A Framework for Executing Complex Querying for Relational and NoSQL Databases (CQNS)

Eman A. Khashan, Ali I. El-Desouky, and Sally M. Elghamrawy

***Abstract* — The increasing of data on the web poses major confrontations. The amount of stored data and query data sources have become needful features for huge data systems. There are a large number of platforms used to handle the NoSQL database model such as: Spark, H_2O and Hadoop HDFS / MapReduce, which are suitable for controlling and managing the amount of big data. Developers of different applications impose data stores on difficult tasks by interacting with mixed data models through different APIs and queries. In this paper, a complex SQL Query and NoSQL (CQNS) framework that acts as an interpreter sends complex queries received from any data store to its corresponding executable engine called CQNS. The proposed framework supports application queries and database transformation at the same time, which in turn speeds up the process. Moreover, CQNS handles many NoSQL databases like MongoDB and Cassandra. This paper provides a spark framework that can handle SQL and NoSQL databases. This work also examines the importance of MongoDB block sharding and composition. Cassandra database deals with two types of sections vertex and edge Portioning. The four scenarios criteria datasets are used to evaluate the proposed CQNS to query the various NOSQL databases in terms of optimization performance and timing of query execution. The results show that among the comparative system, CQNS achieves optimum latency and productivity in less time.**

***Index Terms* — NoSQL, Query processing, Querying, Hadoop HDFS, Spark Mongo, Spark connector, parallel k-means, clustering, query optimization, Cassandra, partitioning.**

I. Introduction

The popularity of NoSQL systems is caused by their efficiency in handling unstructured data and backing up effective design schemes that give the system users supreme flexibility and scalability. This paper identifies a relational database and several categories of NoSQL databases with structural features: key-value, graph, column and document databases. Likewise, every NoSQL database has a special query language and does not support the criteria of other systems. The main problem that many researches focused on, is that there is no standard way for expressing, executing and optimizing complex queries across NoSQL databases [1], [4]. Currently, data stores have several diversified APIs. The programmers of applications based on multiple data stores must be familiar with these APIs during the process of coding these applications. As a result of the variety and changes in the data models of various databases, there is no standard way to solve the problem of implementing queries for various NoSQL data stores. The reason is because of a lack of a combined access model for diversified data stores. The programmers must challenge themselves with the execution of these queries, which are hard to optimize. On the other hand, optimization puts certain criteria into consideration, such as data transformation and movement costs, which might be expensive for big data. Figure 1 shows a diagram of integrating heterogenous relational and NoSQL datasets to an example of scientific social network.

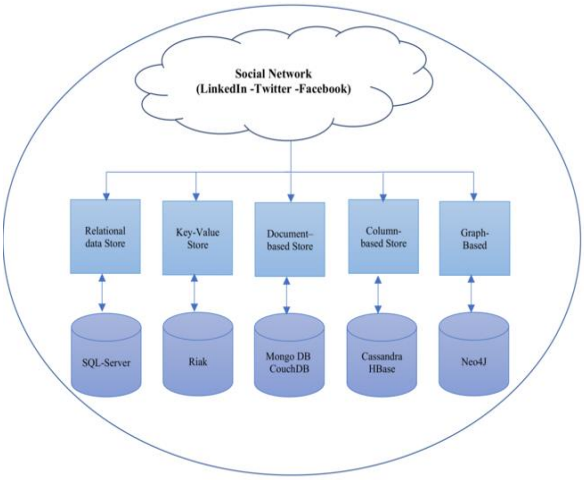

Fig. 1. Connecting heterogenous data sets to social network.

All of these reasons encourage sharing in the interoperability between two or more varied and powerful frameworks. In this paper, Mongo and Cassandra [37] focus on being the most popular way to help companies make business decisions. Several researchers and developers have focused on this problem. The variety of relational and NoSQL data models (relational, key value, ordered key-value, document, semi-structured and graph databases) and query languages (SQL, Cassandra Query Language (CQL), MapReduce querying language, etc.) is the main difficulty. Salami et al. [1] Identify a common data model and use algebra to address complex declarative inquiries. In this technique, queries are handled in multiple data stores called VDS, that is, default data stores. The optimization stage is carried out by a two-step broker. First, the selection and project processes are defined down to the local data stores. This allows to reduce the amount of data exchange. Second, an ideal distributed plan is designed with a dynamic programming method. The distributed plan seeks to reduce I / O and CPU costs and to charge and convert data. However, this technique is limited to redressing an ODBAPI query and some query operators. Another method was developed by

Published on September 22, 2020.
Eman A. Khashan, Mansoura University, Egypt.
Ali I. El-Desouky, Mansoura University, Egypt.
Sally M. Elghamrawy, MISR Higher Institute for Engineering & Technology, Egypt.
(corresponding e-mail: sally_elghamrawy@ieee.org)

P.Sangat et al. [2] called DIMS. In DIMS Most data generated by ubiquitous sensing applications have the character of time series, such as monitoring data of power station, and from others a pattern of interrelationship emerges, for instance the correlation between patients, disease, and symptoms. Further, high sampling frequency and high data generation rate also feature. To satisfy the needs of various requirements, a data storage system should have various abilities, such as making different schemes and profiles for different applications. Song et al [15], they present the design, implementation and evaluation of Haery, a column dedicated to big data. Haery is built on Hadoop HDFS and distributed computing framework relying on MapReduce. Haery's download and query performance results are the most stable and effective. But there is more cost in time when data volume increases. Haery proposed the following models and algorithms: Key-Cube, an improved Z-order based linearization algorithm and an address tree, Accumulation, which is a key-cube expansion approach, Query algorithms to implement queries on key-cubes and physical storage and the system architecture, components and implementation of Haery. The rest of this paper is organized as follows. In section 2, this paper presents the related work. In section 3, the proposed CQNS framework, which has three layers. In section 4, research work discusses implementation and evaluation of CQNS. Section 5 provides conclusions and future work.

II. Related Work

Several researchers and developers have focused on Querying NoSQL Databases problem [4], [5], [20]. The variety of relational and NoSQL data models (relational, key value, ordered key-value, document, semi-structured and graph databases) and query languages (SQL, Cassandra Query Language (CQL), MapReduce querying language, etc.) is the main difficulty. G. Baruffa et al. [3] characterized a Spectrum Sensing that provides a service to allow end users to easily access and process wireless spectrum data. To reduce the latency of services provided by the platform, that adjust the data processing chain. They took an interest in Mongo and Cassandra databases and did not consider the rest of the databases. Khan et al. [4]. and Duggan et al. [5] it offers frameworks that called PolyWeb and BigDAWG, respectively. PolyWeb and BigDAWG retain data sources in a primary format, that is, without serializing them in a common data format. In PolyWeb, SPARQL queries are translated into the original query language for these sources. PolyWeb indexes each data source to predict the query and creates deep left plans. Despite the efficiency, the current methods are not able to exploit knowledge about the main features of integrated data sources, and produce custom query plans for selected sources to collect data from the data lake. In contrast, the QODM [6] approach produces distinct schema using the data model and data schema of an application for NoSQL databases. This approach will not prevent programmers from using any NoSQL database. document and relational data stores are integrated in a hybrid mediation approach proposed by Roijackers et al. [7]. However, this approach does not consider other NoSQL data stores. Sharma et al. [8] In this research paper studying the performance of RDBMS, Document based No SQL data base (MongoDB) and Graph based No SQL Data base (Neo4j). they got unexpected results in case of neo4j as it took longest time as compared to MongoDB and PostGre SQL. However, few NoSQL databases are supported only by these frameworks, so the programmer has to make designs for data models of an application and choose a proper strategy for data mapping. CQNS is a proposed framework for improving and estimating complex queries for relational databases and other types of NoSQL data stores. For this purpose, a unified data model is proposed that uses a suitable environment such as Apache Spark with MongoDB [18], [24] to optimize the qualification of the data ingestion process. The CQNS framework transforms each query process received from any dataset to the matched Engine after using Hadoop/HDFS and Hadoop/MapReduce with parallel k-means clustering for processing data without physical transformation data.

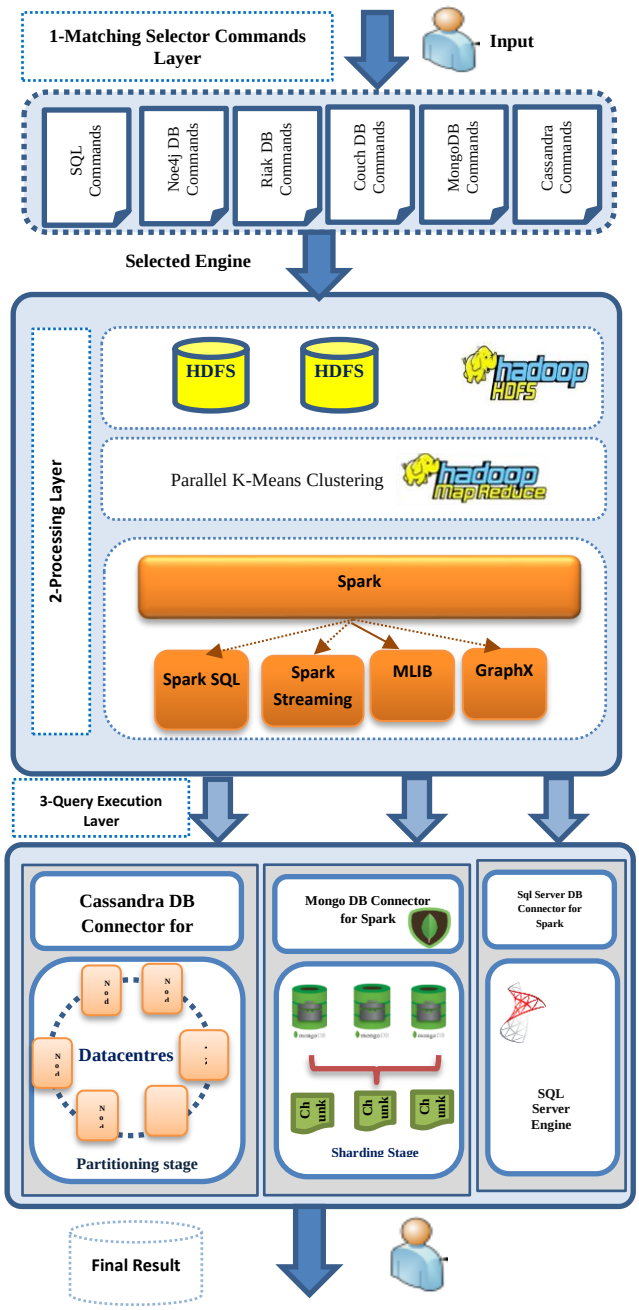

Fig. 2. CQNS Framework.

III. PROPOSED CQNS FRAMEWORK

This section introduces the proposed CQNS approach, which is capable of executing complex queries across heterogeneous data stores. This framework consists of three layers, as shown in Fig. 2: Matching Selector layer, processing layer, and query execution layer. In the following sections, this paper discusses the different layers of the proposed CQNS framework.

A. Matching Selector Layer

This layer receives any SQL or NoSQL database query to match the sentences of the query given by the user with the stored libraries that hold a number of statements for each database type either SQL or NoSQL from the database engine and then compares the sentence with the stored libraries to define the required database engine. This paper prepared a set of libraries for each of the databases that are studied, such as SQL as example of relational database and MongoDB, Cassandra as an example of NoSQL databases. Indeed, this approach symbolizes the combined parts among every deployed data storage and delivers a unified model to the following layer of the framework. This model contains the particular operations of every database. It is noteworthy that the user has to add a particular implementation of the data store if he/she needs to integrate an extra database. In the following figures, an explanation is given for testing the query statements for the databases used. This paper used SQL Server (as an example of a relational database), MongoDB and Cassandra (as examples of NoSQL databases). Fig. 3 a explains the stored SQL libraries statements for SQL database while Fig. 3 b and 3c explain the CRUD statements for MongoDB and Cassandra DB, respectively, as examples of the NOSQL Database libraries used in this paper.

Select Operation
Select * from Customers;
Select * from customers where id=3
Insert Operation
Insert into customers (id, Name, position, phone) values(1,'John','Egypt','0100000')
Update Operation
Update customers set position='USA' where id=25
Delete Operation
Delete * from customers where id=25

Fig. 3 a. SQL libraries.

Select Operation
- db. customer. Find ();

Insert Operation
- db. customer. Insert ({cust_id: 'appl01', branch: 'main', status: 'A'})

Update Operation
- db. customer. Update ({custage: {$gt: 2}}, {$set: {branch: 'main'}}, {multi: true})

Delete Operation
- db. CustomerCollection.deletemany();
- db. CustomerCollection.remove();

Fig. 3 b. Mongo libraries.

Select Operation
- SELECT * FROM customer;

Insert Operation
- INSERT INTO customer (custid, branch, status) VALUES ('appl01', 'main', 'A');

Update Operation
- UPDATE customer SET comments ='='Rides hard, gets along with others, a real winner' WHERE id = fb372533 -eb95-4bb4-8685-6ef61e994caa IF EXISTS;

Delete Operation
- DELETE lastname FROM Customer WHERE id = 'c7fc eba0-c141-4207-9494-a29f98de6f';
- DELETE FROM DB. Customer WHERE id= 2;

Fig. 3 c. Cassandra libraries.

Select Operation
riak-shell>select Name, Position from Customers where id > 1234560 and region = 'South Atlantic' and state = 'South Carolina'
Insert Operation
riak-shell>INSERT INTO customers VALUES ('SC', '2018-01-01T15:00:00', 'sunny', 43.2, 0x3af6240c1000035dbc), ('SC', '2017-01-01T16:00:00', 'cloudy', 41.5, 0x3af557bc4000042dbc), ('SC', '2017-01-01T17:00:00', 'windy', 33.0, 0x3af002ee10000a2dbc);

Fig. 3 d. Riak libraries.

Select Operation
"selector": {
"year": {**"$gt"**: 2010} },
"fields": ["_id", "_rev", "year", "title"],**"sort"**: [{**"year"**: "asc"}], **"limit"**: 2, **"skip"**: 0, **"execution_stats"**: **true**
Insert Operation
INSERT INTO `travel-sample` (KEY, VALUE)VALUES ("key1", { "type" : "hotel", "name" : "new hotel" }) RETURNING *
Update Operation
curl -X PUT http://127.0.0.1:5984/database_name/document_id/ -d '{ "field" : "value", "_rev" : "revision id" }'
Delete Operation
$ curl -X DELETE http://127.0.0.1:5984/my_database/001?rev=1-3fcc78daac7a90803f0a5e383
{"ok":true,"id":"001","rev":"2a561d56de1ce3305d693bd156"}

Fig. 3 e. Couch libraries.

Algorithm 1 illustrates the method of discovering the database type of the query to be executed based on the libraries stored in the application to show the selected engine database.

According to Algorithm 1, the results of matching patterns and input values, one of the following decisions will be followed:

If the patterns are identical to the SQL database, the application will continue to run the path of the SQL database. If the patterns are identical to the Mongo database, the application will continue to run the path of the Mongo database. If the patterns are appropriate for the Cassandra database, the path for the Cassandra database will be followed.

If you want to apply patterns to other databases, you must add their own libraries.

```
Input: qs query Statement
Output: SQL or NoSQL database Engine
1. parsing Query Statement (qs).
2. Declare Arr[6]={ sql, Mongo, Cassandra, riak, Neo4j, Couch}
3. For I =0 to 5
   {
   if(arr[i] == qs) selected_Engine=arr[i]
   break
   }
4. Switch selected_Engine
   Case sql
        Connect to sql Engine
   Case Mongo
        Connect to MongoDB Engine
   Case Cassandra
        Connect to CassandraDb Engine
   Case riak
        Connect to Riak Engine
   Case Neo4j
        Connect to Neo4j Engine
   Case Couch
        Connect to Couch Engine
   Case else
        No Engine
   End switch
5. If selected_Engine.='Noengine', then display error page and stop running,
   else continue running & execute query sa.
```

Algorithm 1. Matching Selector algorithm.

B. CQNS Processing Layer

Information on the technologies used and the setup environment for this experiment is provided briefly in this section. CQNS deployed and used Hadoop/HDFS [1] to store the incoming data. Hadoop is an open source distributed computing platform that mainly consists of the distributed computing framework MapReduce and the distributed document system HDFS [3]. MapReduce [9], [22] is a software platform for parallel processing programming of large-scale data pieces. The MapReduce strategy is applied to the k-means clustering algorithm and clustered for the data factors. The k-means [19] algorithm can be successfully parallelized and clustered on hardware resources. MapReduce can be utilized for k-means clustering. The results also show that the clusters shaped using MapReduce are similar to the clusters produced using a sequential algorithm. Once HDFS takes data, this process breaks information down into separate blocks and distributes those blocks to different nodes in the cluster, thus enabling high-efficiency parallel processing. The data from HDFS is accessed by a Spark streaming program for handling before being stored in MongoDB in the server of the database. Resilient distributed datasets (RDDs) are an abstraction presented by Spark [13]. RDDs symbolize a read-only multiset of data objects divided into a group of machines that continue operating as designed despite internal or external changes (fault-tolerant way). Spark is considered the first system of programming languages in general and is used as an interactive way to handle big data [36] sets for clustering. A Complex Querying over NoSQL databases Algorithm (CQNSA) using MongoDB and the MongoDB Connector for Spark is proposed using an open source NoSQL database that is designed for high scalability, effectiveness, and availability. This CQNSA is shown in Algorithm 2.

```
1:  input qu: A query
2:  output schedule: The optimal execution schedule of the query qu
3:  selectAttributes;  # Parsing the SELECT clause
4:while (exist (attribute Att in the clause SELECT)) do
5:  select_Attributes:add(entity_Set_Of(Att); Att)
6:end while
7:  initNodes;  # Parsing the FROM clause
   Join_Conditions; # Parsing the WHERE clause to identify distributed joins
8: while (exist (condition Con in the clause WHERE)) do
9:  entity_Set_Left site_Of_Left_Entity_Set(Con)
10:  _site _siteOf(_entity_Set_Left)
11:  if _is_Join_Condtion(C) then
12:  if _site == _site_Of_(Entity_Set_Right(Con)) and _can_Join(site) then
13:  _init_Nodes:_merge(_entity_Set_Left; _EntitySetRight(Con))
14:  else
15:  _Join__Conditions:_add(Con)
16:  end if
17:  else
18:  _init_Nodes:_add__Restriction__Condition(entity_Set_Left; Con) end if
19:  end while
20:  schedule _init_Nodes
21: for i = 1 to _number of _join__conditions do
22:  for each _condition Con in _Join__Conditions do
23:  for each schedule s1 in schedules do
24:  for each schedule s2 in schedules do
25:  if _match(s1; s2; Con) then
26:  _new_Conditions:_add(_conditions_Of(s1); _conditions_Of(s2); Con)
                                # Adding_a_new_join_condition
27:  _node_createVDS_Join(s1; s2; Con)
28:  schedules:add(node)# Creating a new schedule with a VDS join node
29:  end if
30:  end for
31:  end for
32:  end for
33: end for
```

Algorithm 2. CQNSA algorithm.

C. Query Execution Layer

Instead of storing the data as tables with columns and rows, the data are stored as documents. Every document can be one of the relational matrices of the numerical values or the overlapping interrelated arrays or matrices. These documents are serialized as JSON objects and stored internally using JSON binary encryption known as BSON in MongoDB; the data is partitioned and stored on several servers called shard servers for simultaneous access and effective read/write operations. MongoDB and Apache Spark are integrated seamlessly by this connector. MongoDB aggregation pipelines and a problem of how to assign a group of objects into groups, called blocks, so that the objects within the same group, partitioning is by using a cluster assignment function C:X→{1,2,….,k} when X is a set of objects, the Number of clusters $K \in \mathbb{Z}^+$and Distance function $d \in \mathcal{R}0^+$ between all pairs of objects in X, partition X into K disjoint sets x1,x2,…..,x_k such that $\sum_k \sum_{x,x' \in X_k} d(x,x')$ with N = |X|, the number of distinct cluster assignments possible as follows [33]:

$$S(N,K) = \frac{1}{K!}\sum_{k=1}^{K} -1^{K-k} \binom{K}{k} k^N \qquad (1)$$

D. MongoDB Engine

Sharding is a way to distribute data across multiple devices. This paper presents MongoDB, which uses sharding

to support deployments using very large datasets and high-productivity processes. Database systems that contain large datasets or high-productivity applications can challenge the capacity of a single server. For example, high query rates can exhaust the CPU capacity for the server. A range of sizes greater than the system's RAM can help to confirm the I/O capacity of the drivers. A database can have a mixture of sharded and unsharded collections. Sharded collections are partitioned and distributed across the shards in a cluster. Unsharded collections are stored on a primary shard. Each database has its own primary shard as shown in Fig. 4.

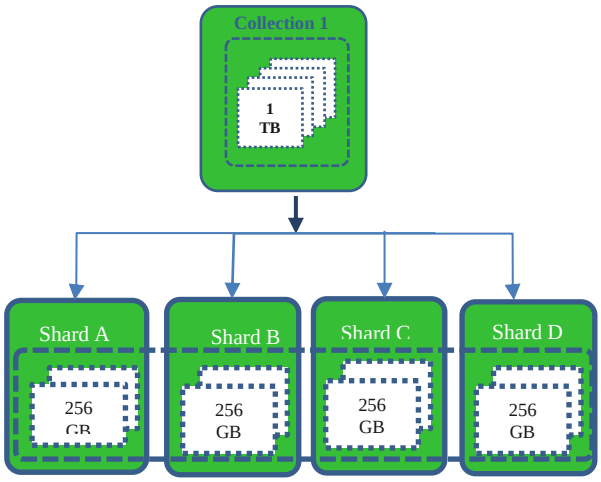

Fig. 4. Sharding Mongo DB Stage.

In addition to JSON schema validation, MongoDB manages validation with query filter expressions using query operations, with the exceptions of $near, $nearSphere, $text, and $where. Figure 5 explains a JSON example of specifying validator rules using the following query expression:

```
db.createCollection( "Personscontacts",
  { validator: { $or:
    [       { P_phone: { $type: "string" } },
      { P_email: { $regex: /@mongodb\.com$/ } },
      { P_status: { $in: [ "Unknown", "Incomplete" ] } }
    ]
  }} )
```

Fig. 5. JSON example for specifying validator rules.

E. Cassandra Engine

The Apache Cassandra database has linear scalability and proven tolerance for hardware or cloud infrastructure, and these attributes make this database an ideal platform for important data. This paper presents replication supported by the Cassandra database across multiple data centers that is best in class, providing less downtime for users and peace of mind by knowing that it can overcome regional interruptions. This paper proposes two kinds of partitioning methods that can work with the Cassandra database: vertex partitioning and edge partitioning. Later, this study will introduce how can research paper dealing with these methods. This paper investigates vertex partitioning and edge partitioning to show differences in the results about them. This paper investigates vertex partitioning and edge partitioning to show differences in the results about them.

F. Data Partitioning

Cassandra divides the database into smaller, partially overlapping datasets that are stored locally on each node. Thus, unlike other NoSQL databases such as HBase, Cassandra does not require a shared file system (for example, HDFS). A hash function is used to distribute basic registry keys for the nodes. This process is performed by dividing the scope of the hash key into subdomains called partitions (also called token ranges). In blocks without repeating (RF = 1), each node can be configured to store unique partitions locally. In this section, the necessary background will be provided and presented with the data and account models we target. Table 1 contains the partitions variables used in this paper. This paper uses formula (2) to calculate the size of data partitions.[35]

$$NV = Nr(Nc - NpK - Ns) + Ns \quad (2)$$

TABLE 1: THE PARTITIONS VARIABLES USED IN THIS PAPER

Symbol	Description
Nv	The number of values (or cells) in the partition
Nr	The number of values per row.
Nc	the number of columns.
NpK	the number of primary key columns.
Ns	The static columns.

IV. CQNS EVALUATION

CQNS is utilized to store, manage and execute queries of big data and greatly facilitates the developer's task. In this paper, the proposed model rewrites each query into the particular query language of the integration data store. The processing layer in CQNS turns results into a suitable format such as JSON before responding to the system users. Therefore, the overhead is considered reasonable to some extent. Because of memory management trouble in the driver, there is a probability that the performance of CQNS will degrade after 50000 entities. The results of experiments testing MongoDB and Cassandra DB are shown in the following sections.

A. Datasets

To assess the suggested framework strategy for querying NoSQL databases, the research work performed experimental tests using four scenario benchmark datasets for hybrid datasets. A description of the dataset used in this paper can be downloaded from http://snap.stanford.edu/data/. All of the datasets are related to complex querying. The datasets used in our research have a large number of features (thousands of rows) that are suitable for displaying the efficiency of our strategy for high-dimensional data. The ego-Facebook, ego-Twitter and soc-LiveJournal1 datasets were acquired from different types of database engines.

B. System analysis details

This section provides brief information on the techniques used and preparing the environment for the experiment. This study creates and publishes Hadoop, HDFS, Apache Spark, MongoDB, MongoDB link for Spark, Cassandra DB connector, Cassandra DB for spark server and SQL on the environment with the following specifications.

1. Hadoop / HDFS Setup: This paper deploys Hadoop /

HDFS version 2.10.0 using standard configuration parameters. The application server is running on HDFS to store incoming data and access software Spark Access data from HDFS for processing before inclusion in SQL, MongoDB and Cassandra DB in the database server.

2. Spark Setting: Apache Spark is a widely open source framework. Spark introduces a stripping called elastic Distributed Data Sets (RDDs), which represent a multiple read-only set of data items divided across a set of devices that are maintained in Error tolerant method. Spark is the first system to interactively use a general-purpose programming language to collaboratively Large data sets on a block. Apache Spark 2.3.2 version is published in standalone mode and control configs of application code. For example, the experiments have set spark.serializer as org.apache.spark.serializer.KryoSerializer. The application was created Using Scala 2.10.4.

3. Database Connector for Spark: This connector provides a seamless integration between matched Database and Apache Spark. It effectively uses database assembly lines and secondary indexes to extract, filter and process the sub-data required for the Spark process. Additionally, to maximize performance over a large distribute Datasets, they link the RDDs to the source database node and reduce the data transfer across the cluster.4. Hardware: The Spark app server has 16 dedicated hubs, 64GB of memory, 459GB of hard drives, and 64-bit Ubuntu GNU / Linux. The mongo dB database server and both Shard 4 server have dedicated cores, 16GB memory and 130GB HDD, while each of the initialization servers contains 1 hard disk, 4GB and 30GB. The Cassandra database server. 16 GB memory, 130 GB hard drives, and Microsoft SQL 2017 server has been installed with the same infrastructure specifications previously mentioned.

C. Cost Model

The cost of implementation is the sum of the costs of each process that composes the implementation plan. It should be noted that the cost does not directly represent time. Of course, more cost means more time. It is used to compare two query execution plans, but not to directly estimate response time. To evaluate the cost formula, the matrix multiplication between the row vector containing the coefficients α, β, and γ was calculated. A column vector contains the values of the parameters defined in the catalog, and a fixed variable called const which is a scalar and can be a cardinality, selectivity, etc. In addition, if the parameter does not depend on a specific measurement (CPU cost, I/O cost, or cost of connections), this will take the latter an empty value in the column vector. Matrix multiplication [1] is calculated as follows:

$$(\boldsymbol{const}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) \begin{pmatrix} t_{cpu} \\ t_{i/o} \\ \boldsymbol{t_{cconn}} \end{pmatrix} = \boldsymbol{const}\left(\boldsymbol{\alpha} \times \boldsymbol{t_{cpu+}} \boldsymbol{\beta} \times \boldsymbol{t_{i/o+}} \boldsymbol{\gamma} \times \boldsymbol{t_{cconn}}\right) \quad (3)$$

Performance estimation is an important point for a new framework. This estimation is shown in the outcomes of total cost, average time, and ingestion rate. Table 2 displays sample catalogue variables. These outcomes are utilized to estimate the efficiency of the proposed framework. The outcomes are calculated by the following equations:

$$TotalCost = \alpha \times t_{cpu+} \beta \times t_{I/o+} \gamma \times t_{cconn} \quad (4)$$

$$Average\ time = \frac{Total\ cost}{No.of\ joins} \quad (5)$$

$$Ingestion\ rate = \frac{No.of\ records}{Average\ time} \quad (6)$$

D. CQNS Join Queries

The purpose of these experiments is to estimate the impact and validate the proposed optimization process. Thus, various designs utilized to optimize the process are tested to ensure their efficiency and integration. The purpose of this paper is to contrast various strategies to measure both the response time and the latency time precisely and accurately. This purpose is possible due to experiments that are performed in this proposal using two variables of join queries. For variable No. 1, every entity set might join not less than one entity set and not more than two entity sets, creating a linear form, which is generally utilized in several applications. For the second variable, all entity sets executed in a query join a similar entity set, creating a star form, which is widely used in data warehouse applications. The two variables of queries are implemented through four different possibilities. It is necessary to know whether the sub-queries in these possibilities are implemented in a parallel or sequential way and whether external join implementations are enabled or disabled.

• Possibility 1: This possibility will occur at the time of running a join query sequentially while just utilizing the VDS and integrated data store for push-down operations (most simple possibility).

• Possibility 2: This possibility will occur at the time of executing a join query sequentially utilizing both the VDS and integrated data store for exterior joins and the push-down process.

• Possibility 3: This possibility will occur at the time of executing a join query in parallel while just utilizing the VDS. In this case, to implement push-down operations, parallelism is performed in the integrated data store.

• Possibility 4: This possibility will occur at the time of operating a join query in parallel while just utilizing the VDS and integrated data store. The implementation of the external joins and the push-down process will be parallelized. Furthermore, the experiments are performed with two datasets that are different in size. The first of these datasets is medium-sized, while the second is large-sized. Note that these various sizes are multiplied by hundreds of factors. There are ten entity sets that are different in size, starting from some megabytes and ascending to hundreds of megabytes, used in the experiments and saved on five various types of data stores. The experiments are conducted by proposing to utilize four querying processes, such as linear and star joins, and increasing the join counts for both from three to ten. Moreover, the subsequent constraints on the variables in the catalogue are considered. In fact, the variables in the catalogue are considered. The variables a, b and c shown in Table 2 symbolize CPU timing, input/output timing and connection timing, which are correspondingly assigned to five data stores and the VDS.

TABLE 2: SAMPLE CATALOGUE VARIABLES

Data stores	α, β, γ	convert	ships	scans	load	Init ETSL	Cardinality	Init_Join	Join_No
D1	2	6	6	2	2	20	EntityA=4000000	5	0; 2
D2	2	8	8	2	2	20	EntityB=1000	20	4
D3	2	2	2	2	2	2	EntityC = 3000	2	2
D5	2	6	6	2	2	2	Entity= 10000	6	16
VDS	2	2	2	2	2	8	__	10	2

E. CQNS Linear Join Experiments

According to possibilities 1 to 4 and the number of joins, the overall time estimations from the time model are presented in Figures 7a and 7b. In Figures 8a and 8b, the results are explained for accessing very large-sized data, such as Big Data-Context. Every cardinality is multiplied by 100. Based on these results, the main aspects of this proposition are proved for creating an optimal implementation plan without dependence on the variable of the join query. The following conclusions have been obtained on the basis of the results shown in Figures 7c, 7d, 8c and 8d. In fact, the assumption in this paper is that only data stores D1, D2 and D3 support the join query implementation. Subsequently, the integrated data store is supplied to be able to save one or more multiple entity sets, e.g., in storage area D2. It is remarkable that the cardinality demonstrated in the chart shows the number of entities in the proposed entity set for a medium-sized dataset. The external joins show the following significance: both possibility 2 and possibility 4 indicate the importance of utilizing the integrated data store in the implemented join queries. In fact, a better aggregate cost is obtained in comparison to that of the possibilities that are executed using the VDS. The gain in possibilities 1 and 2 and that in possibilities 3 and 4 are computed to obtain the average gain, which is equal to 92.23%. The significance of the parallelism is as follows: Parallelized implementation of external joins and push-down operations proved to be advantageous and created a significant gain compared to that of the two other possibilities (in which the implementation is sequential). By comparison with the other possibilities, the average gain is equal to 86.67%.

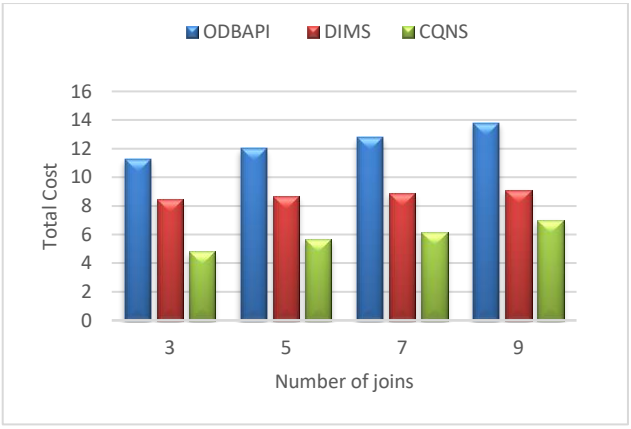

Fig. 7-a. Possibility 1.

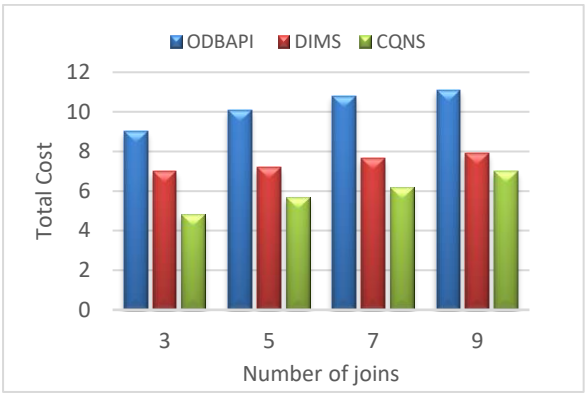

Fig. 7-b. Possibility 2.

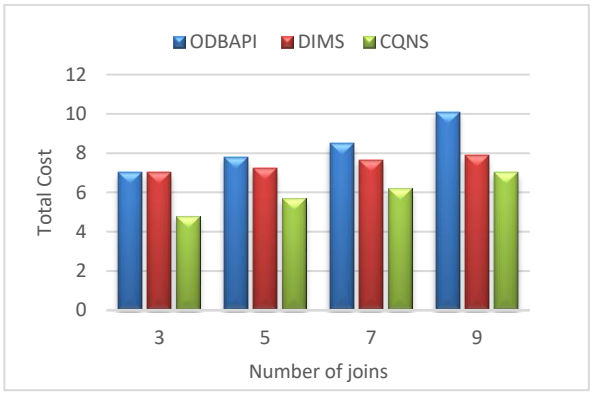

Fig. 7-c.Possibility 3.

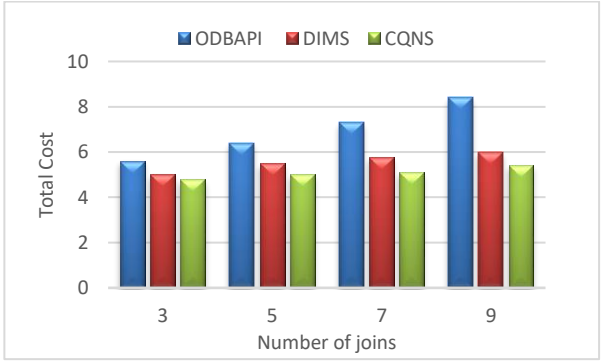

Fig. 7-d. Possibility 4.

Fig. 7. Comparison of Possibilities 1 to 4 for ODAPI, DIMS and CQNS on Linear Joins.

F. CQNS Star Joins Experiments

When possibilities 3 and 4 are applied using star joins, the gain becomes great. In fact, this result is due to the parallel implementation of all compression operations at the same time. The importance of integrating different optimization strategies is as follows: For possibility 4, the work of the integrated data store is maximized, and queries are executed in parallel. The possibility having the best total cost compared to the simple cost is possibility 1. In fact, the average gain was found to be 99.98%. Figures 8-a,8-b,8-c and 8-d explain the difference results between ODBAPI, DIMS and our framework CQNS.

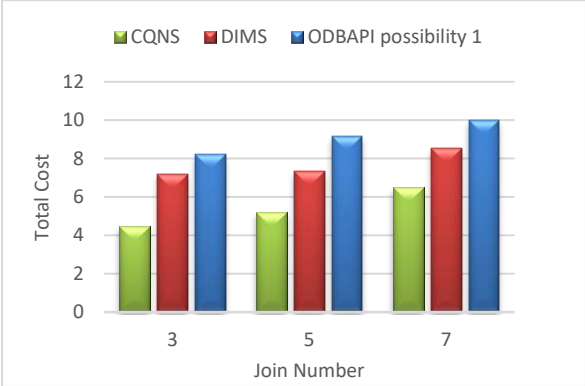

Fig. 8-a. Possibility 1.

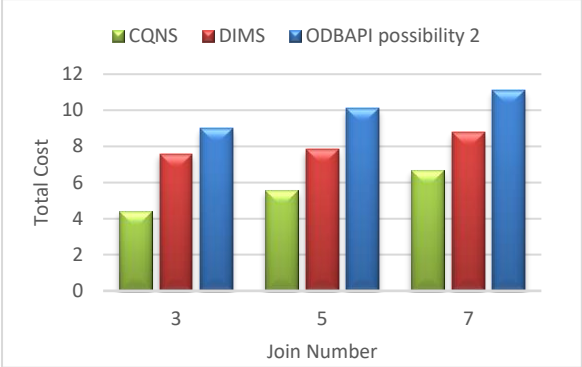

Fig. 8-b. Possibility 2.

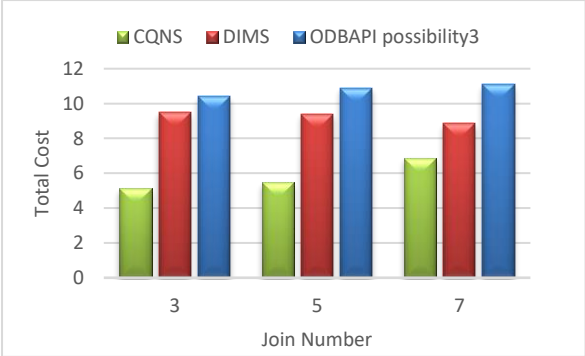

Fig. 8-c. Possibility 3.

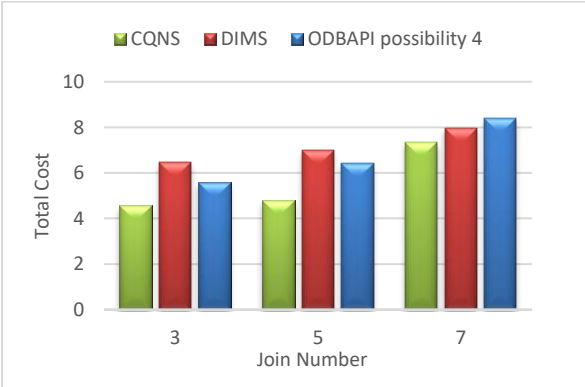

Fig. 8-d. Possibility 4.

Fig. 8. Comparison of Possibilities 1 to 4 for ODAPI, DIMS and CQNS on Star Joins.

G. CQNS Optimization Time in MongoDB

An adaptive schema is proposed to improve the performance and time of query execution, putting the size of the research area into consideration. Experiments that maximize the work of the integrated data store are shown in Fig. 9 and 10 and are very significant, particularly when this approach is utilized to execute very large quantities of data in addition to parallelism. The significance of merging the previous layers to obtain cost optimization is also proved in these experiments. The greater the number of records increased, the more the MongoDB sharding efficiency decreased, compared with that of the No-Sharding databases, as shown in Fig. 9, and this decreased efficiency influenced the execution time. By comparing Spark core and Spark SQL, the experiments found that the former differs from the latter because Spark SQL is loaded first. There is an increase in time, as shown in Fig.10.

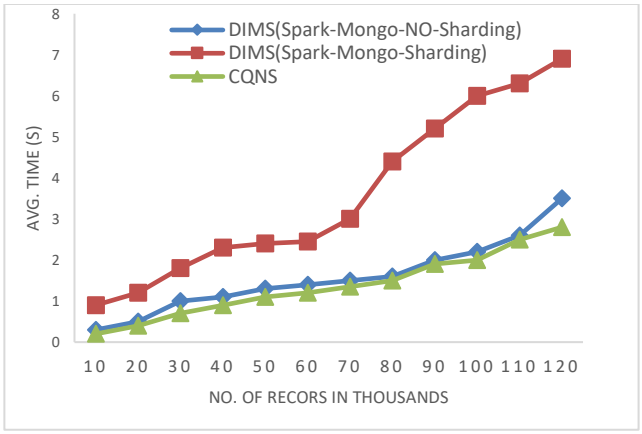

Fig. 9. Comparison of the average time using Mongo import and Apache Spark with Sharding and No-Sharding and CQNS.

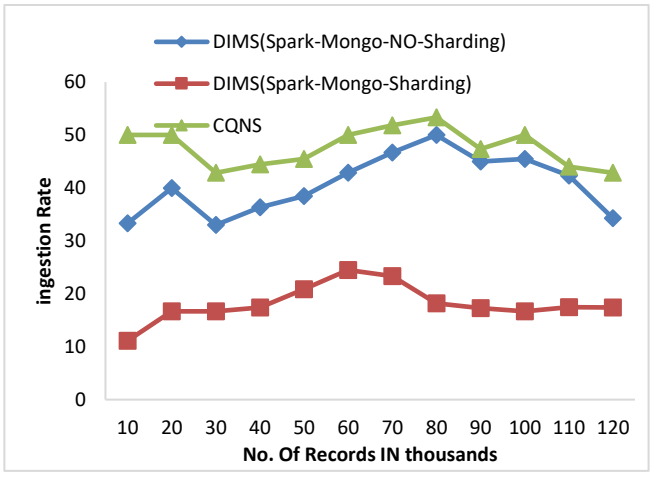

Fig. 10. Comparison of the ingestion rate using DIMS Mongo import, DIMS Apache Spark and CQNS.

H. An Experiments on Cassandra DB

The latency of read and write operations for vertex partitioning and edge partitioning when varying the number of nodes, is compared in the same environment of testing for possibilities 1, 2 and 3. The results for workloads A and C are shown in Fig. 11,12, respectively. From the last two experiments, it is obvious that among the compared systems, the CQNS framework, when dealing with the Cassandra dataset with partitioning, achieves the best latency throughput values compared with those without using partitioning and using partitioning.

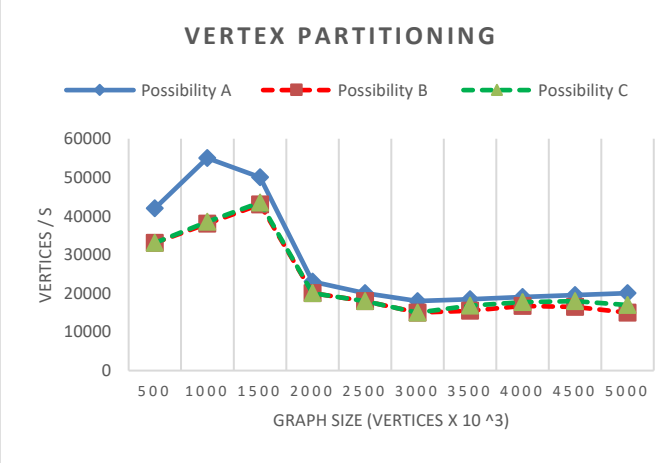

Fig. 11. Latency Time Using Vertex Partitioning.

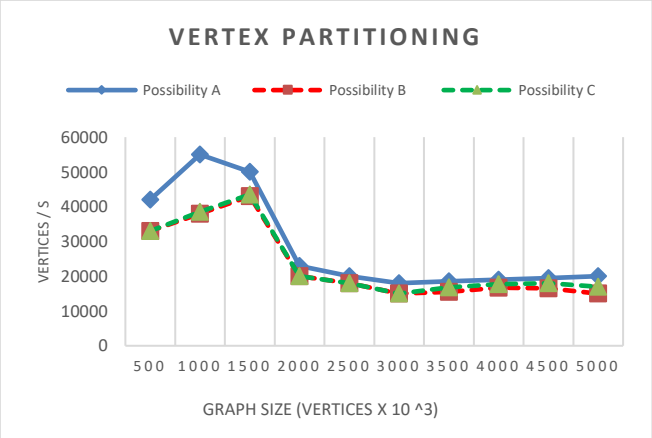

Fig. 12. Latency Time Using Edge Partitioning.

I. Performance and cost over CQNS and Haery framework

Since CQNS writes data in HDFS directly without writing data, Figure 13 shows the average download speed for HBase, Cassandra, MongoDB, CQNS with Cassandra and CQNS with Mongo on data sets of different data sizes. In the download experience, CQNS's upload performance averaged 0.2, 10.2, 1.7, and 20.23 times higher than HBase, Cassandra, MongoDB, and Haery, to continue. the Cassandra with CQNS framework achieved better results than Haery [15], but the results of Haery using Mongo database are relatively better than CQNS results with no sharding.On the other hand, when CQNS applied the sharding technique, the results obtained are better than results obtained from Mongo and Cassandra databases, when using a large size of data.

J. Results Analysis and Discussion

In this section, the core algorithms of CQNS and the results are evaluated and compared with some popular NoSQL and relational databases using generated datasets and various query workload. The experiments conducted provided a comparison between CQNS and ODBPI to measure the cost time based on the number of joins. The results obtained, when using the Hadoop and spark, reflects higher performance compared to recent algorithms. CQNS evaluation implemented two types of joins: linear join and star join. And the results obtained from linear joins proved that they are better than star joins results. In addition, a comparison with the DIMS framework to measure the average time and ingestion time have been implemented using two different databases: Mongo and Cassandra databases. The CQNS results when using Mongo with sharding technique is better than using Mongo without sharding technique, especially when the size of the data is very large. When the comparison was done on the Cassandra database, it got better results when using the portioning technique than using it without portioning. This study also compared two types of partitioning in Cassandra database. According to CQNS experiments the edge portioning has got better results than vertex, especially when using a large size of data. When adding the comparison process with Haery framework, the Cassandra with CQNS framework achieved better results than Haery, but the results of Haery using Mongo database are relatively better than CQNS results with no sharding. On the other hand, when CQNS applied the sharding technique, the results obtained are better than results obtained from Mongo and Cassandra databases, when using a large size of data.

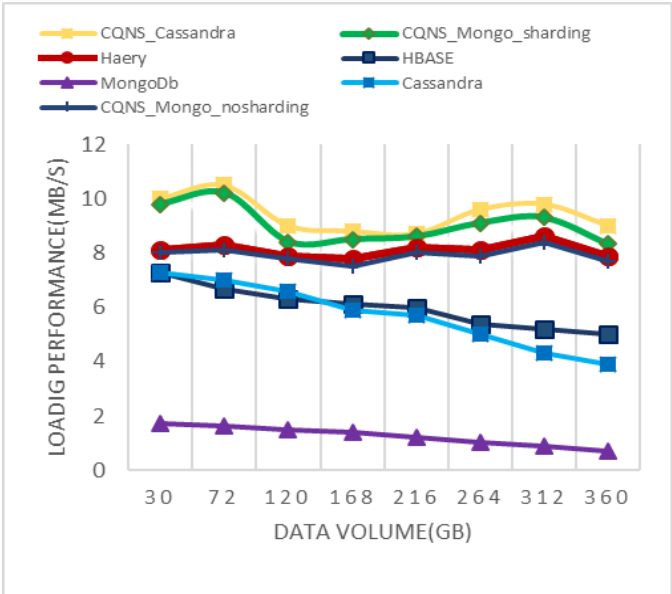

Fig. 13. Comparison of loading speed among six data stores.

V. Conclusion And Future Work

It is difficult to perform complex queries across many data stores. Unlike applications that use a single data store, there is a request for additional efforts. This paper introduced a framework to handle complex query database. This framework consists of three layers. The first of which is responsible for defining the database engine that id matched with the user query sentences. In second layer the system sends user queries to a processing layer containing Hadoop HDFS to store data, the k-aggregation algorithm with MapReduce. The last layer either works with the sql engine or the selected NOSQL Engine to do the job required. It should be noted that firstly, a vector holding the names of the SQL and NOSQL Engine is created to help in defining the database engine matched with the user query. This paper proposes a time model for calculating time cost and therefore it used Sharding technology with Mongo database queries to segment data and reduce the time used to query. On the other hand, this study used two types of partitioning in Cassandra database, one of them is the Edge and the second is Vertex.

There is an intention to develop a future plan that will improve performance of the revised approach and add libraries for more databases to suit the needs of the different users. Moreover, other types of NoSQL databases such as Sybase, Oracle, Access and other NoSQL databases may be considered to demonstrate and expand the proposed framework.

REFERENCES

[1] R. Sellami, B. Defude, "Complex Queries Optimization and Evaluation Over Relational and NoSQL Data Stores in Cloud Environments," Ph.D. dissertation, University of Paris-Saclay, France, 2017. [Online].

[2] P. Sangat, M. Indrawan-Santiago, D. Taniar, Sensor data management in the cloud: Data storage, data ingestion, and data retrieval, Concurrency Computat: Pract Exper. 2018; 30: e4354., 2017. [Online]. Available: https://doi.org/10.1002/cpe.4354.

[3] Baruffa, G., Femminella, M., Pergolesi, M., & Reali, G.: Comparison of MongoDB and Cassandra Databases for supporting Open-Source Platforms tailored to Spectrum Monitoring as-a-Service. IEEE Transactions on Network and Service Management (2019).

[4] Khan, Y., Zimmermann, A., Jha, A., Gadepally, V., D'Aquin, M., & Sahay, R.: One size does not fit all: querying web polystores. Ieee Access, 7, 9598-9617 (2019).

[5] Duggan, J., et al.: The BigDAWG polystore system. SIGMOD Rec. 44(2), 11–16 (2015)

[6] Xiang Li, Zhiyi Ma, Hongjie Chen, "QODM: A query-oriented data modeling approach for NoSQL databases," 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA). [Online].

[7] J. Roijackers and G. H. L. Fletcher, "On bridging relational and document-centric data stores," in Big Data - 29th British National Conference on Databases, BNCOD'13, 2013, pp. 135–148.

[8] Sharma, M., Sharma, V. D., & Bundele, M. M. (2018, November). Performance Analysis of RDBMS and No SQL Databases: PostgreSQL, MongoDB and Neo4j. In 2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE) (pp. 1-5). IEEE.

[9] M. Armbrust and et al., "Spark SQL: relational data processing in spark," in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015, 2015, pp. 1383–1394.

[10] H. Garcia-Molina and et al., "The TSIMMIS approach to mediation: Data models and languages," J. Intell. Inf. Syst., vol. 8, no. 2, pp. 117–132, 1997.

[11] IBM, "Ibm nosql: Ibm informix - introducing nosql capabilities a technical white paper," Tech. Rep., November 2013.

[12] S. K Pandey, Sudhakar, Context based Cassandra query language , 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT).

[13] Aaron Schram and Kenneth M. Anderson., "MySQL to NoSQL: data modeling challenges in supporting scalability Tucson, Arizona, USA — October 19 - 26, 2012.

[14] Ferro, M., Fragoso, R., & Fidalgo, R. (2019, July). Document-Oriented Geospatial Data Warehouse: An Experimental Evaluation of SOLAP Queries. In 2019 IEEE 21st Conference on Business Informatics (CBI) (Vol. 1, pp. 47-56). IEEE.

[15] Song, J., He, H., Thomas, R., Bao, Y., & Yu, G. (2019). Haery: a Hadoop based Query System on Accumulative and High-dimensional Data Model for Big Data. IEEE Transactions on Knowledge and Data Engineering.

[16] Samanta, A. K., Sarkar, B. B., & Chaki, N. (2018, November). Query Performance Analysis of NoSQL and Big Data. In 2018 Fourth International Conference on Research in Computational Intelligence and Nasholm, Petter. "Extracting data from NoSQL databases." University of Gothenburg, Gothunburg (2012).

[17] Communication Networks (ICRCICN) (pp. 237-241). IEEE.

[18] Agarwal, S. & Rajan, K.S; "Performance analysis of MongoDB versus PostGIS/PostGreSQL databases for line intersection and point containment spatial queries"; vol. 24(6), pp. 671–677, Springer; doi: https://doi.org/10.1007/s41324-016-0059-1.

[19] J.M. Patel, "Operational NoSQL Systems: What's New and What's Next?", Computer, vol. 49, no. 4, pp. 23-30, Apr. 2016.

[20] Elghamrawy, S.M. and Hassanien, A.E., 2017. A partitioning framework for Cassandra NoSQL database using Rendezvous hashing. The Journal of Supercomputing, 73(10), pp.4444-4465.

[21] Dipietro, Salvatore, Rajkumar Buyya, and Giuliano Casale. "PAX: Partition-aware autoscaling for the Cassandra NoSQL database." NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2018.

[22] Vasavi, S., M. Padma Priya, and Anu A. Gokhale. "Framework for Geospatial Query Processing by Integrating Cassandra with Hadoop." Knowledge Computing and Its Applications. Springer, Singapore, 2018. 131-160.

[23] Mearaj, I., Maheshwari, P., & Kaur, M. J. (2018, November). Data Conversion from Traditional Relational Database to MongoDB using XAMPP and NoSQL. In 2018 Fifth HCT Information Technology Trends (ITT) (pp. 94-98). IEEE.

[24] Zhang, D., Wang, Y., Liu, Z., & Dai, S. (2019). Improving NoSQL Storage Schema Based on Z-Curve for Spatial Vector Data. IEEE Access, 7, 78817-78829.

[25] Yassine, F., & Awad, M. A. (2018, November). Migrating from SQL to NOSQL Database: Practices and Analysis. In 2018 International Conference on Innovations in Information Technology (IIT) (pp. 58-62). IEEE.

[26] Gunawan, R., Rahmatulloh, A., & Darmawan, I. (2019, July). Performance Evaluation of Query Response Time in The Document Stored NoSQL Database. In 2019 16th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering (pp. 1-6). IEEE.

[27] Pratama, F. A., & Mutijarsa, K. (2018, October). Query Support for Data Processing and Analysis on Ethereum Blockchain. In 2018 International Symposium on Electronics and Smart Devices (ISESD) (pp. 1-5). IEEE.

[28] Abbas, Zainab, et al. "Streaming graph partitioning: an experimental study. " Proceedings of the VLDB Endowment 11.11 (2018): 1590-1603.

[29] R. Sellami, S. Bhiri, and B. Defude, "Supporting multi data stores applications in cloud environments," IEEE Trans. Services Computing, vol. 9, no. 1, pp. 59–71, 2016.

[30] R. Sellami, "Supporting multiple data stores-based applications in cloud environments," Ph.D. dissertation, University of Paris-Saclay, France, 2016. [Online]. Available: https://tel. archives-ouvertes.fr/tel-01280236.

[31] R. Sellami and B. Defude, "Using multiple data stores in the cloud: Challenges and solutions," in Data Management in Cloud, Grid andP2P Systems - 6th International Conference, Globe 2013, Prague, Czech Republic, August 28-29, 2013. Proceedings, 2013, pp. 87–98.

[32] R. Sellami, S. Bhiri, and B. Defude, "ODBAPI: A unified REST API for relational and nosql data stores," in 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, June 27 - July 2, 2014, 2014, pp. 653–660.

[33] R. Sellami and et al., "Automating resources discovery for multiple data stores cloud applications," in CLOSER 2015 – Proceedings of the 5th International Conference on Cloud Computing and Services Science, Lisbon, Portugal, 20-22 May, 2015., 2015, pp. 397–405.

[34] https://docs.mongodb.com/manual/core/sharded-cluster-requirements/

[35] https://github.com/johnnywidth/cql-calculator

[36] Abdel-Hamid, N.B., ElGhamrawy, S., El Desouky, A. and Arafat, H., 2018. A dynamic spark-based classification framework for imbalanced big data. Journal of Grid Computing, 16(4), pp.607-626.

[37] Elghamrawy, S.M., 2016, October. An adaptive load-balanced partitioning module in Cassandra using rendezvous hashing. In International Conference on Advanced Intelligent Systems and Informatics (pp. 587-597). Springer, Cham.

Eman A. Khashan is MSc student at the Computer Science Department, Faculty of Engineering, Mansoura University, Egypt. She obtained a B.Sc. in Computer Sciences. Her research interest is on NoSQL Databases, big data, data Clustering, Join queries, Hadoop HDFS/MapReduce, and Spark. She is focusing on the management of applications deployment on multiple data stores providers. She has been delivered lectures and has been gave a practical training in the grants from the Ministry of Communications and information technology with collaboration with MCIT. She received a certificate MCSA (Microsoft® Certified Systems administrator) Inc. she is also having MCSD (Microsoft® Certified Solution Developer) and MCP (Microsoft® Certified Professional). She also did practical training for students who obtained the ITI program to qualify them for the job market. She worked as .net developer for Windows, Web and Mobile applications.

Ali Ibrahim Eldesouky is a full professor for Computers Engineering and Systems department at Faculty of engineering, Mansoura University in Egypt. He is also a visiting part time professor for MET Academy. He obtained his MA and Ph.D. from the University of Glasgow in the United States of America. He teaches in American and Mansoura universities; he took over many. positions of leadership and supervision of many scientific papers. He has published hundreds of articles in well-known international journals. He was an experienced member of the Subcommittee of the Communications and Information Engineering Program. Also, he was a member of the Scientific Council for the Bachelor's Program in Communication Engineering and Information Systems. He worked as head of the Department of Computer and Systems Engineering, Faculty of Engineering, Mansoura University.

Sally M. El-Ghamrawy is the Head of Communications & Computer Engineering Department at **MISR higher Institute for Engineering** & Technology and part time Associate Professor at Electrical & Computers Engineering Department, Faculty of Engineering, **British University in Egypt BUE** and at computers engineering department -Faculty of engineering, **Mansoura University** in Egypt. She received a Ph.D. degree in **2012** in Distributed Decision Support Systems Based on Multi Intelligent Agents and received a M. SC degree in Automatic Control Systems Engineering in **2006 from** computers engineering department -Faculty of engineering, **Mansoura University**, and received B. Sc. in Computers Engineering and Systems in **2003**. She received the Federation of Arab Scientific Research Councils **Prize 2019** - in the field of artificial intelligence. She is delivering lectures, supervising graduation projects, master's thesis, and doctoral dissertations. She was delivering lectures and gave a practical training in the grants from the Ministry of Communications and information technology with collaboration with IBM. She received a certificate A+ International Inc. CompTIA. She is a member in Scientific Research Group in Egypt. Her research focuses on Big Data analysis, No-SQL databases, Artificial Intelligence techniques, and software engineering. She is the author of number peer-reviewed publications, receiving best paper awards. She is also an **IEEE Member.**