Federated Query processing for Big Data in Data Science

Manoj Muniswamaiah, Tilak Agerwala and Charles C. Tappert
Seidenberg School of CSIS, Pace University, White Plains, New York
{mm42526w, tagerwala, ctappert}@pace.edu

***Abstract*—As the number of databases continues to grow data scientists need to use data from different sources to run machine learning algorithms for analysis. Data science results depend upon the quality of data been extracted. The objective of this research paper is to implement a federated query processing framework which extracts data from different data sources and stores the result datasets in a common in-memory data format. This helps data scientists to perform their analysis and execute machine learning algorithms using different data engines without having to convert the data into their native data format and improve the performance.**

***Index Terms*—Machine learning, big data, federated query, database, query optimizer, in-memory data, data science.**

I. Introduction

Within an organization data is stored in different databases since no one database can support all types of data been collected from the sources. Hence querying all the databases for required datasets is one of the essential tasks for data scientists. Ability to access data stored across organization is important for input and training machine learning algorithms. Data scientist may be interested in querying the relational database and store the queried data in an array database to synch it with the images and further store the extracted features from the images in a graph database to use it with the machine learning algorithms [1]. This type of tasks are increasingly common today and execution of federated query which has cross table joins is similar to the traditional execution of the query plan. Federated querying is poorly supported due to lack of solutions for efficiently moving and combining the intermittent query results obtained from different databases [2].

Machine learning mathematical models are built using training data in order to make decisions without having to be programmed to execute various tasks. These algorithms are been used in various fields. With the growth of big data and technology, machine learning has become a mainstream presence for data scientists. The algorithms used depends upon the kind of problem been solved. There are two main categories of machine learning algorithms regression and classification. Regression consists of numeric data while classification involves mostly non-numeric data. Apart from these categories there is supervised learning and unsupervised learning [3].

.

Data scientists need to clean the data to be used for machine learning by excluding irrelevant attributes. This can be implemented during the data pre-processing pipeline. Some of the popular machine learning algorithms are: Linear regression used for numerical data. Logistic regression and Support Vector Machines used in binary classification of data. Linear discriminant analysis used for classification in multi-category. Decision tree, KNN, Learning Vector Quantization and Naïve Bayes are used in both classification and regression models. AdaBoost and XGBoost are ensemble algorithms which builds on the previous models [4].

Federated querying involves data movement among different databases, each of them having their own data formats. Traditional Extract-Transfer-Load practice collects data from different sources and performs data cleansing process which are later loaded to the data warehouse and data marts from which the users can retrieve data for performing their tasks. Databases support serialization of data from internal native format to external one. Generally it requires exporting the data from one format to another using a common data format and later obtaining the result. This task is costly because the data needs to be serialized from the source and persisted in a disk and later it needs to be de-serialized and imported into the target database and be queried. Another approach to support federated querying for data scientists is to use a common memory oriented columnar data format which avoids using disk and can be utilized by all databases and execution engines [5]. Common data implementation protocols like JDBC are good in transferring data between databases efficiently. This paper extends these JDBC protocols which are widely used today to create a common in-memory data format which can be utilized by data scientists for execution of their machine learning algorithms from various sources efficiently and with improved performance [6].

II. PROBLEM STATEMENT

Consider a data scientist working in a healthcare domain who wants to conduct hypothesis and confirm them for interesting patterns and outliers by using machine learning algorithms. The patient data collected would be stored in different databases. The medical device reading would be stored in a time series database, doctor notes and prescription in a NoSQL and patients details in a relational database.

Typical approach which data scientists adopt is to query data from databases and later export the intermediate results into a file system. Since this approach requires the intermediate data results to be materialized it leads to performance issues. Data movement of large datasets through file systems is expensive.

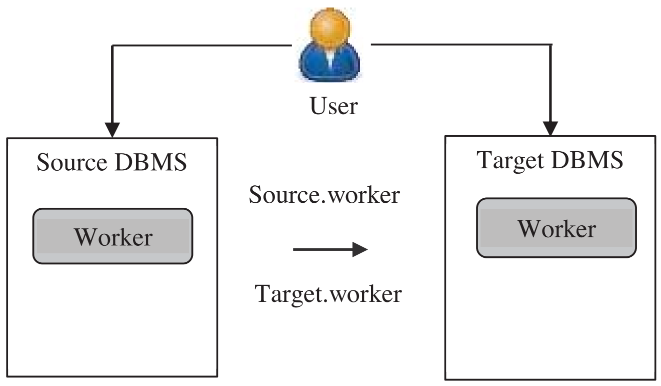

Figure 1: Data conversion from source to target database

As shown in Figure 1, the data first needs to be converted from source to target database format using an intermediate data materialization format and then later the query needs to be executed on the target database. This is a tedious task and does not scale well. This research work extends JDBC protocol which are supported by all the databases by storing the result set fetched from different databases in a common in-memory columnar format. This eliminates the need to serialize and de-serialize data which is a costly operation and allows different execution engines to make use of this common in-memory data for the execution of machine learning algorithms. It allows different data scientists to manipulate the data according to their needs and perform the tasks [7].

III. ARCHITECTURE

Big data needs to be stored and processed in different databases which are specialized for storage in different formats. There are also multiple data execution engines which process this data. Databases and execution engines use their own data formats and as the data moves between them, it is serialized and de-serialized based on the source and target system data formats. This creates a performance bottleneck to data scientists for execution of machine learning algorithms [8].

Most of the databases support JDBC protocol for querying of data. The data types used for each of their versions is different and results in performance overhead. In this research apache arrow [8] has been used and extended which is cross-language independent in-memory data storage format for processing of data by various engines. The data retrieved from different data sources by the federated query would be stored in the same in-memory format for processing. The customized federated SQL framework queries the data sources and converts the JDBC objects in to an in-memory data formats which can be used by

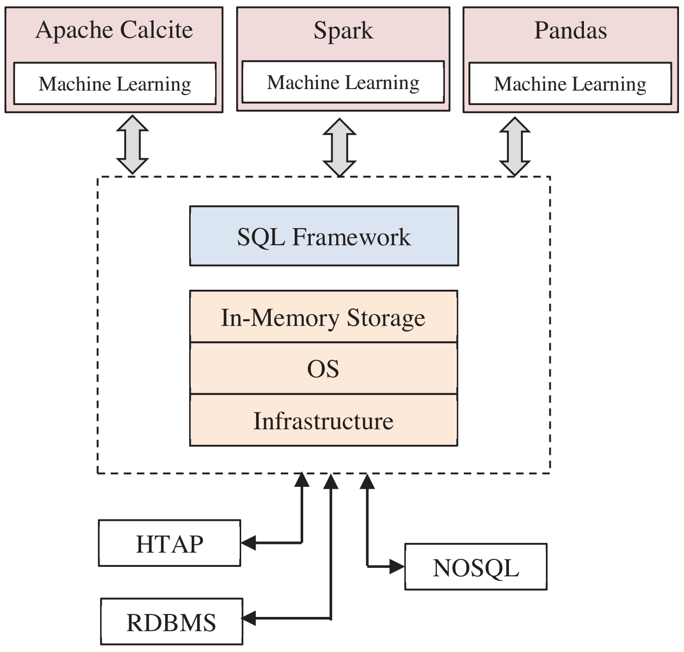

Figure 2: Overall Architecture of federated query for machine learning processing

various engines and help data scientists in their tasks. The row-wise data from the data sources are converted into columnar format in-memory vectors which can be utilized by upstream data pipelines. It provides inter-process communication and there is no copy of the data during processing.

Data libraries such as PySpark, Pandas, Numpy, Apache calcite, JDBC protocols use their own data format representations. By using common in-memory format interoperability between these systems can be increased. It can also be adopted for machine learning frameworks like TensorFlow which reduces the training and processing times. Advantages of using federated query framework with common in-memory format includes having a common data access to all the databases, there is zero-copy of the data, it is cache-efficient for analytical workloads and provides SIMD(single instruction, multiple data) factorization processors aiding machines learning algorithms. It assists in random access of the data handling flat tables and nested data structures like JSON with less overhead. JSON data structures has been widely adopted in organizations for its support to represent business problems in hierarchical structure. It also helps in reducing the network congestion between server and client application for distributing the data load.

The latency for fetching data from data sources depends upon the number of disk seeks and data obtained per each seek. In row-wise databases the data is written continuously to the disk which is efficient for write operations but a data scientist may need to fetch selective columns from larger set of data from the database for machine learning algorithms. Having a columnar representation of data would reduce latency of the query and also reduce number of disk seeks. Columnar data in-memory varies from columnar data on the disk format which is determined by I/O latency factor and also type of compression been adopted. In-memory columnar format is made faster by the cache locality and the SIMD vectors. It improves the data movement within the system without serialization and deserialization of the data when a federated query fetches from two different data sources. This research extends a customized SQL framework to fetch data from different sources through JDBC protocol and store them in an in-memory columnar format. This helps in having a common base datasets which can be manipulated later by users according to their task requirements.

The in-memory data format is interoperable and the data can be used from one process to another without serialize and deserialization. It also supports nested hierarchical data. With today's CPU which is faster and optimized it is important to make full use of its cycles and with data in columnar format it is best for SIMD operations processing.

IV. EVALUATION

The data is loaded into in-memory data format using the JDBC protocol through a federated query from different sources. For experimentation we used MIMIC III dataset [9]. It has over 50,000 to 2,000,000 rows of data related to patient information and medical device generated data been de-identified. The patient data was stored in H2 database [10] and medical device data in Apache Ignite [11]. The queried table contents sample is show below.

Select subject_id, hadm_id, diagnosis from admissions

'BIGINT_FIELD5=40036, 40080, 40084'
'BIGINT_FIELD5=198489, 162107, 195762'
'VARCHAR_FIELD13=SEPSIS, CONGESTIVE HEART FAILURE, INTRACRANIAL HEMORRHAGE'

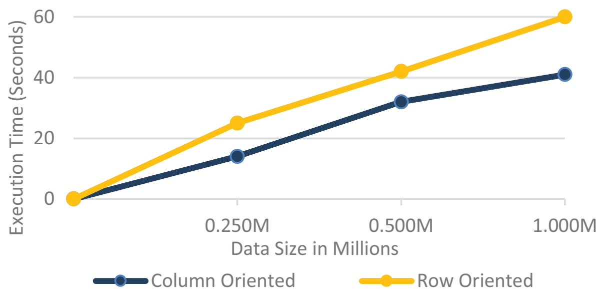

Figure 3: Comparison of data query execution time of in-memory row and columnar data format.

In-memory row-wise H2 database was compared against the in-memory common columnar format. It shows considerable improvement in the latency and execution time as shown in Figure 3. Also when the databases were accessed from the query engine directly for processing there were delays due to conversion of data into their native data formats [12, 13, 14].

Using the in-memory common data format the query engines were able to process efficiently as shown in Figure 4.

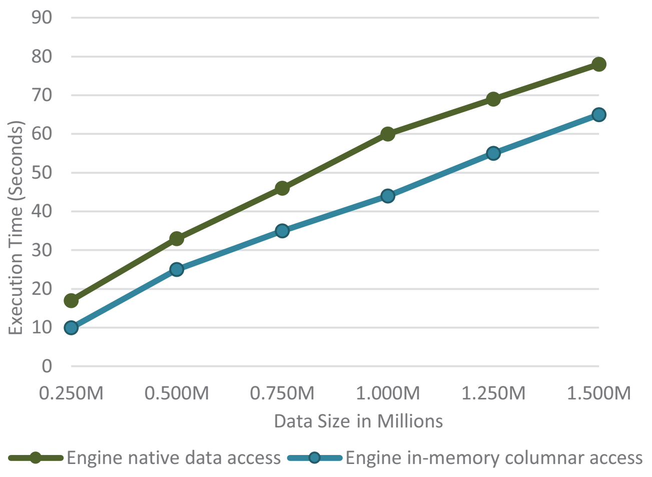

Figure 4: Query execution time by data engine in accessing the native data directly as compared to columnar data

V. CONCLUSION

This research introduces a federated query framework for quicker retrieval of data for data scientists, a tool that efficiently queries data between databases using a common in-memory data format. As a part of future work cloud databases can be integrated with the framework.

REFERENCES

[1] Flach, P. (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.
[2] Jordan, Michael I., and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects." *Science* 349.6245 (2015): 255-260.
[3] Hastie, Trevor, et al. "The elements of statistical learning: data mining, inference and prediction." *The Mathematical Intelligencer* 27.2 (2005): 83-85
[4] https://www.infoworld.com/article/3394399/machine-learning-algorithms-explained.html
[5] Hellerstein, Joseph M., et al. "Adaptive query processing: Technology in evolution." *IEEE Data Eng. Bull.* 23.2 (2000): 7-18.
[6] Yang, Timothy, et al. "Applied federated learning: Improving google keyboard query suggestions." *arXiv preprint arXiv:1812.02903* (2018).
[7] Fujino, Takahisa, and Naoki Fukuta. "A SPARQL query rewriting approach on heterogeneous ontologies with mapping reliability." *2012 IIAI International Conference on Advanced Applied Informatics*. IEEE, 2012.
[8] https://arrow.apache.org/
[9] https://mimic.physionet.org/
[10] https://www.h2database.com/html/main.html
[11] https://ignite.apache.org/
[12] https://calcite.apache.org/
[13] https://spark.apache.org/
[14] https://pandas.pydata.org/