IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 26, 2020, accepted June 8, 2020, date of publication June 15, 2020, date of current version June 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002164

Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments

MAHMOUD EYADA[1], WALAA SABER[2], MOHAMMED M. EL GENIDY[3], AND FATHY AMER[4]

[1]Mathematical and Computer Science Department, Faculty of Science, Port Said University, Port Said 42511, Egypt
[2]Electrical Engineering Department, Faculty of Engineering, Port Said University, Port Said 42511, Egypt
[3]Mathematical and Computer Science Department, Faculty of Science, Port Said University, Port Said 42511, Egypt
[4]Computer and Information Sciences Department, Faculty of Engineering, Cairo University, Cairo 12613, Egypt

Corresponding author: Walaa Saber (walaa_saber@eng.psu.edu.eg)

ABSTRACT The Internet of Things (IoT) introduces a new challenge for Database Management Systems (DBMS). In IoT, large numbers of sensors are used in daily lives. These sensors generate a huge amount of heterogeneous data that needs to be handled by the appropriate DBMS. The IoT has a challenge for the DBMS in evaluating how to store and manipulate a huge amount of heterogeneous data. DBMS can be categorized into two main types: The Relational DBMSs and the Non-relational DBMSs. This paper aims to provide a thorough comparative evaluation of two popular open-source DBMSs: MySQL as a Relational DBMS and MongoDB as a Non-relational DBMS. This comparison is based on evaluating the performance of inserting and retrieving a huge amount of IoT data and evaluating the performance of the two types of databases to work on resources with different specifications in cloud computing. This paper also proposes two prediction models and differentiates between them to estimate the response time in terms of the size of the database and the specifications of the cloud instance. These models help to select the appropriate DBMS to manage and store a certain size of data on an instance with particular specifications based on the estimated response time. The results indicate that MongoDB outperforms MySQL in terms of latency and the database size through increasing the amount of tested data. Moreover, MongoDB can save resources better than MySQL that needs resources with high capabilities to work with less performance.

INDEX TERMS IoT, DBMS, SQL, NoSQL, MySQL, MongoDB, AWS, cloud, multiple non-linear regressions.

I. INTRODUCTION

Nowadays Internet of Things (IoT) technology become the backbone of many industries like smart home systems, industrial control systems, monitoring of pharmacies and hospitals, open-source web data and weather stations. IoT is a system that is based on sensing, collecting and sharing data. This system results in exchanging operations for a large amount of IoT data. Using IoT technology generates a large amount of heterogeneous data like texts, numbers, audio, videos, and pictures. These types of data need to be transferred, processed and stored in a cloud server. Also, they need to be queried and updated on-demand. Storing and managing a large amount of IoT data efficiently is one of the major challenges. Flexible Database Management System (DBMS) is an important target to implement this challenge [1].

The DBMS is a software that is responsible for storing and managing databases. Relational DBMSs (RDBM) [2] are widely used systems. They are based on the Relational model and they use Structured Query Language (SQL) as their application programming interface language. RDBMSs work well for storing structured data and managing their relationships. However, using IoT means dealing with a large amount of heterogeneous data which has a harmful effect on the performance of the traditional RDBMSs. Non-relational DBMSs [4] are proposed to solve the limitations of the traditional RDBMSs. NoSQL databases are schema-free; they are designed to work in harmony with the unstructured data. They

The associate editor coordinating the review of this manuscript and approving it for publication was Tariq Umer.

are supposed to be easily distributed with high scalability and availability. These properties are needed to realize the vision behind the IoT from the data perspective. Managing and storing a large amount of heterogeneous data through an appropriate DBMS is one of the contemporary challenges. Performance evaluation of the Relational and Non-relational DBMS is one of the solutions that help to meet this challenge [3]–[6].

Cloud computing [7] technology enables access to a strong pool of configurable computing resources such as servers, networks, storage, services and applications; which can be rapidly provisioned and released with minimal management effort or service provider interaction. Cloud storage service is an important part of the cloud infrastructure. It can easily deal with a large amount of data. Such databases use the cloud computing paradigm to optimize consistency, availability and partition tolerance.

The main objective of this paper is to find an effective way to manage and store a large amount of heterogeneous data in appropriate DBMS. This objective is implemented by performing comprehensive experiments to compare and evaluate the performance of the two types of databases: MySQL as a Relational DBMS and MongoDB as a Non-relational DBMS. The performance metrics include latency and database size. This evaluation is implemented as follows:

- Evaluate the effect of manipulating heterogeneous IoT data on performance.
- Evaluate the increase of workloads that result from the expansion of the IoT network and increasing the number of connected sensors on the way the data is stored within the database and also on the size of stored data.
- Investigate the effect of improving the cloud instances capabilities on improving the performance.
- Provide a new way for assessing and comparing the DBMSs and choosing the most appropriate one based on formulating a prediction model to estimate the latency of the response time; in terms of realizing the database size and the instance performance.

The rest of this paper is organized as follows. Section 2 describes both background and related work. Section 3 introduces tools of hardware and software which are used in the experimental evaluations. Section 4 presents performance evaluation experiments and results. Section 5 introduces a statistical analysis of predicting the appropriate DBMS that meets user's needs.

II. BACKGROUND AND RELATED WORK

A. BACKGROUND

IoT technology is based on collecting a huge amount of data from different sources such as sensors, tracking devices, smart-phones, social media and vehicles. These types of data are emitted through the network to the appropriate cloud platform to be managed by the appropriate services. IoT applications are categorized; based on the performance requirements, into four segments: critical IoT, massive IoT, industrial automation IoT and broadband IoT. Herein the performance requirements are evaluated in terms of data size, latency, data rate, reliability and availability [8]. The essential aim of IoT technology is to manipulate the data efficiently. Servers and data centers are responsible for manipulating and storing all the received data smoothly. Then the data is processed by appropriate programming language to handle the manipulation operations. Finally, this data is stored in databases [9]. Figure 1 shows the main operations of managing and storing the IoT data.

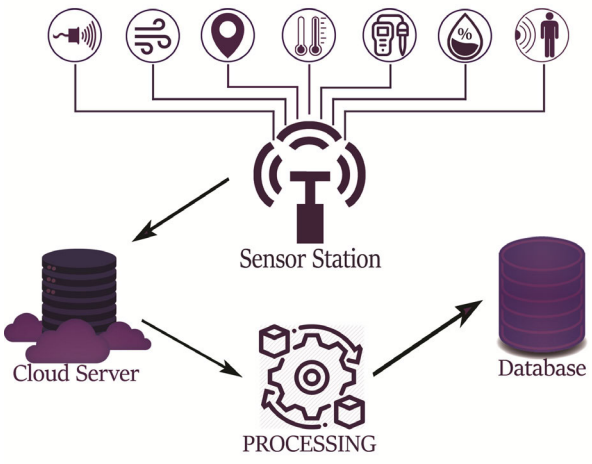

FIGURE 1. Managing and storing the IoT data.

Cloud Service Providers such as Microsoft Azure [10], Google cloud [11] and Amazon Web Services (AWS) [12] are needed for providing business applications, network services and infrastructure to the cloud users. Microsoft Azure is used for general business applications and development environments for Microsoft-centric organizations. Moreover, it is used in cloud-native applications and batch computing. Google cloud is used for cloud-native applications and batch computing as well as projects that leverage the Google cloud platform as a whole. AWS is the most popular in the Infrastructure as a Service (IaaS) market due to the highest share and compute capacity which it uses in comparison with other IaaS providers. AWS has many services, one of them is Elastic Compute Cloud (EC2) that provides the ability to create Virtual Private Server (VPS) such as your PC but with dedicated Internet Protocol Address (IP), number of GB of RAM and Processors cores besides bandwidth and storage space. It is actually one of AWS services, Amazon Simple Storage Service (Amazon S3), that is used to save the dataset in the database server.

Server-side programming languages are responsible for many tasks such as processing the collected data, interacting with servers, interacting with databases and manipulation operations over databases. PHP [13], Python [14] and Node.JS [15] are the most common used server-side programming languages. Now Node.JS is the most commonly used one because it is JavaScript operating

environment, event-driven, asynchronous programming and specially designed for network services.

Good database management systems are needed to store the collected data in an appropriate way. DBMSs are categorized into two main categories: Relational DBMSs and Non-Relational DBMSs. Relational DBMS such as MySQL [16], Oracle [17] and PostgreSQL [18] are commonly known as SQL databases. They are the most common and widely-used databases. SQL requires some predefined schemas to set the structure before you start to deal with the database. Here all your data must follow the same structure which means that a change in the structure would be both difficult and disruptive to any system. Unfortunately, using IoT networks caused generating a large amount of heterogeneous data. Traditional SQL DBMSs were not designed to solve these problems. Non-Relational DBMSs such as MongoDB [19], Cassandra [20] and Hbase [21] are commonly known as Not-only SQL (NoSQL) databases. NoSQL databases have a dynamic schema for heterogeneous data. All types of data in NoSQL databases are stored and managed in many ways such as column-oriented, KeyValue and graph-based; or organized as a document-oriented store.

MySQL is a popular open-source RDBMS that is developed, distributed and supported by Oracle Corporation. Like other Relational systems, MySQL stores data in tables and uses SQL language for database access. In MySQL you pre-define your database schema based on your requirements and set up rules to govern the relationships between fields in your tables. Any change in schema necessitates a migration procedure that can take the database offline or significantly reduce application performance. MySQL supports various types of replication services. Also, its distributed database engine is more robust than the PostgreSQL [22].

MongoDB is a document-oriented Non-relational database that can be used to distribute and store large binary files like videos and images. It stores data as documents in a binary representation called BSON (Binary JSON) objects, which are binary encoded JSON like objects. Related information is stored together for quick query access through the MongoDB query language. Documents in MongoDB can be organized in ''collections''. Fields can vary from document to document; there is no need to declare the structure of documents to the system – documents are self-describing. If a new field needs to be added to a document, then the field can be created without affecting all other documents in the collection, without updating a central system catalog and without taking the system offline. Therefore, it has a better performance than other databases in terms of resource usage and long-term storage to work with large amounts of IoT sensor data [23], [24]. MongoDB also supports indexing over embedded objects and arrays. It interacts efficiently with memory storage, complex data and dynamic queries compared to the other NoSQL databases. MongoDB is a scale-out based scheme that provides flexibility to work in case of hardware expansion [25], [26].

B. RELATED WORK

Many researches were introduced to compare between SQL and NoSQL DBMSs based on IoT data structure. The main metrics in these researches were storage, syntax, and latency of queries, database connection time and schema design. In [27], it discussed the possibility of a hybrid database between MongoDB and MySQL. It shows that using MongoDB in conjugate with MySQL improves the performance of using the MySQL database. And in [28] authors showed a disparity in performance between the data models reference, embedded and hybrid model. The hybrid model was based on merging collections to improve the performance and reduce the storage size.

In [29], [30] the performance of Relational and Non-relational DBMSs was evaluated using two models: one was implemented on a small scale of data and the other was implemented on a large scale of data. The performance was evaluated based on the CRUD operations to get the advantages and disadvantages of both kinds of DBMS. The performance was evaluated in terms of the time of queries and the size of data. However, the comparison was implemented on simple structured data and one type of MongoDB schema was examined, which didn't show the benefits of MongoDB.

In [31], [32] Relational and Non-relational databases were examined on IoT data. The experiments were implemented on MongoDB, PostgreSQL and MySQL. The results showed that NoSQL was better than SQL in different scenarios. However, fixed numbers of sensors were used in the experiments. Also, a single type of MongoDB schema was examined.

This paper is proposed to solve a lot of the previous limitations. It compares between MySQL database as a Relational database and MongoDB database as a Non-relational database in two different schema types; reference and hybrid, as a Non-relational database. More enhancements are produced on both databases to increase the flexibility and reduce the redundancy. It uses the IoT benchmark to evaluate the performance of the two types of databases on the IoT data. Moreover, different scenarios with different numbers of sensors are examined. The performance is evaluated in terms of size and latency. In addition, the experiments were implemented on different cloud servers with different hardware specifications. Furthermore, statistical analysis is introduced to predict the best DBMSs you should use depending on data size and server performance.

III. THE PROPOSED PERFORMANCE EVALUATION

A. EXPERIMENT OVERVIEW

The IoT data structure is considered as one of the main heterogeneous data structured types that need appropriate DBMS to deal with. The evaluation of DBMSs varies according to the used criteria. This paper attempts to propose comprehensive criteria for performance evaluation and try to show the behavior of each type of database through this evaluation. The aim of the proposed experiments is to evaluate and compare the performance of the two types of DBMSs: SQL

TABLE 1. Instance specification.

Name	t3.large (VM1)	t3.xlarge (VM2)	t3.2xlarge (VM3)
vCPUs	2	4	8
CPU clock speed	3.1 GHz	3.1 GHz	3.1 GHz
Instance performance (GHz)	6.2	12.4	24.8
Memory (GB)	8.0	16.0	32.0
Network burst bandwidth	5	5	5
EBS burst bandwidth	2.05	2.05	2.05

database and NoSQL database on IoT data. The main metrics in these experiments are the response time and the size of the database. This evaluation is performed in three main parts. *First*, it examines the impact of increasing workloads on the two databases. This part helps to compare the performance of the two databases handling a large scale of IoT data. *Second*, it tests the effect of improving the capabilities of the cloud instance on increasing the performance of the two databases. It helps in deciding which database can save resources better. *Third*, it proposes a prediction model that is concluded from statistical analysis on the measured data of the introduced experiments. This part applies two approaches of prediction and compares between them to estimate the latency of the data; in terms of the data size and instance performance. It also evaluates these two estimation approaches and defines which one is more accurate. The proposed prediction model provides the flexibility to evaluate and compare the two types of databases on any size of data and any instance. It selects the DBMS with low estimated latency.

B. SOFTWARE AND HARDWARE

This section discusses the utilized software and hardware in the proposed performance evaluation. Node.JS LTS version 10.16.3 and NPM version 6.10.2 are used to process the collected data. Ubuntu 16.04 LTS version is used as an operating system to setup MongoDB, MySQL and Node.JS.

Elastic Compute Cloud (EC2) is used during the implementation of this comparison. EC2 remains a core service of the AWS cloud platform. It provides a customer with an opportunity to build and host a software system on the Amazon virtual servers (EC2 Instances). EC2 is a virtual private server (VPS) within a cloud, where storage can be resizable and almost unlimited. With Ec2 service, three types of instances were used t3.large (referred to as VM1), t3.xlarge (referred to as VM2) and t3.2xlarge (referred to as VM3). The T3 instances feature is the Intel Xeon Platinum 8000 series (Skylake-SP) processor with a sustained all core Turbo CPU clock speed of up to 3.1 GHz. Additionally, there is a support for the new Intel Advanced Vector Extensions 512 (AVX-512) instruction set [33]. Table 1 shows the specifications of the instances.

C. DATABASE SETUP

1) IoT BENCHMARK

The pollution database [34] is used as a base for this paper. This database is based on collecting information about elements of air pollution indoors and outdoors. The indoor air frequently and severely polluted more than the outdoor air, which can be refined naturally. The IoT sensor data that depends on the temporal status and variables related to the spatial characteristics of the space being measured should be considered differently from other homogeneous inputs, such as image and audio, because it considers heterogeneous data [35]. Some changes are added to increase the flexibility and decrease the redundancy for working with the intended database. The database can be partitioned into three parts: the sensors part, the location longitude and latitude part and the timestamp part. The first part is concerned with adding a new sensor to a specific station. This part is a multi-value part so it can be changed from adding a new field for each new sensor to add a new record for each new sensor by converting the sensors data fields (i.e. the number of fields is equal to the number of all sensors) to only two fields: sensor_id field and sensor_data field. This change increases flexibility as it allows increasing any number of sensors to the database without affecting its structure.

The second part is concerned with storing the longitude and latitude of the location. This part can be normalized into an independent table or collection that contains three fields: station_id, longitude and latitude for each station location in addition to the original table or collection but with replacing longitude and latitude fields with station_id field. This change reduces the redundancy in the original data as for every sensor data insertion to a table or collection there is no need to add the station location. The third part, the timestamp part, is concerned with instant receiving of sensor data. This part of data is important and must be added to the original table or collection.

With these changes, the database becomes more flexible to deal with MySQL and MongoDB DBMSs. After editing the database, it becomes able to receive and store data from any number of towns, stations and sensors; besides saving the locations of every station the longitude and the latitude with Date and Time of every record.

2) MYSQL DATABASE SETUP

MySQL server 8.0.11 is the version that is used in this evaluation. Table 2 shows the SQL statements that are used to create the tested MySQL database schema specifying the structure of a database for managing a series of town's base stations and the related sensors. Two tables are created for MySQL schema: station_location and town_name.

The station_location table is to save the location of every station. Referential integrity constraint is implemented with FOREIGN KEY to stop any station to be inserted in the town table without being mentioned in the stations_locations. The main target of dividing the dataset in two tables is to reduce the data redundancy.

3) MONGODB DATABASE SETUP

MongoDB version 4.2 is the current stable release version that is applied in this comparison. As in MySQL, two

TABLE 2. MySQL creation queries.

```
CREATE TABLE station_location
(
station_id   VARCHAR (20)  NOT NULL,
Longitude      FLOAT          NOT NULL,
Latitude       FLOAT          NOT NULL,
PRIMARY KEY (station_id)
)
```

```
CREATE TABLE town_name
(
station_id   VARCHAR (20)     NOT NULL,
sensor_id   VARCHAR (20)     NOT NULL,
sensor_data      FLOAT          NOT NULL,
date_time   TIMESTAMP,
PRIMARY KEY (station_id, sensor_id, sensor_data, date_time),
FOREIGN KEY (station_id) REFERENCES stations_locations
(station_id)
)
```

TABLE 3. MongoDB creation queries.

```
db.createCollection ("station_location")
db.stations_locations.createIndex ({"station_id": 1}, {unique: true})
stations_locations:{
  station_id: String,
  longitude: Number,
  latitude: Number }
```

Hybrid Model	Reference Model

db.createCollection
("town_name")
towns_name:{
station_id: String,
sensors: [
{
sensor_id: String,
sensor_data: Number
}
],
date_time:
{type: Date, default: Date.now} }
	db.createCollection
	("town_name")
	towns_name:{
	station_id: String,
	sensor_id: String,
	sensor_data: Number,
	date_time:
	{type: Date, default: Date.now}
	}

collections are created for MongoDB. The first one saves the location of every station, so no record can be inserted in town_name collection from any station that is not inserted in stations_locations collection.

The second collection is the sensors table of all sensors in every station in the town. As stations_locations collection, sensors table controls the insertion query in the town_name collection. Thus, there is no record from the sensor can be inserted when it is not available in sensors collection first. But in MongoDB, the insertion is controlled with insertion query and is not automatically inserted as in the MySQL from the creation query.

MongoDB as NoSQL DBMS has three models to deal with data set: reference, embedded and hybrid. The hybrid model occupies less size of memory than the other reference and embedded models [27]. Table 3 shows the creation collection statements for the tested MongoDB database based on reference and hybrid models.

IV. EXPERIMENT AND RESULT

A. EXPERIMENT ANALYSIS

The proposed experiments manage the data collected through a fixed number of stations (i. e. in this case four stations)

TABLE 4. Database description.

Number of towns	1
Number of stations per the town	4
Station transmitting rate	1000 records / transmission
Number of sensors per each station	1:12

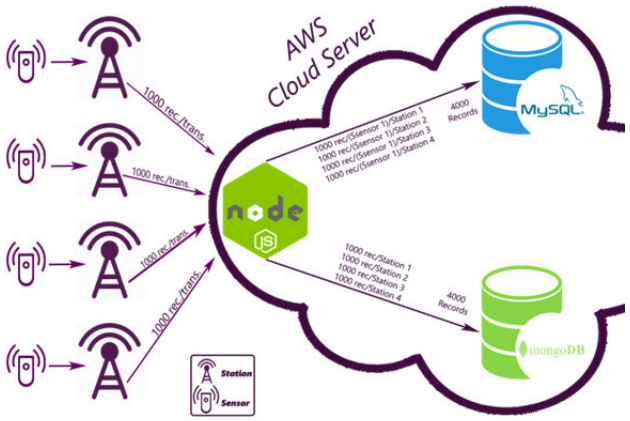

FIGURE 2. One sensor scenario.

and located within one town. The structure of the proposed database is described in Table 4. Each station collects and buffers data every second from its related sensor nodes and sends it to the cloud with a rate of 1000 records per transmission. Several scenarios have been introduced to illustrate the most important database operations. The proposed experiments scenarios were carried out based on increasing the number of sensors connected to each station from 1 to 12 sensors. This leads to increasing number of sensors for the experiments from 4 to 48 by increasing one sensor per station in each experiment.

B. IMPACT OF INCREASING THE NUMBER OF SENSOR NODES ON THE INSERTION OPERATION

Increasing the workloads in the proposed experiments is represented by increasing the number of sensors connected to each station. In each experiment, each station collects the data from all its related connected sensors in data records. The data record contains readings of all the related sensors at a specific time. These collected data records are inserted as records in MySQL and as record objects in MongoDB. To shed light on the effect of increasing the number of connected sensors on the number of inserted records in both databases, two scenarios are introduced in detail.

In the first scenario, one sensor node is connected to each station as shown in Figure 2. After the stations collect data from the sensor nodes, each station sends its data with a rate of 1000 data records per transmission to the cloud for processing and insertion operations. Four stations are transmitting 4000 data records at a time. Each record contains data from one sensor node. Each record is inserted in one record in case of MySQL and one record object in case of MongoDB. The

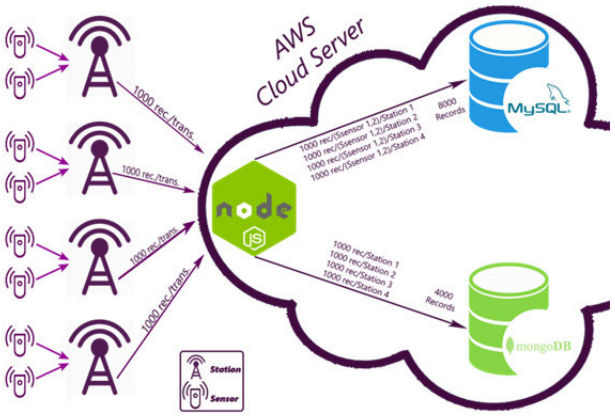

FIGURE 3. Two sensors scenario.

transmitted data is inserted in 4000 records in MySQL and 4000 record objects in MongoDB.

In the second scenario, two sensor nodes are connected to each station as shown in Figure 3. The four stations transmit 4000 data records at a time for the cloud. Each record contains data from two sensor nodes. In MySQL, the data for each sensor needs to be inserted in a separate record. It needs 8000 records to insert the transmitted data. Unlike MongoDB, all sensors data can be inserted into one record object. So, it needs 4000 record objects to insert the transmitted data.

The remaining scenarios are introduced in Table 5 to show the number of inserted records in each case. Due to the structured schema of MySQL, the addition of a new sensor is represented by adding an independent record for this sensor measurement. Therefore, the data record received from the station is separated into independent records one per each sensor data. While MongoDB is a document-oriented database and the addition of new sensor results in a document with a different structure for the newly inserted record object. This means the data record received from the station is inserted in one record object. Increasing the number of connected sensors means increasing the number of records (i.e. vertical increasing) in MySQL and increasing the width of the record object (i.e. horizontal increasing) in MongoDB.

C. IMPACT OF INCREASING THE WORKLOADS ON THE LATENCY

In the proposed scenarios, increasing the workloads is based on increasing the number of connected sensors per station from 1 to 12 sensors and therefore increasing the number of insertion operations. These insertion operations are implemented on three instances with different specifications VM1, VM2 and VM3; which were described in Table 1.

MySQL accepts from 4000 records; in the first insertion operation, to 48000 records; in the last insertion operations. On the other side, MongoDB accepts 4000 records in all cases from the first to the last operations. Table 5 shows the latency of these insertion operations in all cases.

TABLE 5. The latency of the insertion operation.

Sensors number	Database Management Systems							
	MySQL				MongoDB			
	Record numbers (k)	Instance type			Record numbers (k)	Instance type		
		VM1	VM2	VM3		VM1	VM2	VM3
		Latency in (ms)				Latency in (ms)		
1	4	5000	4000	4000	4	885	670	650
2	8	11000	8000	7000	4	948	709	678
3	12	17000	12000	10000	4	1020	750	707
4	16	23000	17000	16000	4	1095	780	729
5	20	29000	22000	19000	4	1152	813	752
6	24	36000	27000	21000	4	1217	852	781
7	28	42000	32000	24000	4	1279	888	811
8	32	48000	37000	27000	4	1334	922	840
9	36	54000	42000	30000	4	1412	966	873
10	40	60000	47000	33000	4	1481	1008	904
11	44	66000	52000	36000	4	1551	1054	939
12	48	72000	57000	39000	4	1623	1101	970

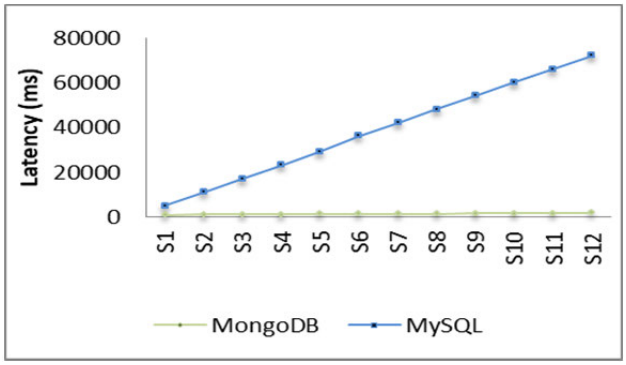

FIGURE 4. Insert query Instance T3.large.

Figure 4 shows the impact of increasing workloads on the latency using VM1 with a performance of 6.2 GHz. It shows that in the first scenario MongoDB decreases the latency compared with MySQL by 82.3%. By increasing the number of connected sensors and in the last scenario, MongoDB decreases the latency compared with MySQL by 97.7%.

Figure 5 shows the impact of increasing the same workloads on the latency using VM2 with a performance of 12.4 GHz. Using VM2 improves the performance of MySQL with 29% over using VM1while it improves the performance of MongoDB with 23% over using VM1. It also shows that in the first scenario, MongoDB decreases the latency compared with MySQL by 83.3%. Also, in the last scenario MongoDB decreases the latency compared with MySQL by 98%.

Figure 6 shows that MongoDB decreases the latency compared with MySQL by 83.7% in the first scenario and by 97.5% in the last scenario using VM3 with a performance

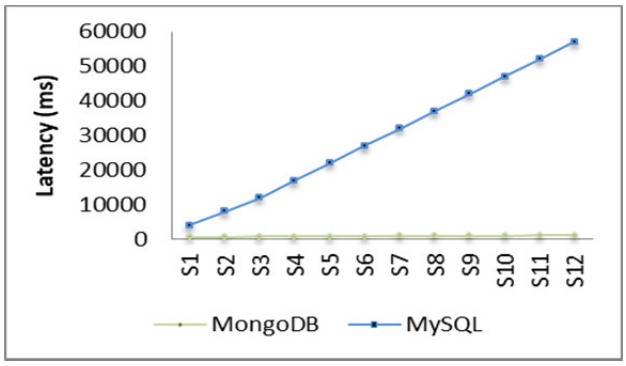

FIGURE 5. Insert query Instance T3.xlarge.

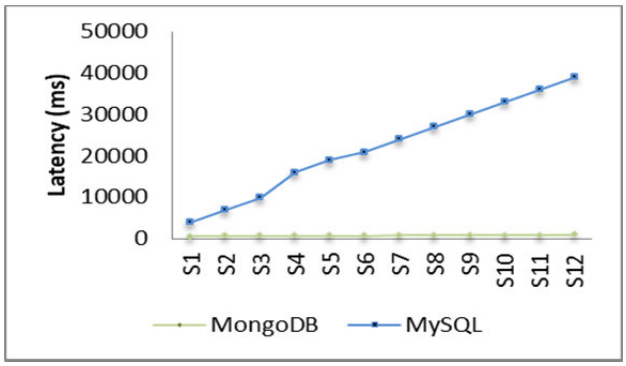

FIGURE 6. Insert query Instance T3.2xlarge.

of 24.8 GHz. Using VM3 improves the performance of MySQL with 43% over using VM1 and with 26% over using VM2. Besides, it improves the performance of MongoDB with 35% over using VM1 and with 8% over using VM2.

The results show that through increasing the number of connected sensors per station and accordingly increasing workloads, the difference in latency between MongoDB and MySQL is also increased in favor of MongoDB. Moreover, the results show that using MongoDB on VM1with low capabilities has average improvement in latency with 43% over MySQL on VM3 with high capabilities. This ensures that MySQL needs instances with high capabilities to work well on the contrary to MongoDB which works very well with all instances capabilities.

D. IMPACT OF INCREASING THE WORKLOADS ON THE DATABASE SIZE

In this section, two types of schemas for the MongoDB database are examined and compared with MySQL database in terms of the database size: hybrid and reference models. Table 6 shows the effect of the same experiments on the database size (i.e. size of the stored data).

Figure 7 shows the impact of increasing the workloads on the three databases: MongoDB based hybrid model, MongoDB based reference model and MySQL databases. It shows that in MongoDB based hybrid model, the database size is increased linearly with increasing the workloads. Also,

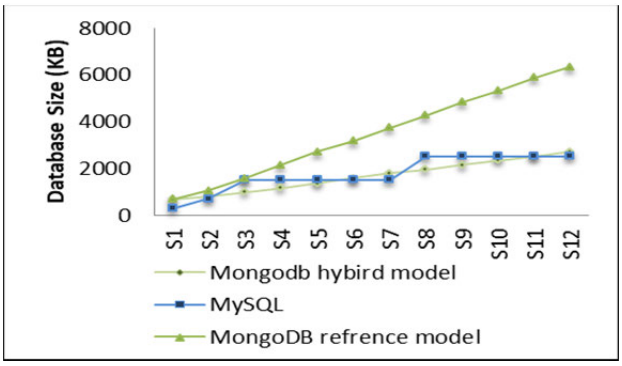

FIGURE 7. The storage size of MongoDB based hybrid model, MongoDB based reference model, and MySQL databases.

TABLE 6. Database size comparison between MongoDB based Hybrid Model, MongoDB based reference model, and MySQL.

Sensors number	Database Management Systems					
	MySQL		MongoDB hybrid model		MongoDB References model	
	Record number (k)	Database Size (KB)	Record numbers (k)	Database Size (KB)	Record numbers (k)	Database Size (KB)
1	4	664	4	682	4	682
2	8	798	4	1040	8	1500
3	12	1500	4	972	12	1560
4	16	1500	4	1156	16	2135
5	20	1500	4	1355	20	2700
6	24	1500	4	1562	24	3170
7	28	1780	4	1920	28	3730
8	32	2500	4	1932	32	4265
9	36	2500	4	2136	36	4835
10	40	2500	4	2340	40	5300
11	44	2500	4	2512	44	5870
12	48	2500	4	2705	48	6340

in MongoDB based reference model, the database size is increased linearly but the rate of increase is higher than the database size of MongoDB based hybrid model. On the other side in MySQL, the database size has periods of linear increase and periods of relative stability. According to scenarios from S1 to S3 and from S7 to S8, database size is increased linearly with increasing the workloads. Whereas the scenarios from S3 to S7 and from S8 to S12, database size is relatively stable with increasing the workloads. This behavior of MySQL is due to its storage nature [4].

The results show that MySQL outperforms the other two databases in case of inserting small number of records and also at the end of each stability period as shown in the highlighted records of table 6 scenarios S1, S2, S6, S7, S11 and S12. For inserting small number of records up to scenario S2, MySQL is much more efficient than the MongoDB based hybrid model by 15% and the MongoDB based reference model by 33%. Also, by increasing the number of inserted

Station_id	Sensor_id	Sensor_data	Date_time
Staion_1	S_1	22.5	2019-05-25T10:45:24Z
Staion_2	S_1	22.6	2019-05-25T10:45:24Z
Staion_3	S_1	22.4	2019-05-25T10:45:24Z
Staion_4	S_1	21.22	2019-05-25T10:45:24Z

FIGURE 8. MySQL records in the case of one sensor scenario.

```
{ station_id:"station_1",
  sensors:[ {   sensor_id:"s_1",  sensor_data:22.31  } ],
  date_time:"2019-05-25T10:05:44Z"},
{ station_id:"station_2",
  sensors:[ {   sensor_id:"s_1",  sensor_data:22.31  } ],
  date_time:"2019-05-25T10:05:44Z"},
{ station_id:"station_3",
  sensors:[ {   sensor_id:"s_1",  sensor_data:12.31  } ],
  date_time:"2019-05-25T10:05:44Z"},
{ station_id:"station_4",
  sensors:[ {   sensor_id:"s_1",  sensor_data:23.1  } ],
  date_time:"2019-05-25T10:05:44Z"}
```

FIGURE 9. MongoDB record objects in the case of one sensor scenario.

records and at the end of each stability period; scenarios from S6 to S7 and from S11 to S12, MySQL outperforms the other two databases. Whereas, at the beginning of each stability period; scenarios from S3 to S5 and from S8 to S10, MongoDB based hybrid model outperforms the other two databases. In addition, the results also show that MongoDB based hybrid model has average improvement in the database size by 6.6% over MySQL and 51.5% over MongoDB based reference model.

E. IMPACT OF INCREASING THE NUMBER OF SENSOR NODES ON THE SELECTION OPERATION

In this section, the selection operation of both databases is evaluated in terms of latency. It is based on retrieving the data of all the connected sensors at a specific time. The selection operations are started by selecting the data that is collected based on one sensor per each station (i.e. the first scenario) and end with selecting the data that is collected based on 12 sensors per each station (i.e. the last scenario), twelve scenarios and selection operations are implemented. The data retrieved in the case of one connected sensor per station is shown in Figures 8 and 9. Also, the data retrieved in the case of two connected sensors per station is shown in Figures 10 and 11. Table 7 shows the impact of the selection operations on the latency in different instances VM1, VM2, and VM3.

The selection operation considers a connection time that connects the DBMS with the database server before retrieving the data. In the proposed experiments, this time is 112 ms in case of MongoDB, and 5 ms in case of MySQL. Figures 12, 13, and 14 show that MongoDB is faster than MySQL in all scenarios when implemented on instances VM1, VM2, and VM3. As a result, to increase the performance of the instance, the latency decreases, the same operation is processed on the three instances but with better performance.

Station_id	Sensor_id	Sensor_data	Date_time
Staion_1	S_1	22.5	2019-05-25T11:25:22Z
Staion_1	S_2	55.1	2019-05-25T11:25:22Z
Staion_2	S_1	22.6	2019-05-25T11:25:22Z
Staion_2	S_2	44.11	2019-05-25T11:25:22Z
Staion_3	S_1	3.4	2019-05-25T11:25:22Z
Staion_3	S_2	25.21	2019-05-25T11:25:22Z
Staion_4	S_1	21.45	2019-05-25T11:25:22Z
Staion_4	S_2	33.21	2019-05-25T11:25:22Z

FIGURE 10. MySQL records in the case of two sensor scenario.

```
{ station_id:"station_1",
  sensors:[ { sensor_id:"s_1",  sensor_data:22.31  },
            { sensor_id:"s_2",  sensor_data:22.31  } ],
  date_time:"2019-05-25T10:22:44Z"},
{ station_id:"station_2",
  sensors:[ { sensor_id:"s_1",  sensor_data:22.31  },
            { sensor_id:"s_2",  sensor_data:50.31  } ],
  date_time:"2019-05-25T10:22:44Z"},
{ station_id:"station_3",
  sensors:[ { sensor_id:"s_1",  sensor_data:22.31  },
            { sensor_id:"s_2",  sensor_data:60.31  } ],
  date_time:"2019-05-25T10:22:45Z"},
{ station_id:"station_4",
  sensors:[ { sensor_id:"s_1",  sensor_data:22.31  },
            { sensor_id:"s_2",  sensor_data:72.31  } ],
  date_time:"2019-05-25T10:22:45Z"}
```

FIGURE 11. MongoDB record objects in the case of two sensor scenario.

TABLE 7. The latency of the selection operation.

Sensors number	Database Management Systems							
	MySQL				MongoDB			
	Record numbers (k)	Instance type			Record numbers (k)	Instance type		
		VM1	VM2	VM3		VM1	VM2	VM3
		Latency in (ms)				Latency in (ms)		
1	4	140	125	96	4	71	56	52
2	8	180	171	145	4	87	68	66
3	12	240	228	189	4	103	87	82
4	16	345	274	230	4	123	104	93
5	20	462	320	247	4	140	116	105
6	24	598	381	315	4	168	133	117
7	28	781	437	349	4	201	149	132
8	32	995	495	391	4	238	165	148
9	36	1258	566	429	4	277	186	160
10	40	1552	633	476	4	315	204	178
11	44	1908	702	522	4	356	230	195
12	48	2308	776	568	4	409	264	217

The results show that VM3 has an average improvement of the selection latency with 63.2% over VM1 and with 22.5% over VM2 in case of MySQL. Also, VM3 has an average improvement of the selection latency with 38% over VM1 and with 12.3% over VM2 in case of MongoDB. These results ensure that MySQL is affected; more than MongoDB, by the instance performance.

The comprehensive results ensure that MongoDB database performance can significantly outperform the MySQL

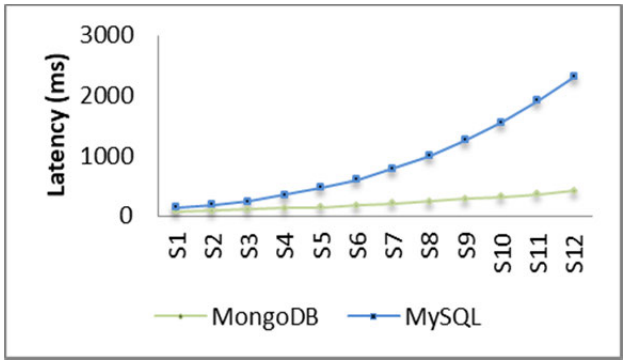

FIGURE 12. Select query Instance T3.large.

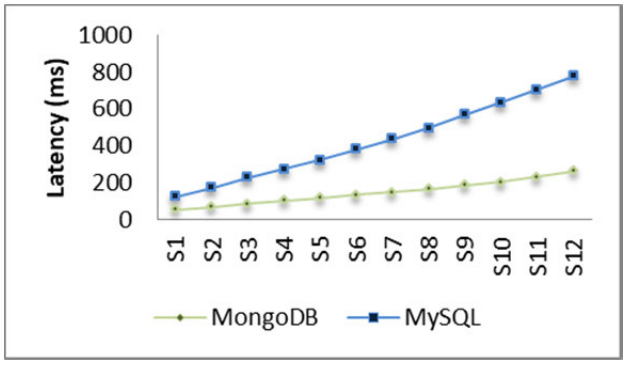

FIGURE 13. Select query Instance T3.xlarge.

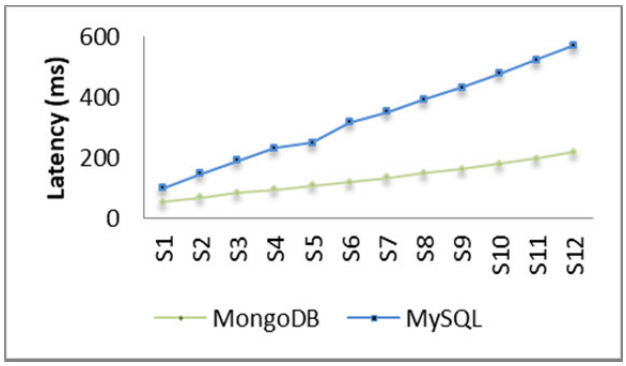

FIGURE 14. Select query Instance T3.2xlarge.

database dealing with a wide range of IoT data. In addition, the results show that MySQL can work well with high-performance instance, unlike MongoDB which works very well with all instance specifications.

V. STATISTICAL ANALYSIS

In this section, a statistical analysis is introduced to estimate the latency of data from a measured data size and instance performance using two approaches: Multiple Linear Regression and Multiple Non-linear Regression. This estimation is implemented on both database types, MySQL and MongoDB, for twofold aims. The first aim is to compare the two types of databases in terms of latency. The second aim is to evaluate the two estimation approaches and select the appropriate one.

A. DATASETS AND METHODS

1) DATASET

The analysis starts by establishing the equations models from a previously measured dataset which contains three parameters; namely latency in milliseconds, data size in KB and instance performance in GHz.

The instance performance was calculated as the product of the virtual central processing unit (vCPU) and the CPU clock speed as shown in Table 1. The data size was calculated as the product of the calculated average size of one record and the number of records from Table 6. Table 8 shows the dataset information that is needed in this analysis for both MySQL and MongoDB databases.

2) MULTIPLE REGRESSION ANALYSIS

This section is proposed to compare two approaches for estimating the latency of the database: Multiple Linear Regression and Multiple Non-linear Regression. This estimation is used to evaluate the performance of both MySQL and MongoDB databases. Residual value is the metric that is used in this comparison.

3) LINEAR REGRESSION

A simple Linear Regression is used to illustrate the relation between the dependent variable y and the independent variable x based on the regression equation [36].

$$y = a_1 x_1 + a_0 \tag{1}$$

The proposed evaluation needs to find a relation between three variables: latency, data size and instance performance. The dependent variable (latency) is related to two independent variables (data size and instance performance). In this case, the Multiple Linear Regression can be implemented as follows [35]:

$$y = a_2 x_2 + a_1 x_1 + a_0 \tag{2}$$

where x_1 is the data size, x_2 is the instance performance and y is the latency. Table 9 shows the parameters notations which are used in this evaluation.

The determiner method is used to solve the previous equation and get the final equation of prediction for both MongoDB and MySQL latency as [36]:

$$\begin{vmatrix} y & x_1 & x_2 & 1 \\ \sum_{i=1}^{n} y & \sum_{i=1}^{n} x_1 & \sum_{i=1}^{n} x_2 & n \\ \sum_{i=1}^{n} x_1 y & \sum_{i=1}^{n} x_1^2 & \sum_{i=1}^{n} x_1 x_2 & \sum_{i=1}^{n} y \\ \sum_{i=1}^{n} x_2 y & \sum_{i=1}^{n} x_1 x_2 & \sum_{i=1}^{n} x_2^2 & \sum_{i=1}^{n} x_2 \end{vmatrix} = 0 \tag{3}$$

By substituting in equation (3) with values from Table 8 to get the values a_2, a_1, and a_0.

TABLE 8. Dataset information: latency, data Size, and Instance performance.

Sensors number	Database Management Systems							
	MySQL				MongoDB			
	Data Size in (KB)	Instance performance Performance			Data Size in (KB)	Instance performance Performance		
		VM1	VM2	VM3		VM1	VM2	VM3
		Latency in (ms)				Latency in (ms)		
1	168	140	125	96	604	71	56	52
2	336	180	171	145	788	87	68	66
3	504	240	228	189	972	103	87	82
4	672	345	274	230	1156	123	104	93
5	840	462	320	247	1340	140	116	105
6	1008	598	381	315	1532	168	133	117
7	1176	781	437	349	1724	201	149	132
8	1344	995	495	391	1916	238	165	148
9	1512	1258	566	429	2108	277	186	160
10	1680	1552	633	476	2300	315	204	178
11	1848	1908	702	522	2492	356	230	195
12	2016	2308	776	568	2684	409	264	217

MongoDB equation will be:

$$y = -3.83x_2 + 0.111x_1 + 35.35 \tag{4}$$

MySQL equation will be:

$$y = -27.257x_2 + 0.581x_1 + 310.345 \tag{5}$$

4) NON-LINEAR REGRESSION

Non-Linear Regression is another approach that can be used to estimate the database latency based on the perception of both data size and instance performance. In this case, the Multiple Non-linear Regression can be implemented as follows [7]:

$$y = a_5x_1^2 + a_4x_2^2 + a_3x_1x_2 + a_2x_1 + a_1x_2 + a_0 \tag{6}$$

where x_1 is the data size, x_2 is the instance performance and y is the latency.

TABLE 9. Parameters' notations used in the proposed evaluation.

Parameter	MEANING
y	Latency in milliseconds (ms)
x_1	Data size in KB
x_2	Instance performance in GHz
$\hat{y}_i$	The estimated value of the latency
$\bar{y}$	The average of the estimated values
R^2	R-Squared value
R^2_{adj}	Adjusted R-squared value
SSE	Sum of squares error.
SST	Sum of squares total.
n	Number of data sample points.
k	Number of variables in the model.

The determiner method is used to solve the previous equation and get the final equation of prediction for both MongoDB and MySQL latency [37] as shown in equation (7).

By substitution; in equation (7), with values from Table 8 to get the values a_5, a_4, a_3, a_2, a_1, and a_0.

MongoDB equation will be:

$$y = 0.446x_2^2 + 0.004x_1x_2 + 0.093x_1 - 11.305x_2 + 76.0968 \tag{8}$$

MySQL equation will be:

$$y = 3.673x_2^2 - 0.0424x_1x_2 + 0.694x_1 - 98.236x_2 + 544.357 \tag{9}$$

5) R-SQUARED AND ADJUSTED R-SQUARED

R-Squared is a statistical measure that predicts the future outcome of an investment and how closely it aligns to a single measured model [8], [39]. It is represented as the difference between the observed value of the dependent variable for the observation y_i and the estimated value of the dependent variable for the observation $\hat{y}_i$ [7]:

$$R^2 = \frac{SSE}{SST} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{10}$$

$$\begin{vmatrix} y & x_1^2 & x_2^2 & x_1x_2 & x_1 & x_2 & 1 \\ \sum_{i=1}^{n} y & \sum_{i=1}^{n} x_1^2 & \sum_{i=1}^{n} x_2^2 & \sum_{i=1}^{n} x1 * x2 & \sum_{i=1}^{n} x1 & \sum_{i=1}^{n} x_2 & n \\ \sum_{i=1}^{n} x_1^2y & \sum_{i=1}^{n} x_1^4 & \sum_{i=1}^{n} x_1^2x_2^2 & \sum_{i=1}^{n} x_1^2x_1x_2 & \sum_{i=1}^{n} x_1^2x_1 & \sum_{i=1}^{n} x_1^2x_2 & \sum_{i=1}^{n} x_1^2y \\ \sum_{i=1}^{n} x_2^2y & \sum_{i=1}^{n} x_1^2x_2^2 & \sum_{i=1}^{n} x_2^4 & \sum_{i=1}^{n} x_2^2x_1x_2 & \sum_{i=1}^{n} x_2^2x_1 & \sum_{i=1}^{n} x_2^2x_2 & \sum_{i=1}^{n} x_2^2y \\ \sum_{i=1}^{n} x_1x_2y & \sum_{i=1}^{n} x_1^2x_1x_2 & \sum_{i=1}^{n} x_2^2x_1x_2 & \sum_{i=1}^{n} x_1^2x_2^2 & \sum_{i=1}^{n} x_2x_1^2 & \sum_{i=1}^{n} x_1x_2^2 & \sum_{i=1}^{n} x_1x_2y \\ \sum_{i=1}^{n} x_1y & \sum_{i=1}^{n} x_1^2x_1 & \sum_{i=1}^{n} x_2^2x_1 & \sum_{i=1}^{n} x_1^2x_2 & \sum_{i=1}^{n} x_1^2 & \sum_{i=1}^{n} x_1x_2 & \sum_{i=1}^{n} x_1y \\ \sum_{i=1}^{n} x_2y & \sum_{i=1}^{n} x_1^2x_2 & \sum_{i=1}^{n} x_2^2x_2 & \sum_{i=1}^{n} x_2^2x_1 & \sum_{i=1}^{n} x_1x_2 & \sum_{i=1}^{n} x_2^2 & \sum_{i=1}^{n} x_2y \end{vmatrix} = 0 \tag{7}$$

TABLE 10. The values of R^2_{adj} in different cases.

Multiple Linear Regressions		Multiple Non-linear Regression	
MongoDB	MySQL	MongoDB	MySQL
0.857	0.636	0.974	0.896

where R^2 is R-Squared, $\bar{y}$ is the average of the estimated values, SSE is the sum of squares error and SST is the sum of squares total.

SSE and SST are calculated as follows [7]:

$$SSE = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 \tag{11}$$

$$SST = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{12}$$

The adjusted R-squared compares the correlation of the investment to several measured models as [7]:

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(n-1)}{n-k-1} \right] \tag{13}$$

where R^2_{adj} is the adjusted R-squared.

Table 10 shows the adjusted R-squared values for both databases based on the two approaches of regressions. High adjusted R-squared means good performance in the estimation processand vice versa. The results show that both the Linear Regression and Non-linear Regression approaches provide better estimation in case of MongoDB than in case of MySQL. Unfortunately, adjusted R-squared is very low 0.636 in case of MySQL based on Linear Regression and cannot be used. To solve this problem, the used dataset is divided in two sets: the first one is from 1 to 6 sensors and the second set is from 7 to 12 sensors.

MySQL first set (1:6) Linear Regressions Equation:

$$y = -6.238x_2 + 0.366x_1 + 135.383 \tag{14}$$

MySQL second set (7:12) Linear Regressions Equation:

$$y = -48.276x_2 + 0.827x_1 + 220.148 \tag{15}$$

MySQL first set (1:6) Non-linear Regressions Equation:

$$y = 0.473x_2^2 - 0.015x_1x_2 + 0.399x_1 - 12.659x_2 + 141.239 \tag{16}$$

MySQL second set (7:12) Non-linear Regressions Equation:

$$y = 6.874x_2^2 - 0.073x_1x_2 + 0.9264x_1 - 150.557x_2 + 591.552 \tag{17}$$

Table 11 shows that dividing the dataset into two sets improving the adjusted R-squared and therefore the performance of estimation in the case of MySQL. However Non-linear Regression approach still provides good performance in estimation 0.974 than the Linear Regression approach. The results show that the Non-linear Regression approach provides better estimation results for MongoDB. On the other hand, after dividing the dataset into two sets, the two Non-linear Regression equations of the divided dataset is the best for MySQL with adjusted R-squared 0.959, and 0.951 for both the sets of data.

TABLE 11. The values of R^2_{adj} in MYSQL cases.

Multiple Linear Regressions		Multiple Non-linear Regressions	
MySQL (1:6)	MySQL (7:12)	MySQL (1:6)	MySQL (7:12)
0.845	0.613	0.959	0.951

VI. CONCLUSION AND FUTURE WORK

In this paper, a comparison between MongoDB and MySQL has been implemented to evaluate the performance of both databases to deal with a large scale of heterogeneous IoT data. Several scenarios were introduced to perform this target. The results showed that increasing workloads in case of using MySQL leads to a considerable loss in performance greater than the case of using MongoDB. Three instances with different capabilities are used to compare the performance improvement for both databases. The proposed evaluation helps to differentiate between the two types of databases based on database workloads, the available resources capabilities, the number of network connected-sensors and the needed requirements. The results showed that there are cases where MongoDB is better than MySQL in terms of both latency and database size. On the other hand, there are other cases where MongoDB is better in terms of latency and MySQL is better in terms of database size; based on the database workload and the number of connected sensors. The differentiation in these cases is based on the available resources capabilities.

Two data models of the MongoDB are introduced: reference and hybrid. The results show that the hybrid model is better than the reference model and also better than MySQL in terms of database size. Two prediction models of estimating the latency from measured data were introduced based on two approaches: linear Regression and Non-linear Regression. The results showed that Non-linear Regression is better than Linear Regression for estimating the latency in both MongoDB and MySQL. A benefit from applying the regressions approach is to select the appropriate DBMS based on the low estimated latency.

As a future work for this paper, a hybrid model between MongoDB and MySQL can be implemented and compared against MySQL and MongoDB. Also, the evaluation process can be performed using big-data workloads. Finally, the estimation process can be implemented using the neural network instead of the Regression process.

REFERENCES

[1] L. Gutierrez-Madronal, L. La Blunda, M. F. Wagner, and I. Medina-Bulo, "Test event generation for a fall-detection IoT system," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6642–6651, Aug. 2019.

[2] B. Diene, J. Rodrigues, O. Diallo, E. Ndoye, and V. V. Korotaev, "Data management techniques for Internet of Things," *Mech. Syst. Signal Process.*, vol. 138, Apr. 2020, Art. no. 106564.
[3] S. Kontogiannis, C. Asiminidis, and G. Kokkonis, "Comparing relational and NoSQL databases for carrying IoT data," *J. Sci. Eng. Res.*, vol. 6, no. 1, pp. 125–133, 2019.
[4] R. Čerešnák and M. Kvet, "Comparison of query performance in relational a non-relation databases," *Transp. Res. Procedia*, vol. 40, pp. 170–177, Jan. 2019.
[5] B. Jose and S. Abraham, "Analysis of aggregate functions in relational databases and NoSQL databases," *Int. J. Comput. Sci. Eng.*, vol. 6, no. 6, pp. 74–79, Jul. 2018.
[6] W. Ali, M. U. Shafique, M. A. Majeed, and A. Raza, "Comparison between SQL and NoSQL databases and their relationship with big data analytics," *Asian J. Res. Comput. Sci.*, pp. 1–10, Oct. 2019.
[7] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–29, Feb. 2019.
[8] Y. Liu, K. Akram Hassan, M. Karlsson, Z. Pang, and S. Gong, "A data-centric Internet of Things framework based on azure cloud," *IEEE Access*, vol. 7, pp. 53839–53858, 2019.
[9] A. Celesti, A. Galletta, L. Carnevale, M. Fazio, A. Lay-Ekuakille, and M. Villari, "An IoT cloud system for traffic monitoring and vehicular accidents prevention based on mobile sensor data processing," *IEEE Sensors J.*, vol. 18, no. 12, pp. 4795–4802, Jun. 2018.
[10] Z. Daher and H. Hajjdiab, "Cloud storage comparative analysis Amazon simple storage vs. microsoft azure blob storage," *Int. J. Mach. Learn. Comput.*, vol. 8, no. 1, pp. 85–89, Feb. 2018.
[11] V. M. Ionescu and J. M. Lopez-Guede, "Comparing Google Cloud and Microsoft Azure platforms for undergraduate laboratory use," in *Proc. Int. Workshop Soft Comput. Models Ind. Environ. Appl. (SOCO)*, San Sebastián, Spain, Oct. 2016, pp. 795–802.
[12] J. Kaur and M. Sharma, "Extending IoTs into the cloud-based platform for examining Amazon Web services," in *Examining Cloud Computing Technologies Through the Internet of Things*. Hershey, PA, USA: IGI Global, 2018, pp. 216–227. [Online]. Available: http://www.igi-global.com
[13] M. Laaziri, K. Benmoussa, S. Khoulji, and M. L. Kerkeb, "A Comparative study of PHP frameworks performance," *Procedia Manuf.*, vol. 32, pp. 864–871, Jan. 2019.
[14] J. M. Volk and M. A. Turner, "PRMS-Python: A Python framework for programmatic PRMS modeling and access to its data structures," *Environ. Model. Softw.*, vol. 114, pp. 152–165, Apr. 2019.
[15] D. Laksono, "Testing spatial data deliverance in SQL and NoSQL database using NodeJS fullstack Web app," in *Proc. 4th Int. Conf. Sci. Technol. (ICST)*, Yogyakarta, Indonesia, Aug. 2018, pp. 1–5.
[16] M. Ohyver, J. V. Moniaga, I. Sungkawa, B. E. Subagyo, and I. A. Chandra, "The comparison firebase realtime database and MySQL database performance using Wilcoxon signed-rank test," *Procedia Comput. Sci.*, vol. 157, pp. 396–405, Jan. 2019.
[17] L. Bienvenu and R. Downey, "On low for speed oracles," *J. Comput. Syst. Sci.*, vol. 108, pp. 49–63, Mar. 2020.
[18] P. Senellart, L. Jachiet, S. Maniu, and Y. Ramusat, "ProvSQL: Provenance and probability management in postgreSQL," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 2034–2037, Aug. 2018.
[19] R. R. Parmar and S. Roy, "MongoDB as an efficient graph database: An application of document oriented NOSQL database," *Data Intensive Comput. Appl. Big Data*, vol. 29, pp. 331–358, Feb. 2018.
[20] M. Ben Brahim, W. Drira, F. Filali, and N. Hamdi, "Spatial data extension for cassandra NoSQL database," *J. Big Data*, vol. 3, no. 1, Dec. 2016.
[21] H. V. Le and A. Takasu, "G-HBase: A high performance geographical database based on HBase," *IEICE Trans. Inf. Syst.*, vol. E101.D, no. 4, pp. 1053–1065, 2018.
[22] C. Asiminidis, G. Kokkonis, and S. Kontogiannis, "Managing IoT data using relational schema and JSON fields, a comparative study," *IOSR J. Comput. Eng.*, vol. 20, no. 6, pp. 46–52, 2019.
[23] V. Jain, "MongoDB and NoSQL databases," *Int. J. Comput. Appl.*, vol. 167, no. 10, pp. 8887–8975, 2017.
[24] J. Fjällid, "A comparative study of databases for storing sensor data," M.S. thesis, Dept. Comp. Sci., Tech. Univ., Stockholm, Sweden, 2019.
[25] 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 J.*, vol. 16, no. 2, pp. 485–497, Jan. 2016.
[26] B. Maity, S. Sen, and N. C. Debnath, "Retracted: Challenges of implementing data warehouse in MongoDB environment," *J. Fundam. Appl. Sci.*, vol. 10, no. 4S, pp. 222–228, 2018.
[27] C. Gyorödi, R. Gyorödi, and R. Sotoc, "A comparative study of relational and non-relational database models in a Web-based application," *Int. J. Adv. Comput. Sci. Appl.*, vol. 6, no. 11, pp. 78–83, 2015.
[28] L. Kumar, S. Rajawat, and K. Joshi, "Comparative analysis of NoSQL (MongoDB) with MySQL database," *Int. J. Modern Trends Eng. Res.*, vol. 2, no. 5, pp. 120–127, May 2015.
[29] Z. Bicevska and I. Oditis, "Towards NoSQL-based data warehouse solutions," *Procedia Comput. Sci.*, vol. 104, pp. 104–111, Jan. 2017.
[30] C. Li and J. Gu, "An integration approach of hybrid databases based on SQL in cloud computing environment," *Softw., Pract. Exper.*, vol. 49, no. 3, pp. 401–422, Mar. 2019.
[31] Y. Rasheed, M. Qutqut, and F. Almasalha, "Overview of the current status of NoSQL database," *Int. J. Comput. Sci. Netw. Secur.*, vol. 19, no. 4, pp. 47–53, Apr. 2019.
[32] C. Asiminidis, G. Kokkonis, and S. Kontogiannis, "Database systems performance evaluation for IoT applications," *Int. J. Database Manage. Syst.*, vol. 10, no. 06, pp. 1–14, Dec. 2018.
[33] *T3. US*. Accessed: Jun. 25, 2019. [Online]. Available: https://aws.amazon.com/ec2/instance-types/t3
[34] Romania. *Pollution Measurements for the City of Brasov in Romania*. Accessed: Jun. 25, 2019. [Online]. Available: http://iot.ee.surrey.ac.uk:8080/datasets.html#weather
[35] J. Moon, S. Kum, and S. Lee, "A heterogeneous IoT data analysis framework with collaboration of edge-cloud computing: Focusing on indoor PM10 and PM2.5 status prediction," *Sensors*, vol. 19, no. 14, p. 3038, Jul. 2019.
[36] D. J. Hand, "Statistical challenges of administrative and transaction data," *J. Roy. Stat. Soc., A (Statist. Soc.)*, vol. 181, no. 3, pp. 555–605, Jun. 2018.
[37] B. Shyti and D. Valera, "The regression model for the statistical analysis of albanian economy," *Int. J. Math. Trends Technol.*, vol. 62, no. 2, pp. 90–96, Oct. 2018.
[38] M. El Genidy, "Multiple nonlinear regression of the Markovian arrival process for estimating the daily global solar radiation," *Commun. Statist.-Theory Methods*, vol. 48, no. 22, pp. 5427–5444, Oct. 2018.
[39] M. M. El Genidy, "Multiple non linear regression model for the maximum number of migratory bird types during migration years," *Commun. Statist.-Theory Methods*, vol. 46, no. 16, pp. 7969–7975, Aug. 2017.

MAHMOUD EYADA received the B.Sc. degree in computer and mathematical science from the Faculty of Science, Port Said University, in 2016. He is a Server Side Back-END Developer, a Database Analyzer, and an IoT Systems Designer work as a freelancer with worldwide projects since his last year in college. His research interests include database management systems, data analysis, the Internet of Things, Internet protocols, wireless sensor networks, and embedded systems.

WALAA SABER received the B.Sc. and M.Sc. degrees in computer and control engineering from Suez Canal University in 2001 and 2008, respectively, and the Ph.D. degree in computer and control engineering from Port Said University, Egypt, in 2014. She is an Assistant Professor with Electrical Engineering Department, Port Said University. Her research interests include computer networking, including cloud computing, clustering, and the Internet of Things.

MOHAMMED M. EL GENIDY received the Ph.D. degree in statistics and computer science from Mathematics Department, Faculty of Science, Mansoura University, Mansoura, Egypt, in 2001. He is currently an Assistant Professor of statistics with the Faculty of Science, Mathematics and Computer Science Department, Port Said University, Port Said, Egypt. His research interests include statistics, probability, order statistics, queues theory, regression, estimation, distributions, mathematical programming, statistical tests, and hypotheses test.

FATHY AMER received the B.Sc. degree in military science and the B.Sc. degree in electrical and communication engineering from the Military Technical College (MTC), Cairo, Egypt, in 1970, the M.Sc. degree in electrical, mechatronics, and communication engineering (major area: electronics and communication) from Azhar University, Cairo, in 1985, and the Ph.D. degree in philosophy in computer science from Computer Science Department, Azhar University. From May 1970 to June 1993, he was an Engineer Officer and a Lecturer with the Military Technical College and various places in the armed forces in the field of computers, information technology, electrical engineering, communications, and electronic insurance. From September 2012 to August 2013, he was the Dean of the Higher Institute of Engineering and Technology, Obour, Egypt. From September 2013 to September 2016, he was a Professor Emeritus with the Department of Information Technology, Faculty of Computers and Information, Cairo University. Since October 2016, he has been the Vice Dean of Community Affairs and the Environment, Misr University for Science and Technology. He is currently a Professor with the 6th of October University. His research interests include local, expanding and international networking, databases, multimedia, the Internet of Things, and information technology.