Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2022, 9(7):146-150**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

An Adaptive Spark-based Framework for Querying Large-Scale NOSQL and Relational Databases

Naveen Muppa

10494 Red Stone Dr Collierville, Tennessee

Abstract The growing popularity of big data analysis and cloud computing has created new big data management standards. Sometimes, programmers may interact with a number of heterogeneous data stores depending on the information they are responsible for: SQL and NoSQL data stores. Interacting with heterogeneous data models via numerous APIs and query languages imposes challenging tasks on multi-data processing developers. Indeed, complex queries concerning homogenous data structures cannot currently be performed in a declarative manner when found in single data storage applications and therefore require additional development efforts. Many models were presented in order to address complex queries Via multistore applications. Some of these models implemented a complex unified and fast model, while others' efficiency is not good enough to solve this type of complex database queries. This paper provides an automated, fast and easy unified architecture to solve simple and complex SQL and NoSQL queries over heterogeneous data stores (CQNS). This proposed framework can be used in cloud environments or for any big data application to automatically help developers to manage basic and complicated database queries. CQNS consists of three layers: matching selector layer, processing layer, and query execution layer. The matching selector layer is the heart of this architecture in which five of the user queries are examined if they are matched with another five queries stored in a single engine stored in the architecture library. This is achieved through a proposed algorithm that directs the query to the right SQL or NoSQL database engine. Furthermore, CQNS deal with many NoSQL Databases like MongoDB, Cassandra, Riak, CouchDB, and NOE4J databases. This paper presents a spark framework that can handle both SQL and NoSQL Databases. Four scenarios' benchmarks datasets are used to evaluate the proposed CQNS for querying different NoSQL Databases in terms of optimization process performance and query execution time. The results show that, the CQNS achieves best latency and throughput in less time among the compared systems.

Keywords This proposed framework can be used in cloud environments or for any big data application to automatically help developers to manage basic and complicated database queries. CQNS consists of three layers: matching selector layer, processing layer, and query execution layer. The matching selector layer is the heart of this architecture in which five of the user queries are examined if they are matched with another five queries stored in a single engine stored in the architecture library. This is achieved through a proposed algorithm that directs the query to the right SQL or NoSQL database engine. Furthermore, CQNS deal with many NoSQL Databases like MongoDB, Cassandra, Riak, CouchDB, and NOE4J databases. This paper presents a spark framework that can handle both SQL and NoSQL Databases. Four scenarios' benchmarks datasets are used to evaluate the proposed CQNS for querying different NoSQL Databases in terms of optimization process performance and query execution time.

1. 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 much research focused on, is that there is no standard method to execute complex queries across NoSQL Databases. 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 due to 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. Fig 1 shows a diagram of integrating heterogenous relational and NoSQL datasets to an example of scientific social network.

2. 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 stages. Matching Selector stage, processing stage, and query execution stage. In the following sections, this paper discusses the different stages of the proposed CQNS.

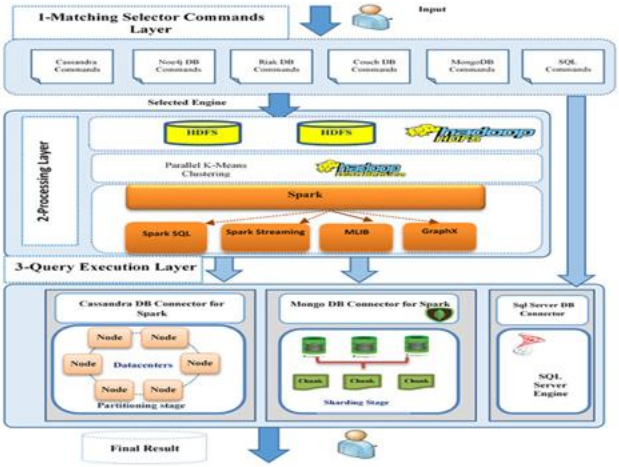

Figure 1: The proposed framework (CQNS).

3. Matching Selector Stage

This stage 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 an example of relational database and MongoDB, Cassandra, Couch, Riak and NOE4J 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 stage 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). The stored SQL libraries statements for SQL database while explain the CRUD statements for MongoDB and Cassandra DB respectively, as examples of the NoSQL Database libraries used in this paper.

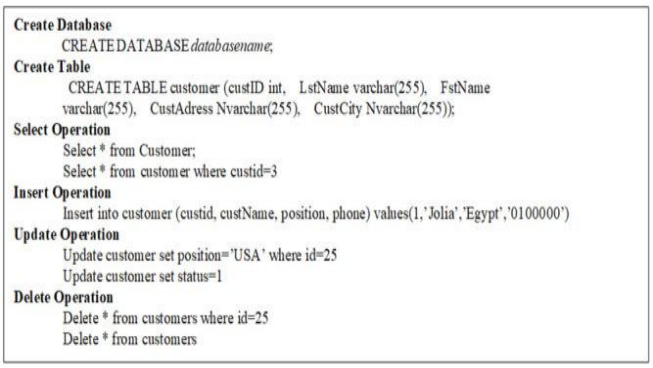

Figure 2: SQL Libraries.

4. CQNS Processing Stage

Within this section you will shortly find details about technology and the setup process for this study. CQNS deployed and used Hadoop/HDFS 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. The formula (1) uses to calculate HDFS node storage (H) required:

H: denoted the HDFS node storage required

C: is the compression ratio and completely depends on the type of compression used and size of the data.

R: It is the replication factor which is 3 by default in production cluster.

S: S denotes the initial amount of data you need to move to Hadoop.

I: I represent the intermediate data factor which is usually 1/3 or ¼. It is Hadoop's intermediate working space used to store the intermediate results of different tools like Hive.

1.2: 1.2 or 120% more than the total size.

$$H=C*R*S(1-I)*1.2 \quad (1)$$

MapReduce 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 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. 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 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.

5. Query Execution Stage

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: $X \rightarrow \{1, 2, \ldots, k\}$ when X is a set of objects, the Number of clusters $K \in Z+$ and Distance function $d \in R0^{+}$

between all pairs of objects in X, partition X into K disjoint sets x1, x2, ….., xk such that ∑k∑x,x'∈Xkd(x,x') With N = |X|, the number of distinct cluster assignments possible as follows

S(N,K)=1K!∑Kk=1−1K−k(Kk)kN

MongoDB engine Sharding is a way to distribute data across multiple devices. This work provides MongoDB which utilizes Sharding to benefit implementations using very huge databases and structures that are highly efficient. Data stores which contain large datasets or high-productivity applications may challenge a single server 's ability. High query rates for example can exceed the server's CPU capability. A number of sizes greater than the RAM of the device will help validate driver I / O capability. A database may have a mix of sharded collections and unshared collections. Sharded sets are divided into a cluster and spread throughout the shards. Unshared collections on a main shard are stored. As shown in Fig 9 each database has its own main shard.

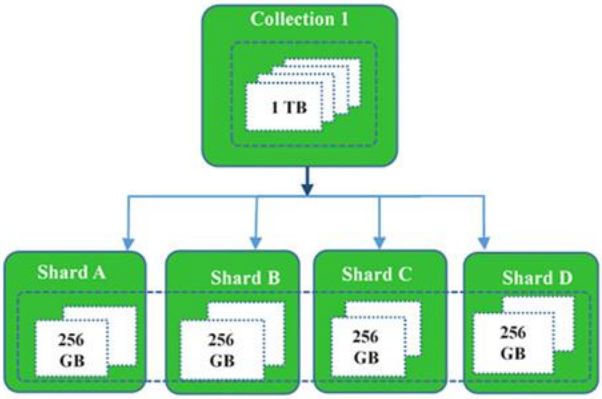

Figure 3: Sharding Mongo DB Stage.

6. CQNS Evaluation

CQNS is used to store, manage, and execute bigdata queries and makes the development job very much easier. In this paper, the proposed model rewrites each query into the particular query language of the integration data store. The processing stage 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.

7. Cost Model

The cost of implementation is the sum of the costs of each process that composes the implementation plan. This is worth noting that costs do not reflect time directly. More cost means more time, of course. It is used to compare two question execution plans, but not for estimating the responding time directly. The multiplication of the matrix between the α, β and α coefficients row was determined to test the expense formulation according to each data store. A column vector includes the values of the catalog parameters and a fixed variable known as const, which is a scale and may be a cardinality, a number, etc. Furthermore, the value of the column vectors will be empty if the parameter is not based on a particular calculation (CPU cost, I / O cost or the costs of connections). This is determined as follows: Matrix multiplication:

$$\text{const}(\alpha\beta\gamma)\begin{pmatrix} \text{tcpu} \\ \text{ti/o} \\ \text{tconn} \end{pmatrix}=\text{const}(\alpha\times\text{tcpu}+\beta\times\text{ti/o}+\gamma\times\text{tconn})$$

Where tcpu symbolized the CPU time, ti/o symbolized the Input /Output time and tconn symbolized the connection time to engine.

The α, β and α are represents the coefficients that were considered to evaluate the cost model. These of parameters are separately defined to each data store, and a constant variable called const which is a scalar and can be a cardinality, selectivity, etc.

The projection cost and selection processes that are referenced by costProjcton and CostRestrction are started with forms respectively (see formula 7 and formula 8). The operating cost is the linear mix of variables for initProjcton (resp. InitSelection) and scanning and dropping (resp. selection). As the input variable N,

the cost Projcton indicates the integrated data store in which the connection is executed, variable H indicates the sum of the entity size specified in the entry, and the estimate function named estm. This function allows to measure how much the primary projection (resp. selection) is carried out. The range and cardinality variables are referred to:n: a node, h: length of an entity set, projatt: Projection attributescstproj(n,h,estm(projatt,h))=initproj(n)+scan(n)*h+proj(n)*estm(projatt,h)

References

[1]. https://figshare.com/articles/dataset/The_advantages_and_disadvantages_of_the_proposed_framework_and_the_most_recent_frameworks_/15442181/