QDrill: Query-Based Distributed Consumable Analytics for Big Data

Shadi Khalifa, Patrick Martin
School of Computing
Queen's University
Kingston, ON, Canada
khalifa, martin@cs.queensu.ca

Dan Rope, Mike McRoberts
IBM
Washington D.C., USA
drope, mtm@us.ibm.com

Craig Statchuk
IBM
Canada
craig.Statchuk@ca.ibm.com

***Abstract*— Consumable analytics attempt to address the shortage of skilled data analysts in many organizations by offering analytic functionality in a form more familiar to in-house expertise. Providing consumable analytics for Big Data faces three main challenges. The first challenge is making the analytics algorithms run in a distributed fashion in order to analyze Big Data in a timely manner. The second challenge is providing an easy interface to allow in-house expertise to run these algorithms in a distributed fashion while minimizing the learning cycle and existing code rewrites. The third challenge is running the analytics on data of different formats stored on heterogeneous data stores.**

In this paper, we address these challenges in the proposed QDrill. We introduce the Analytics Adaptor extension for Apache Drill, a schema-free SQL query engine for non-relational storage. The Analytics Adaptor introduces the Distributed Analytics Query Language for invoking data mining algorithms from within the Drill standard SQL query statements. The adaptor allows using any sequential single-node data mining library (e.g. WEKA) and makes its algorithms run in a distributed fashion without having to rewrite them. We evaluate QDrill against Apache Mahout. The evaluation shows that QDrill outperforms Mahout in Updatable model training and scoring phase while almost keeping the same performance for Non-Updatable model training. QDrill is more scalable and offers an easier interface, no storage overhead and the whole algorithms repository of WEKA, with the ability to extend to use algorithms from other data mining libraries.

Keywords- Big Data; Analytics; SQL; Data Mining;Distributed;Apache Drill; WEKA;

I. INTRODUCTION

Big Data Analytics is a multidisciplinary mix of art and science that extracts useful insights from Big Data, which is data that arrives in huge volumes, at a rapid velocity, with no obvious way of telling the veracity of it. Data that outgrown the ability to be stored and processed by many of the traditional systems [1]. Analytics offers organizations the means to discover hidden patterns in such data and use these patterns to predict the likelihood of future events. However, organizations need to overcome a number of challenges to reap the benefits of Analytics.

The *first challenge* is enabling the distributed execution of the existing "arsenal" of data mining algorithms. The majority of existing data mining libraries (e.g. *R*[1], *WEKA*[2], *RapidMiner*[3], etc.) only support sequential single-node execution of these algorithms. This makes these libraries unsuitable for dealing with Big Data.

Scalable distributed data mining libraries like *Apache Mahout*[4], *Cloudera Oryx*[5], *Oxdata H2O*[6], *MLlib*[7] [2] and *Deeplearning4j*[8] rewrite the data mining algorithms to run in a distributed fashion on *Hadoop* [3] and *Spark* [4]. These libraries are developed by searching the algorithms for parts to be executed in parallel and rewriting them. This process is complex, time consuming and the quality of the modified algorithm depends entirely on the contributors' expertise. This makes these libraries hard to maintain and extend [5].

Another approach to distribute the data mining algorithms while keeping the same familiar interface is to add support for *MapReduce* [6] to the sequential single-node data mining libraries to enhance their scalability. *Distributed Weka Base*[9], *Distributed Weka Hadoop*[10] and *Distributed Weka Spark*[11] [5] packages extend WEKA to access the *Hadoop Distributed File System (HDFS)* [7], *Hadoop* and *Spark*, respectively. *RHadoop*[12] allows running R code on Hadoop and access to HDFS. These extensions, however, leave it to users to put the data into the right format, create the right meta-data and write the MapReduce wrappers for the data mining algorithms.

The *second challenge* is for organizations to acquire the needed skill set to carry out the Analytics process. This is a problem since Analytics is multidisciplinary. It is the application of computer science, data storage, data mining and machine learning, statistical analysis, artificial intelligence and pattern recognition, visualization, operations research, Business Intelligence (BI) and business and domain knowledge to real-world data sources to bring understanding and insights to data-oriented problem domains [8].

Consumable Analytics is one of the main trends to address this challenge and overcome the Analytics skill gap by making Analytics more adoptable. Consumable Analytics refers to increasing the impact of the skills already existing in the organization by producing tools that make analytics easier to build, manage, and consume [9]. Consumable Analytics

[1] R: https://www.r-project.org/
[2] WEKA: http://www.cs.waikato.ac.nz/ml/weka/
[3] RapidMiner: https://rapidminer.com/
[4] Mahout: https://mahout.apache.org/
[5] Oryx: https://github.com/cloudera/oryx
[6] H2O: http://0xdata.com/h2o-2/
[7] MlLib: https://spark.apache.org/mllib/
[8] Deeplearning4j: http://deeplearning4j.org/
[9] DistributedWekaBase: http://goo.gl/wcJrCa
[10] DistributedWekaHadoop: http://goo.gl/69lVLE
[11] DistributedWekaSpark: http://goo.gl/sWngFD
[12] RHadoop: https://goo.gl/CsZad3

can be in the form of using a familiar interface or programming language. It can be in the form of simplifying or hiding the distributed deployment, data access and execution of the analytics jobs.

The *third challenge* is providing seamless data integration, which involves joining data of different formats (structured, semi-structured and unstructured) that can be distributed across heterogeneous data stores (HDFS, Relational Databases, NoSQL Databases, etc.). Most of the existing libraries use an Extract-Transform-Load (ETL) operation to extract the data from the different stores and transform their format to an acceptable schema. This approach is time consuming, requires defining a schema for the data and requires having all data available beforehand.

Apache Drill[13] is an open source implementation of the proprietary Google Dremel [10] (publicly available as Google BigQuery[14] service). Drill is a schema-free SQL query engine for non-relational storage (HDFS, Hive, MongoDB, etc.). It allows analyzing multi-structured and nested data directly without any ETL using a schema-on-read approach. It provides a JDBC/ODBC interface for querying and joining data from different sources using standard SQL statements. Drill thus makes use of the existing SQL skillsets and BI tools within an organization. Drill implements Massive Parallel Processing (MPP), nested data modelling and columnar storage to improve retrieval efficiency for interactive analytics scenarios.

In this work, we introduce *QDrill* which uses Apache Drill to address the seamless data integration challenge (Third challenge). QDrill then extends Drill by adding the Analytics Adaptor to address the other two challenges. The adaptor allows integrating any sequential single-node data mining library into Drill and running its algorithms in a distributed fashion from within the Drill SQL statements (First challenge). The adaptor does not require any modifications to the data mining algorithms. It uses the library APIs to access the algorithms and run them in parallel. The adaptor hides all data format transformations from users. It also provides users with the *Distributed Analytics Query Language (DAQL)*, a familiar SQL interface to invoke the data mining algorithms, eliminating the need to learn new programming languages or learning the data mining library APIs (Second challenge).

The rest of the paper is organized as follows: Section II summarizes the related work presented in the existing Analytics Query Languages. Following that, Apache Drill architecture is summarized in section III. Our Analytics Adaptor extension and the Distributed Analytics Query Language are introduced in section IV. Section V illustrates the evaluation of the proposed QDrill. Finally, section VI has the conclusions and future work.

II. Related Work

To many people working with data is synonymous to querying databases using SQL. However with Big Data, data does not only reside in relational databases. Analytics Query Languages (AQLs) provide a unified SQL interface over the different Big Data stores with a standard JDBC/ODBC interface to connect to the familiar BI tools. Some AQLs also provide data mining capabilities.

From these solutions, the Data Mining Query Language (DMQL) [11] attempts to establish a standard for data mining query languages. The DAEDLUS Framework [12] introduces the MO-DMQL that can be expressed using the 3W algebraic framework [13], which is similar to relational algebra. Microsoft introduces the Data Mining Extension (DMX) query language [14] in its SQL server to run in-database analytics. Hivemall [15] extends HiveQL [16] with a scalable data mining library. Hivemall allows models to be created and used from within HiveQL statements. Hivemall statements are then transformed to Hadoop batch jobs for execution. Meo et al. [17] propose a specialized SQL extension for only association rules mining. SQL-TS [18] is another specialized SQL extension that is highly optimized for complex time series queries and searching patterns in sequence.

Looking at the existing AQL solutions, we find the following shortcomings. Both DMQL and MO-DMQL are theoretical with no implementations. They also do not discuss distributed queries for handling Big Data. The language from Meo et al. only supports modeling association rules and SQL-TS only supports complex time series queries. The closest to achieving a fully functioning AQL for Big Data Analytics are DMX and Hivemall. They support different data models and use SQL and HiveQL, respectively for data exploration and preparation. However, they rewrite the data mining algorithms to run in a distributed fashion, thus their supported algorithms are still limited compared to other sequential single-node data mining libraries.

There are also some SQL-on-Hadoop solutions that offer basic querying functionalities on Hadoop, like filtering, aggregation and selection, but have no data mining capabilities. Among these solutions Apache Hive [16] translates the user's HiveQL queries to MapReduce batch jobs. Google Dremel [10] (publicly available as Google BigQuery service), Cloudera Impala[15] and Apache Drill[13] provide interactive querying on Big Data. Spark SQL (previously known as Shark) [19] uses in-memory computations to further accelerate query processing. Microsoft SCOPE [20] is a SQL-like language that creates optimized query execution plans inspired by parallel databases optimization techniques for Microsoft's MapReduce solutions Cosmos and Dryad [21].

III. Apache Drill

Apache Drill is an open source software with a very active contributors list. Drill is designed to allow accessing and joining data from heterogeneous non-relational data sources while providing the familiar interface of relational data stores. Drill uses the standard SQL syntax and JDBC/ODBC interface to submit queries to the non-relational data stores.

Drill supports queries on self-describing data like JSON, with the ability to flatten the nested data. It supports on-the-fly schema discovery, which enables execution to begin without knowing the structure of the data. Based on the data description within the submitted query, Drill automatically

[13] Drill: https://drill.apache.org/

[14] BigQuery: https://cloud.google.com/bigquery

[15] Impala: https://goo.gl/po3rIk

compiles the query during the execution phase to create the schema. As a result, Drill can handle schema-less data. This removes the constraint of having an ETL process and maintaining schemas before data can be analyzed.

Unlike other frameworks that translate queries to MapReduce jobs, Drill uses the Massive Parallel Processing (MPP) paradigm. The MPP paradigm splits the processing and data IO across multiple nodes, dividing the job across them. MPP nodes use a messaging interface to coordinate the job execution. This paradigm allows parallel search, processing and fetching of data.

Architecture-wise, Drill uses a multi-level processing architecture. Leaf processes communicate with the storage layer to optimize accessing the data in parallel. The leaf processes pass partial results to the intermediate processes, which perform parallel operations on the intermediate results. Intermediate processes then pass the aggregated results to the root process, which performs further aggregation and provides the final results to the client application. Drill, however, does not implement a dedicated root process. Instead, any Drill node (aka *Drillbit*) can accept queries and become the root process (aka *Driving Drillbit*), leading to a better load balancing when multiple queries are submitted.

Each *Drillbit* has a *Storage Adaptor* to optimize and provide access to the various data stores. The Storage Adaptor works with *Storage Plugins* that transform the data loaded from a data store to a unified internal data structure so that data of different formats coming from different sources can be joined and processed. In addition, Storage Plugins inform the execution engine of any native capabilities to speed up the processing, such as predicate pushdown. *Drill 1.4* comes with the following Storage Plugins preinstalled[16]: File System (CSV, JSON, Parquet, and Delimited Text), HDFS, HBase, Hive, MongoDB, S3, RDMS (MySQL, Oracle DB, MS SQL Server, and Postgres). The Storage Adaptor can also include user defined storage plugins for the other data stores.

Query execution-wise, when a Drill SQL query is submitted to an application, the Drill JDBC/ODBC interface is used to forward the query to a Drillbit that becomes the Driving Drillbit and coordinates the execution. The Driving Drillbit parses the query to a *logical plan* that describes the work required to generate the query results and defines which data sources and operations to apply. A *cost-based optimizer* is then used to apply various types of rules to rearrange the logical plan operators and functions to speed up the execution. The cost-based optimizer generates a *physical plan* that describes how to execute the query.

The physical plan is given to the *Parallelizer* which creates the *execution plan* by splitting the physical plan into multiple execution phases (*fragments*) that can be executed in parallel. The Parallelizer first fragments the plan into *major fragments.* A major fragment represents a phase of one or more related operations. Each major fragment is then parallelized into as many *minor fragments* as can be run at the same time on the cluster. A minor fragment is a logical unit of work that runs inside a thread. Drill schedules the minor fragments on nodes with data locality. Otherwise, Drill schedules them in a round-robin fashion on the available Drillbits.

IV. QDrill

Drill is powerful in terms of accessing and joining data from heterogeneous sources, which is usually a cumbersome task when done in data mining libraries. On the other hand, Drill does not have any data mining capabilities. Developing data mining algorithms for Drill is time consuming and so would likely be limited to a handful of algorithms, nothing compared to those available in the well-established data mining libraries. The proposed QDrill with the *Analytics Adaptor* solves these issues by using Drill to load and join data from heterogeneous sources and using the pre-existing data mining algorithms of well-established data mining libraries to train and score data mining models.

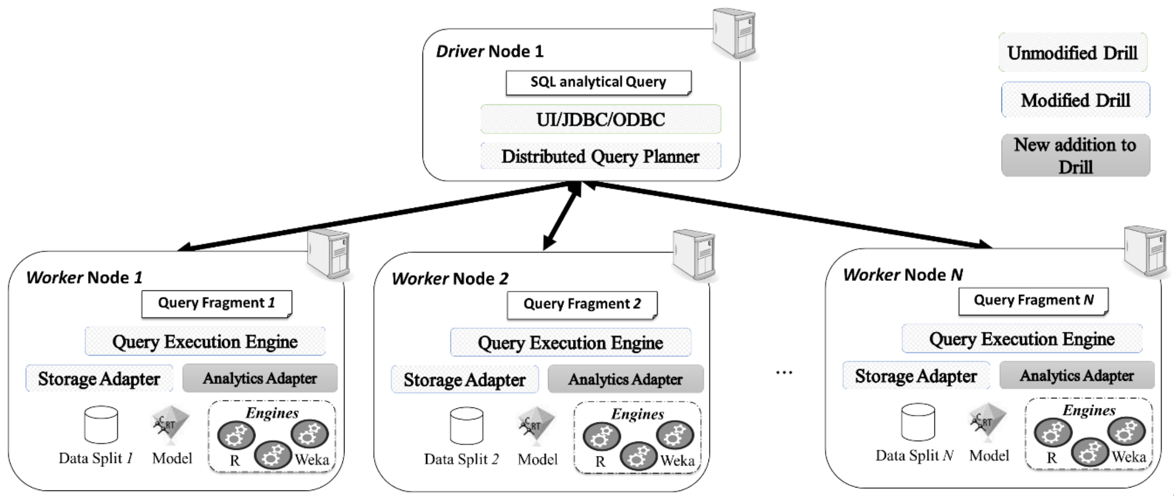

Figure 1. QDrill structure.

[16] Drill Storage: https://goo.gl/EFGTuo

The proposed *Analytics Adaptor* works in an analogous way as Drill's Storage Adaptor. It optimizes and provides access to various data mining libraries. The Analytics Adaptor works with *Analytics Plugins* that transform the data loaded by Drill to a data structure understandable by the data mining libraries. This way, algorithms from more than one library can be used together, leaving it to the Analytics Adaptor to resolve the inter-library data format conversion. In addition, the plugins invoke the APIs of the data mining library to train and score data mining models. All these details are hidden from users.

Back to the Consumable Analytics challenges, QDrill addresses the first challenge of providing distributed analytics by using the proposed Analytics Adaptor. It allows using any sequential single-node data mining library (e.g. WEKA) and make its algorithms run in a distributed fashion without having to rewrite them.

QDrill addresses the second challenge of providing an easy interface for in-house expertise to use by using the proposed Distributed Analytics Query Language. It allows invoking data mining algorithms from within the Drill standard SQL query statements. This allows in-house expertise to use the SQL language they are familiar with while having QDrill do the distributed deployment, data access and execution of the analytics jobs behind the scene.

For the third challenge of running the analytics on data of different formats stored on heterogeneous data stores, QDrill relies on Drill's Storage Adapter. The Storage Adapter allows accessing and joining data from heterogeneous sources and putting them in a common format to be processed by the analytics algorithms.

As a prototype, we extended *Apache Drill 1.2* as outlined above and created a plugin for the *weka-dev-3.7.13* data mining library in the Analytics Adapter in order to access the WEKA data mining algorithms using DAQL. The plugin also converts the data loaded by Drill to the ARFF format accepted by WEKA. In addition, we created a *Model Storage Plugin* that can store and load WEKA models from any data store supported by Drill. The full system architecture is illustrated in Fig. 1, showing the unmodified components of Drill, the modified components and the newly added ones.

A. *Model Training and Scoring*

Data mining models can be split to two types according to the data required to do the training. *Non-Updateable Models* require the full dataset available beforehand to do the training. *Updatable Models* are incremental models that can be trained using one instance (record) at a time. QDrill's Analytics Adaptor uses two training approaches, one for each model type.

- For Non-Updateable Models, Drill fetches the data in parallel, sends all data to the Driving Drillbit where the Analytics Adaptor trains the model on this single node. This approach does not speed up the training process as it is still being done on a single machine. However, it speeds up the data loading process. The limitation of this approach is that the dataset size is limited to the amount of available memory on the Driving Drillbit since a Non-Updatable Model will need to put all data in memory to do the training.
- For Updatable Models, Drill fetches the data for the Analytics Adaptor to process it in parallel. Each Drillbit trains an intermediate model, which are then aggregated at the Driving Drillbit to produce the final model. This approach speeds up both data fetching and model training as both operations are done in parallel on all of the available Drillbits. This approach is also not limited to the amount of available memory since only one record needs to be in memory at a time

The Analytics Adaptor uses the proposed Model Storage Plugin to store and distribute the trained model across all Drillbits to be used later for parallel scoring. In the scoring phase, the Model Storage Plugin on each Drillbit loads the trained model and the Analytics Adaptor feeds the loaded model with records from the data split available on that Drillbit, one record at a time. It brings the model to where data exists, unlike the traditional approach where data needs to be imported to the library. This approach speeds up the scoring process by distributing both data fetching and scoring. It is also highly flexible as it facilitates scoring on only a subset of the data and scoring records as they arrive, which is very beneficial for real-time scoring scenarios.

B. *Distributed Analytics Query Language (DAQL)*

The standard SQL syntax supported by Drill is extended in this work to provide the analytics functionality producing the *Distributed Analytics Query Language (DAQL)*. The DAQL adds a number of keywords and User Defined Functions (UDFs) to SQL.

Fig. 2 illustrates using the DAQL to train a WEKA model in a distributed fashion using the proposed Analytics Adaptor and the WEKA Analytics Plugin. The first SQL statement changes the storage location to a writable location. The second SQL statement tells the Drill Storage Adaptor to use the introduced Model Storage Plugin to save the model after training. The third SQL statement fetches the training data from any Drill-support data store using the `FROM` clause. The `FROM` clause can also have a join between two heterogeneous data sources. The `WHERE` clause specifies any conditions on the records to fetch. The SQL then uses the new `qdm_train_weka` UDF to define the classifier algorithm, set its arguments, specify the data columns to use for training and specify the label column, respectively. A question mark (`?`) can be used for the `<algorithm>` to display a list of supported algorithms and for the `<args>` to display a list of valid arguments for the selected algorithm. Finally, the statement uses the new `TRAIN MODEL` clause to save the trained model under `<model name>`.

```
SQL-1> USE dfs.tmp;
SQL-2> ALTER SESSION SET `store.format`='model';
SQL-3> TRAIN MODEL <model name> AS
       SELECT qdm_train_weka('<algorithm>','<args>',
                                       columns, label_column)
       FROM `<Data Source>`
       WHERE <conditions>;
```

Figure 2. Training a WEKA model using DAQL

Fig. 3 illustrates using DAQL to update an existing updatable model with new training records. The first SQL statement changes the storage location to a writable location. The second SQL statement tells the Drill Storage Adaptor to use the Model Storage Plugin. The third SQL statement fetches the new training dataset using the `FROM` clause. The new `APPLYING` keyword in the `FROM` clause tells Drill to fetch the trained model file `<old model name>`. The `WHERE` clause specifies any conditions on the records to fetch. The SQL then uses the new `qdm_update_weka` UDF to define the classifier algorithm, set its arguments, specify the model to update, specify the data columns to use for training and specify the label column, respectively. Finally, it uses the `TRAIN MODEL` clause to save the new trained model under `<model name>`.

```
SQL-1> USE dfs.tmp;
SQL-2> ALTER SESSION SET `store.format`='model';
SQL-3> TRAIN MODEL <model name> AS
          SELECT qdm_update_weka('<algorithm>','<args>',
                  mymodel.columns[0], mydata.columns,
                                         mydata.label_column)
          FROM `<Data Source>` AS mydata
                        APPLYING <old model name> AS mymodel
          WHERE <conditions>;
```

Figure 3. Updating an updatable WEKA model using DAQL

Fig. 4 illustrates using DAQL to score unlabeled data using a trained model. The first SQL statement changes the storage location to a writable location. The second SQL statement tells the Drill Storage Adaptor to save the scored records in CSV format. The third SQL statement fetches the unlabeled data using the `FROM` clause. The `APPLYING` keyword in the `FROM` clause tells Drill to fetch the trained model file `<model name>`. The `WHERE` clause specifies any conditions on the records to fetch. The SQL then uses the new `qdm_test_weka` UDF to apply the trained model on the unlabeled data. The UDF defines the classifier algorithm, set its arguments, specify the model to use for scoring and specify the data columns to use for scoring, respectively. This UDF outputs a label for each record in the unlabeled dataset. Finally, the SQL statement uses the `CREATE TABLE` clause to save the records along with their label in a new table `<results>`.

```
SQL-1> USE dfs.tmp;
SQL-2> ALTER SESSION SET `store.format`=csv';
SQL-3> CREATE TABLE <results> AS
          SELECT mydata.columns,
                    qdm_test_weka('<algorithm>','<args>',
                       mymodel.columns[0], mydata.columns)
          FROM `<Data Source>` AS mydata
                            APPLYING <model name> AS mymodel
          WHERE <conditions>;
```

Figure 4. Scoring a trained WEKA model using DAQL

The DAQL extends Drill's standard SQL to add analytics capabilities by supporting calls from within the SQL statements to any supported data mining library. By distributing the algorithms and handling data ETL, the DAQL allows users to do analytics in a clear and scalable way with less lines of code compared to scripting and programming languages.

V. EVALUATION

We carried out a set of experiments with our prototype to determine if QDrill offers a consumable analytics solution for Big Data scenarios. The experiments look at the impact of three important factors in Big Data scenarios on the performance of QDrill, namely the volume of data, the dimensionality of data and the horizontal scalability of the approach. In all the experiments we compare the performance of QDrill with that of *Apache Mahout 0.11* on *Hadoop 2.0*, whose algorithms are designed to be natively distributed.

The *Naïve Bayes* algorithm is used throughout this evaluation as it exists in both solutions. We compare two cases for training within QDrill. The first case uses the Updatable Naïve Bayes (*QDrill U*) where data fetching and training are distributed. The second case uses the regular Naïve Bayes (*QDrill NU*) where only data fetching is distributed and training is done sequentially in mini-batches on a single node. Scoring is the same for the two training cases. We compare all results with Mahout Distributed Naïve Bayes for Hadoop. Default configurations are used for all systems and algorithms throughout the evaluation. Evaluation is carried out in terms of execution time and storage overhead. We checked the accuracy of the resulting trained models for all three QDrill U, QDrill NU and Mahout and they all had nearly the same accuracy.

Amazon EC2 `T2.Large`[17] instances are used. They have two cores (3.3GHz each), 8GB RAM, 8GB Swap space and a 100GB Solid State Disk. Our cluster uses 10 of these nodes providing 20 cores, 80GB of RAM and 700GB of distributed HDFS storage space that is configured to have three replicas per block.

A. *Exp. 1: Data volume analysis*

Experiment 1 evaluates QDrill's performance, both in training and scoring, as more records are thrown at it. In this experiment, the number of columns per record is fixed to five columns (all integer) and the number of labels is fixed to two labels. Datasets of 100, 300, 600 and 1000 million records are used. The average of three runs is presented in each case.

For both training and scoring, most of Mahout's processing time is spent on loading the data into the vector format required for processing. This causes Mahout to have an expensive storage overhead and a longer processing time. Thus, for large datasets (100 million records plus) and small number of columns (5 columns), QDrill outperforms Mahout

[17] https://aws.amazon.com/ec2/instance-types/

both in training and scoring as illustrated in Fig. 5 and 6, respectively.

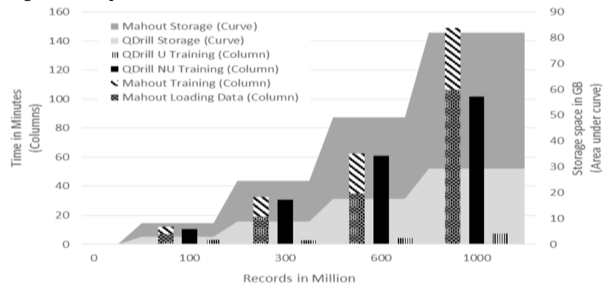

Figure 5. Effect of number of records on training

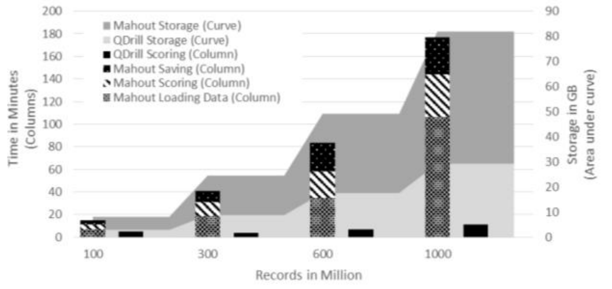

Figure 6. Effect of number of records on scoring

QDrill processes the data as-is, leaving it to the Analytics Plugins to do the data preparation on-the-fly as the data loads. This on-the-fly in-memory data format preparation saves time and eliminates QDrill storage overhead.

In the training phase (Fig. 5) and for the same number of records, QDrill Updatable Model training (*QDrill U*) consumes less time than QDrill Non-Updatable Model training (*QDrill NU*). Unlike *QDrill NU* where all training is done on a single node, *QDrill U* is fully parallelized where each Drillbit trains a local model and then all models are aggregated at the Driving Drillbit. From Fig. 5 we can see that QDrill's worst case performance presented by *QDrill NU* is still better than that of Mahout because QDrill uses in-memory data format conversion and processing, while Mahout uses the disk. That has been said, WEKA only supports a small handful of updatable models, making *QDrill NU* the most common scenario which yields the same performance as Mahout. We are currently investigating ways for speeding up QDrill NU while achieving the same model accuracy.

It worth mentioning here that we tried running *Mahout 0.11 on Spark* to include in this comparison. Even though, the 100 million record dataset size is 5.27GB and we have 80GB of RAM and 80 GB of Swap space, Mahout on Spark ran out of memory. This makes QDrill more suited for scenarios where fast results are required while having limited memory.

B. *Exp. 2: Data dimensionality analysis*

Experiment 2 evaluates QDrill's performance, both in training and scoring, as more columns are thrown at it. In this experiment, the number of records is fixed to 100 million records and the number of labels is fixed to two labels. Datasets of 5, 15, 25, 50 and 100 columns (all integer) are used.

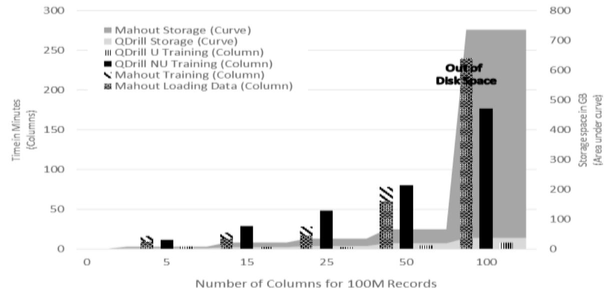

Figure 7. Effect of number of columns on training

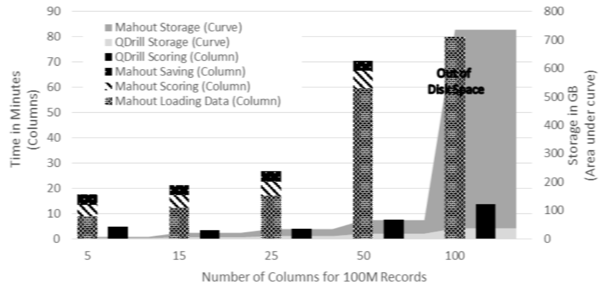

Figure 8. Effect of number of columns on scoring

For smaller numbers of columns Mahout outperforms *QDrill NU* during the training phase, as illustrated in Fig. 7, because a single node has to do all the training. However, as the number of columns increases, the dataset size increases along with Mahout's data loading and preparation time. This causes *QDrill NU* to perform better relative to Mahout as the dataset size becomes larger. Furthermore, *QDrill NU* gets the job done for any data size fitting in the existing storage as it does not have any storage overheads. Mahout fails as the data size increases (e.g the 36.51GB dataset of 100 million records with 100 columns per record) and there becomes not enough disk space to store the vector format required for processing.

The *QDrill U* training and QDrill scoring still vastly outperforms Mahout, as illustrated in Fig. 8. For both these operations, QDrill uses in-memory MPP to fetch and process the data in parallel.

C. *Exp. 3: Scalability analysis*

Experiment 3 evaluates QDrill's scalability, both in training and scoring. In this experiment, the number of records is fixed to 100 million records, the number of columns is fixed to 25 columns (all integer) and the number of labels is fixed to 2. Only the QDrill Updatable model training (*QDrill U*) is included in this evaluation as the QDrill Non-Updatable model training always runs on a single node and thus will not scale significantly. Evaluation is carried on 1, 2, 5 and 10 computing nodes. Each running a Drillbit and a MapReduce client.

From Fig. 9, adding a second node can lead to a 200% and 600% speed up for QDrill training and scoring, respectively. Moving from 5 nodes to 10 nodes yields 200%

speed up for both QDrill training and scoring. This shows how QDrill is flexible to scale to as many available nodes.

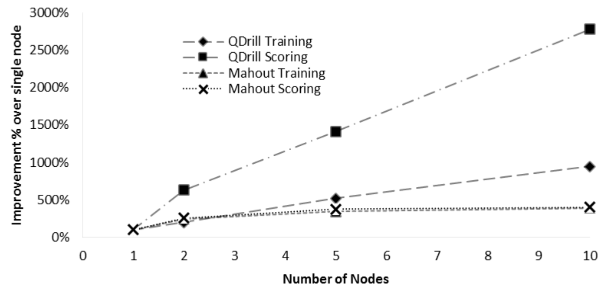

Figure 9. QDrill versus Mahout scaling

QDrill scoring shows better scalability than training. For scoring, everything is done in parallel as the trained model is distributed on all Drillbits and each Drillbit uses the model to score the data split locally available on the node. For training, adding more nodes creates more intermediate models. This speeds up the training as the data is split among a larger number of models being trained in parallel. However, this also slows down the models aggregation process that is still executed on a single node.

Mahout scaling, on the other hand, becomes stagnant as it reaches the default maximum configured number of Mappers and Reducers. Thus adding more nodes does not affect the execution time.

VI. CONCULSIONS AND FUTURE WORK

In this work, we present an approach based on SQL for distributed analytics on Big Data. The proposed approach addresses the three main challenges for providing Big Data consumable analytics. Addressing the first challenge, QDrill introduces the Analytics Adapter and Plugins enabling distributed model training and scoring for any sequential single-node data mining library without any algorithm rewrites. Addressing the second challenge, QDrill introduces the Distributed Analytics Query Language extension to Drill SQL to provide an easy-to-use interface familiar to database in-house experts so that they can run data mining algorithms in a distributed fashion without learning any new programming languages. Addressing the third challenge, QDrill handles putting the data in the format required by the data mining libraries and relies on Drill to access and join data from heterogeneous sources.

The empirical studies show that we achieve better performance than the natively distributed data mining Mahout library. This validates our hypotheses that we can achieve the same distributed performance without rewriting the algorithms. Using the proposed approach that allows reusing sequential single-node algorithms, we reduce the development time of distributed data mining libraries, allow access to more algorithms, provide an easier-to-use interface and increase the scalability.

Distributing the Non-Updatable models training using the proposed approach is still challenging and is the target of our future work. Our future work also includes adding more Analytics Plugins for popular libraries such as R and experimenting with other classes of data mining problems such as clustering.

The source code of the modified Apache Drill (QDrill) including the Analytics Adaptor and WEKA plugin is publicly available at https://svn.riouxsvn.com/qdrill/

ACKNOWLEDGMENT

This work is supported by IBM CAS research and NSERC.

REFERENCES

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Byers, "Big Data: The Next Frontier for Innovation, Competition, and Productivity," McKinsey Global Institute, 2011. Retrieved February 3rd, 2016 from http://bit.ly/McKinseyBigDataReport

[2] E. Sparks, A. Talwalkar, V. Smith, X. Pan, J. Gonzales, T. Kraska, M. Jordan, and M. Franklin, "MLI: An API for Distributed Machine Learning," In Proc. of the 2013 IEEE 13th Int'l Conf. on Data Mining (ICDM), 2013, pp.1187-1192.

[3] T. White, "Hadoop: The Definitive Guide (1st ed.)," O'Reilly Media, Inc., 2009.

[4] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," In Proc. of the 2nd USENIX Conf. on Hot topics in cloud computing (HotCloud'10), 2010.

[5] A. Koliopoulos, P. Yiapanis, F. Tekiner, G. Nenadic, and J. Keane, "A Parallel Distributed Weka Framework for Big Data Mining using Spark," In Proc. of 2015 IEEE International Congress on Big Data, 2015, pp.9-16.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," In Proc. of the 6th Conf. on Symposium on Operating Systems Design & Implementation - Volume 6 (OSDI'04), Vol. 6, 2004.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," In Proc. of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST '10), 2010, pp.1-10.

[8] D. Turner, M. Schroeck, and R. Shockley, "Analytics: The real-world use of big data in financial services," IBM Institute for Business, and Saïd Business School at the University of Oxford, 2012. Retrieved February 3rd, 2016 from http://goo.gl/T6hD4k

[9] IBM, "IBM Global Technology Outlook 2012," IBM, 2012. Retrieved February 3rd, 2016 from http://goo.gl/63zjGl

[10] S. Melnik, A. Gubarev, J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," In Proc. VLDB Endow.3, 1-2, 2010, pp.330-339.

[11] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane, "DMQL: A data mining query language for relational databases," In Proc. of SiGMOD'96, 1996, pp.27-34.

[12] R. Ortale, E. Ritacco, N. Pelekis, R. Trasarti, G. Costa, F. Giannotti, G. Manco, C. Renso, and Y. Theodoridis, "The DAEDALUS framework: progressive querying and mining of movement data," In Proc. of the 16th ACM SIGSPATIAL Int'l Conf. on Advances in geographic information systems (GIS '08), Article 52, 2008, 4 pages.

[13] T. Johnson, L. Lakshmanan, and R. Ng, "The 3W Model and Algebra for Unified Data Mining," In Proc. of the 26th Int'l Conf. on Very Large Data Bases (VLDB '00). 2000, pp.21-32.

[14] Z. Tang, J. Maclennan, and P. Kim, "Building data mining solutions with OLE DB for DM and XML for analysis," SIGMOD Rec. 34, 2, 2005, pp.80-85.

[15] M. Yui and I. Kojima "A Database-Hadoop Hybrid Approach to Scalable Machine Learning," In Proc. of the 2013 IEEE Int'l Congress on Big Data (BIGDATACONGRESS '13), 2013, pp.1-8.

[16] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," In Proc. VLDB Endow. 2, 2, 2009, pp.1626-1629.

[17] R. Meo, G. Psaila, and S. Ceri, "A New SQL-like Operator for Mining Association Rules," In Proc. of the 22th Int'l Conf. on Very Large Data Bases (VLDB '96), 1996, pp.122-133.

[18] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi, "A Sequential Pattern Query Language for Supporting Instant Data Mining for e-Services," In Proc. of the 27th Int'l Conf. on Very Large Data Bases (VLDB '01), 2001, pp.653-656.

[19] R. Xin, J. Rosen, M. Zaharia, M. Franklin, S. Shenker, and I. Stoica, "Shark: SQL and rich analytics at scale," In Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD '13), 2013, pp.13-24.

[20] J. Zhou, N. Bruno, M. Wu, P. Larson, R. Chaiken, and D. Shakib, "SCOPE: parallel databases meet MapReduce," The VLDB Journal 21, 0035, 2012, pp.611-636.

[21] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," SIGOPS Oper. Syst. Rev. 41, 3 (March 2007), 2007, pp.59-72.