3.3. Updating data in NoSQL

Regarding testing the performance of changing the temperature in NoSQL using an updating query, there is a different way to update documents in a collection. When updating and finding documents, MongoDB provides some aggregation operations. In this paper we use pipeline aggregation with a join between collections. Documents are entered as inputs to a multi-stage pipeline that converts documents into a combined result; this section, shows that a pipeline can also be successfully created using a join between two collections without embedding and duplicating the data; the stages of this operation are shown in Figure (8).



```
1. Function Pipeline Aggregation (NoSQL)
2. Input (NR)
3. If    X= (Submit Pipeline Aggregation) then Start of pipeline aggregation operation
4.                First operation (Select collection A)
5.                Second operation $limit (Picks n documents from input sets)
6.                Third operation $skip( Ignores first n documents from input set)
7.                Fourth operation "$geoNear"($geoNear) to Returns an ordered   stream of documents
in  Initial-collection based on the proximity to a geospatial point.
8.                Fifth operation (Distinct('_id') in Initial-collection)
9.                Sixth operation (Select Join-collection)
10.               Seventh operation $match (Related two collections depend on ID)
11.               Eighth operation $out (All documents returned from previous operations)
12. End Function
```

Figure 8: Pipeline aggregation stages in NoSQL with a join between two collections.

The database for NoSQL includes Initial-collection and Join- collection; the initial- collection aims to store all the data, but the second collection was created to store only updating data and to insert data from the initial-collection to the second collection randomly. For example, if we want to update 10^3 documents from 10^6 in the first collection, we select 10^3 documents in the initial- collection (first collection) in random order, inserting them in to the second collection (join- collection) and creating a join between them depending on their key field. One of the disadvantages of NoSQL is that it is not possible to join the data of two collections; instead, it is only possible to data. However, in this paper, we solved this problem by creating a join between data for their collections (initial- collection and join-collection). The goal of the second table is to increase performance; in this case less data is scanned for updating.

4. RESULTS AND DISCUSSION

The results of the implementation algorithms described in section 3 will be presented and discussed in this section. This section will test the performance and present a comparison of different SQL and NoSQL operations. Identifying the test environment includes the hardware and software configuration that will be used during the test:

- CPU: Intel Core i5-4300U CPU @1.90GHZ 2.10 GHz on a personal laptop.
- RAM: 8GB
- HDD: 256GB SSD
- OS: Windows 10 Professional 64Bit
- Software: Two difference software programs are used, MY-SQL and MongoDB.
- Test performance: Performance is tested by calculating the response time within the process (start query and end query).

4.1. SQL performance

The results of the performance test are presented in this section with regard to inserting and modifying data in SQL.

4.1.1. Inserting performance

To test the inserting data performance, multiple (bulk) insert for different data sets are used and calculated in seconds. The first dataset starts with 50,000 records and the last one has 3000000 records as shown in Table (1).

Table 1: Performance test for multi (bulk) insert operation in SQL.

NO.	Data set (Records)	Response time to insert (Seconds)
1	50000	4.77
2	100000	8.72
3	150000	17.25
4	250000	25.54
5	500000	59.80
6	1000000	84.86
7	1500000	85.02
8	2000000	390.83
9	2500000	460.38
10	3000000	606.53

4.1.2. Updating performance

To test the updating data performance, index and multiple update are used. The first update 5,000 records from 50,000 records and the last one update 300,000 records from 3,000,000 records as shown in Table (2)

Table 2: Performance test for update query in SQL.

NO.	Data set (Records)	Records to updating	Response time to update (Seconds)
1	50,000	5,000	0.32
2	100,000	10,000	0.7
3	150,000	15,000	1.64
4	250,000	25,000	3.85
5	500,000	50,000	6.53
6	1,000,000	100,000	40.55
7	1,500,000	150,000	169.47
8	2,000,000	200,000	145.49
9	2,500,000	250,000	172.24
10	3,000,000	300,000	271.17

4.2. NoSQL performance

This section presents the test and comparison results for entering and updating NoSQL documents using different algorithms.

4.2.1. Inserting performance

To test the inserting data performance, multiple (batch) insert for different data sets are utilized. The first dataset starts with 50,000 documents and the last one has 16,000,000 documents as shown in Table (3).

Table 3: Test performance to multi insert data in NoSQL

NO.	Data set (Documents)	Response time to insert documents (Seconds)
1	50000	1.61
2	100000	2.34
3	150000	4.10
4	250000	5.83
5	500000	11.46
6	1000000	23.26
7	1500000	34.50
8	2000000	46.02
9	2500000	57.14
10	3000000	67.87
11	16,000,000	357

4.2.2. Updating performance

To test the updating data performance, pipeline aggregation and join between collections are used for different data sets. The first update 5,000 documents from 50,000 documents and the last one update updated 300,000 documents from 3,000,000 documents as shown in Table (4).

Table 4: Updating query by pipeline aggregation operation.

NO.	Data set (Documents)	Documents to updating	Update Documents (Seconds)
1	50000	5,000	0.21
2	100000	10,000	0.42
3	150000	15,000	0.65
4	250000	25,000	1.09
5	500000	50,000	2.32
6	1000000	100,000	4.67
7	1500000	150,000	6.994
8	2000000	200,000	9.55
9	2500000	250,000	11.524
10	3000000	300,000	14.22

4.3. Comparison between SQL and NoSQL

Taking into considering all the results, we can determine the best way to increase performance for entering and updating records (documents) for a large amount of data in both SQL and NoSQL. In this section a comparison is made between SQL and NoSQL for 10 different data sets from 50000 to 16,000,000 records. As shown in Table (5), NoSQL is more suitable for inserting bigdata. For example, inserting $2*10^6$ records in SQL took 390.83 seconds but $16*10^6$ documents (2.2 GB) can be inserted in the same number of seconds in NoSQL.

The other comparison between SQL and NoSQL concerns the updating of data; for instance, in SQL $3*10^5$ records can be updated from $3*10^6$ records within 271.17 seconds, but for NoSQL the equivalent time is only 14.22 seconds.

Table 5: Comparison of test results between SQL and NoSQL.

Data set	Insert records SQL(Sec)	Insert documents NoSQL (Sec)	Records to update	Update records SQL (Sec)	Update documents NoSQL (Sec)
50000	4.77	1.61	5000	0.21	0.21
100000	8.72	2.34	10000	0.7	0.42
150000	17.25	4.10	15000	1.64	0.65
250000	25.54	5.83	25000	3.85	1.09
500000	59.80	11.46	50000	6.53	2.32
1000000	84.86	23.26	100000	40.55	4.67
1500000	85.02	34.50	150000	169.47	6.994
2000000	390.83	46.02	200000	145.49	9.55
2500000	460.38	57.14	250000	172.24	11.524
3000000	606.53	67.87	300000	271.17	14.22
4000,000		73.87	400,000		19.21
10,000,000		242	450,000		23.3
16,000,000		357	500,000		28.3

The results of each comparison show that NoSQL is best used to improve performance when dealing with big data, as shown in Figure (9).

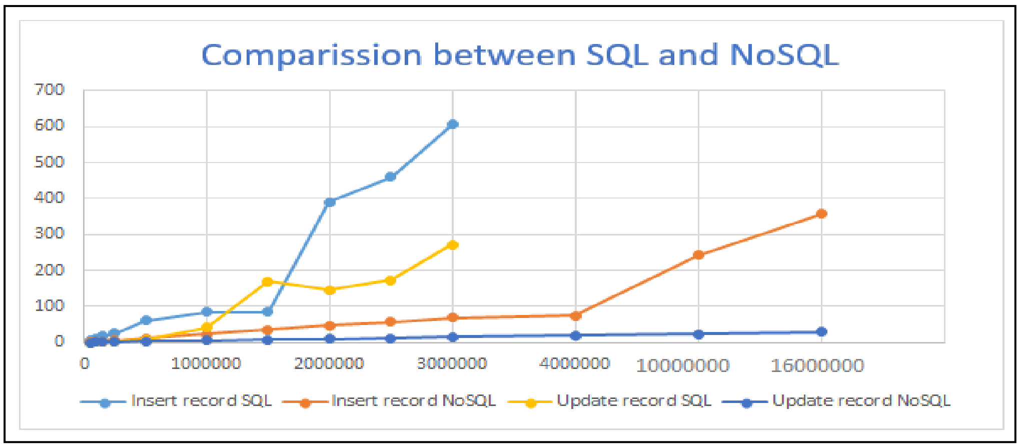
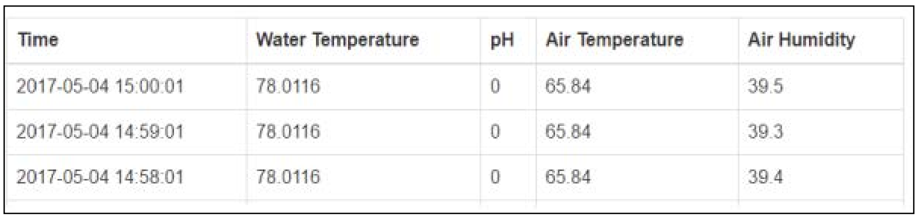


Figure 9: comparison of execution time results between SQL and NoSQL for inserting and updating data.

4.4. Comparison between this paper and previous studies.

Studies have been conducted on the performance of relational (SQL) and non-relational databases (NoSQL), each with different results. In this section, a comparison is presented between the response times from this paper and one of the last studies conducted in 2017 which used the same dataset for their comparison as shown in Figure (10).



Time	Water Temperature	pH	Air Temperature	Air Humidity
2017-05-04 15:00:01	78.0116	0	65.84	39.5
2017-05-04 14:59:01	78.0116	0	65.84	39.3
2017-05-04 14:58:01	78.0116	0	65.84	39.4

Figure 10: Dataset used by the DEVIN study for SQL and NoSQL comparison [21]

1. SQL comparison between DEVIN SINK in 2017[21] and this paper for inserting and updating data. Table (6), Figure (11) and Figure (12) present the results of the SQL comparison.

Table 6: Comaprsion of SQL response times between the DEVIN study and this paper for inserting data

N. record	DEVIN study Insert data	This paper Insert data	DEVIN study update data	This paer update data
300,000	100	41	50	3
500,000	220	57	100	7
1000,000	500	120	250	19

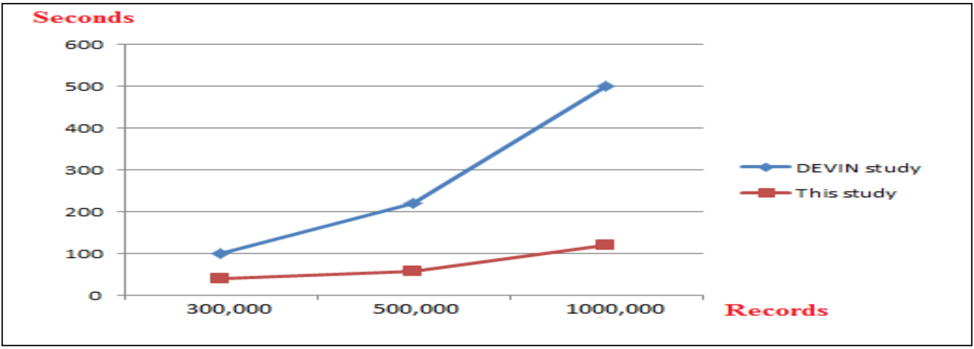


Figure 11: Comaprsion of SQL response times between the DEVIN study and this paper for inserting.

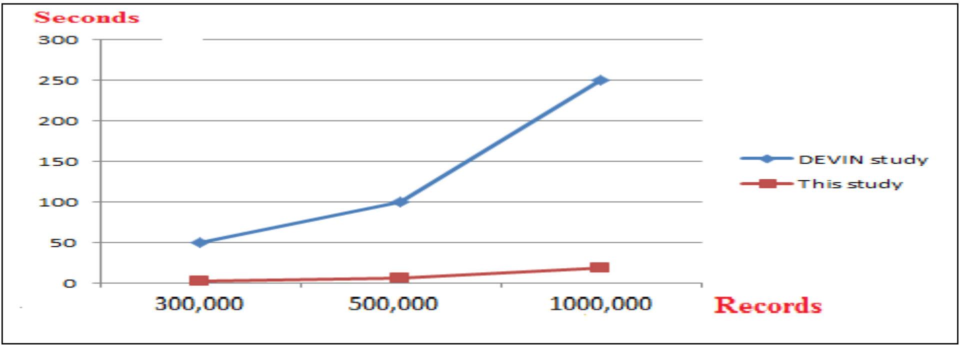


Figure 12: Comaprsion of SQL response times between the DEVIN study and this paper for updating.

2. NoSQL comparison between DEVIN SINK in 2017[21] and this paper for inserting and updating data. Table (7), Figure (13) and Figure (14) present the results of NoSQL comparison.

Table 7: Comaprsion of SQL response times between the DEVIN study and this paper for inserting

N. record	DEVIN study Insert data	This paper Insert data	DEVIN study update data	This paper update data
300,000	100	4	100	1
500,000	300	8	250	1.7
1000,000	500	17	1100	4

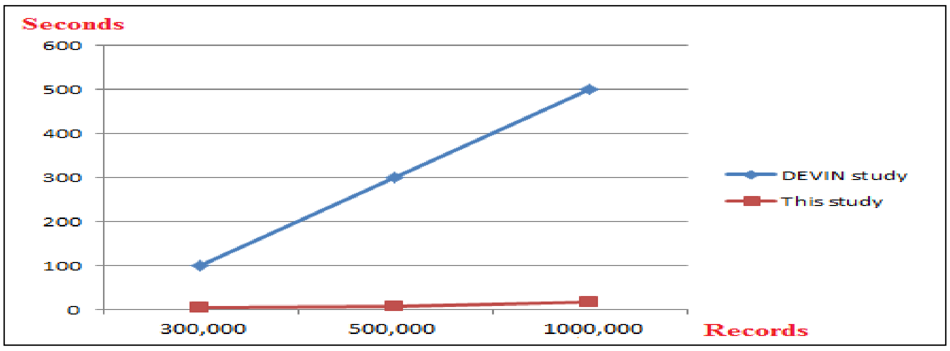


Figure 13: Comaprsion of NoSQL response times between the DEVIN study and this paper for inserting.

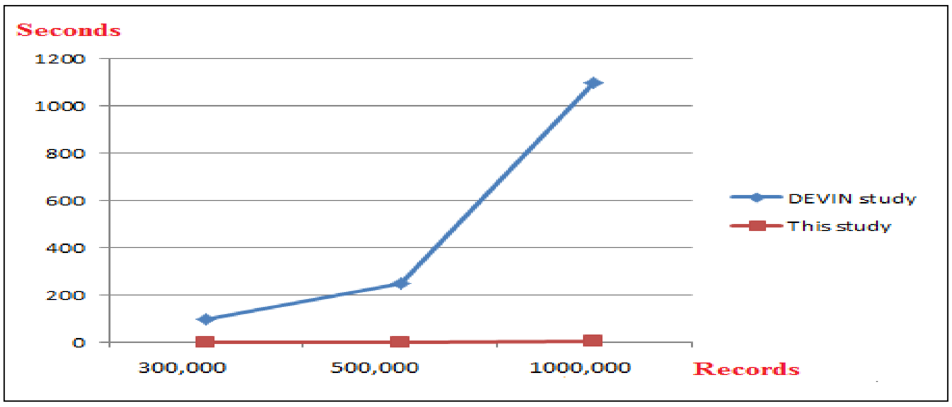


Figure 14: Comaprsion of NoSQL response times between the DEVIN study and this paper for updating

All of the above charts show that the various methods and algorithms used in this paper can improve performance compared to one of the most recent studies conducted with the same data set.

5. CONCLUSION

Big data is a term which is used to describe a massive volume of both structured and unstructured data. A real-time temperature monitoring system is one of the applications of big data; it enables the processing and measurement of massive volumes of temperature data.
In this paper, some different algorithms have been proposed to increase SQL and NoSQL database performance and the results have been compared with the most recent study in this filed.
The algorithms related to SQL database contains indexing such as a primary index for improving bulk insertion and multiple update operations at the same time.

The algorithms related to NoSQL databases, also contain indexing, batch insert (multiple insert), pipeline aggregation. Batch insert is used to insert multiple documents (by default 101 documents in one iteration), but in this paper this number can be increased to 10000 documents in one iteration, and pipeline aggregation can be used between two collections and a join can be created for matching between them without the duplication of data.
After improving performance in SQL and NoSQL, a comparison between these two databases was conducted for inserting and updating operations in all the datasets from 50,000 records to 3,000,000 records for SQL, and from 50,000 documents to 16,000,000 documents (2GB) in NoSQL. The results showed that NoSQL is much faster than SQL.

6. SUGGESTIONS FOR FUTURE WORKS

There are various possible suggestions for how far this paper can be extended, and the work can be scheduled as follows:
1.Using geospatial indexes for latitude and longitude coordinate instead of one-dimensional indexing.
3.Using sensor systems for collecting big data instead of simulation method.
4.Creating an online monitoring system. Online monitoring consists of more than just vibration sensors. Online monitoring systems can be wired, or wireless. With recent advances in battery and wireless transmission technology wireless online monitoring of machines and condition has become easier, less costly, and technologically advanced as compared to traditional solutions
5. Increasing Horizontal and vertical scaling. Horizontal scaling involves adding more machines whereas vertical scaling involves adding more power (CPU, RAM) to an existing machine.

REFERENCE

[1] Y. Arora and D. Goyal, "Big data: A review of analytics methods & techniques," in 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), 2016, pp. 225-230.
[2] C. Luo, "Survey of Parallel Processing on Big Data," 2017.
[3] V. Rubin and T. Lukoianova, "Veracity roadmap: Is big data objective, truthful and credible?," Advances in Classification Research Online, vol. 24, p. 4, 2013.
[4] G. Bello-Orgaz, J. J. Jung, and D. Camacho, "Social big data: Recent achievements and new challenges," Information Fusion, vol. 28, pp. 45-59, 2016.
[5] K. Al-Barznji and A. Atanassov, "A survey of Big Data Mining: challenges and techniques," in Proceedings of 24th International Symposium" Control of Energy, Industrial and Ecological Systems, Bankia, Bulgaria, 2016.
[6] H. K. Omar and A. K. Jumaa, "Big Data Analysis Using Apache Spark MLlib and Hadoop HDFS with Scala and Java," Kurdistan Journal of Applied Research, vol. 4, pp. 7-14, 2019.
[7] O. Kushanova, "Building, Testing and Evaluating Database Clusters: OSA project," 2014.
[8] E. Andersson and Z. Berggren, "A Comparison Between MongoDB and MySQL Document Store Considering Performance," ed, 2017.
[9] H. Ansari, "Performance Comparison of Two Database Management Systems MySQL vs MongoDB," ed, 2018.
[10] Y.-S. Kang, I.-H. Park, J. Rhee, and Y.-H. Lee, "MongoDB-based repository design for IoT-generated RFID/sensor big data," IEEE Sensors Journal, vol. 16, pp. 485-497, 2015.
[11] S. S. Nyati, S. Pawar, and R. Ingle, "Performance evaluation of unstructured NoSQL data over distributed framework," in 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2013, pp. 1623-1627.
[12] K. Fraczek and M. Plechawska-Wojcik, "Comparative analysis of relational and non-relational databases in the context of performance in web applications," in International Conference: Beyond Databases, Architectures and Structures, 2017, pp. 153-164.
[13] S. Agrawal and A. Patel, "AStudy ON GRAPH STORAGE DATABASE OF NOSQL," International Journal on Soft Computing, Artificial Intelligence and Applications (IJSCAI), vol. 5, pp. 33-39, 2016.
[14] S. Venkatraman, K. Fahd, S. Kaspi, and R. Venkatraman, "SQL versus NoSQL movement with big data analytics," Int. J. Inform. Technol. Comput. Sci, vol. 8, pp. 59-66, 2016.
[15] S. H. Aboutorabi, M. Rezapour, M. Moradi, and N. Ghadiri, "Performance evaluation of SQL and MongoDB databases for big e-commerce data," in 2015 International Symposium on Computer Science and Software Engineering (CSSE), pp. 1-7, 2015.

[16] P. Kookarinrat and Y. Temtanapat, "Analysis of range-based key properties for sharded cluster of mongodb," in 2015 2nd International Conference on Information Science and Security (ICISS), pp. 1-4, 2015.
[17] S. Ahmed, "A RESTFUL API WITH MONGODB," California State University, Sacramento, 2018.
[18] D. P. Seaman, J. J. Chaves, and K. S. Bugbee, "Benchmarking Big Data Cloud-Based Infrastructures," 2017.
[19] C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, "A comparative study: MongoDB vs. MySQL," in 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), 2015, pp. 1-6.
[20] J. Ajdari and B. Kasami, "MapReduce Performance in MongoDB Sharded Collections," International Journal Of Advanced Computer Science And Applications, vol. 9, pp. 115-120, 2018.
[21] D. Sink, "A Real-time Database System for Managing Aquarium Data," Appalachian State University, 2017.
[22] V. Abramova, J. Bernardino, and P. Furtado, "Which nosql database? a performance overview," Open Journal of Databases (OJDB), vol. 1, pp. 17-24, 2014.
[23] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing nosql mongodb to an sql db," in Proceedings of the 51st ACM Southeast Conference, p. 5, 2013.
[24] D. Merriman, E. Horowitz, and C. T. Westin, "Aggregation framework system architecture and method," ed: Google Patents, 2018.