Big Data Analytics Integrating a Parallel Columnar DBMS and the R Language

Yiqun Zhang, Carlos Ordonez, Wellington Cabrera
University of Houston, TX, USA

***Abstract*—Most research has proposed scalable and parallel analytic algorithms that work outside a DBMS. On the other hand, R has become a very popular system to perform machine learning analysis, but it is limited by main memory and single-threaded processing. Recently, novel columnar DBMSs have shown to provide orders of magnitude improvement in SQL query processing speed, preserving the parallel speedup of row-based parallel DBMSs. With that motivation in mind, we present COLUMNAR, a system integrating a parallel columnar DBMS and R, that can directly compute models on large data sets stored as relational tables. Our algorithms are based on a combination of SQL queries, user-defined functions (UDFs) and R calls, where SQL queries and UDFs compute data set summaries that are sent to R to compute models in RAM. Since our hybrid algorithms exploit the DBMS for the most demanding computations involving the data set, they show linear scalability and are highly parallel. Our algorithms generally require one pass on the data set or a few passes otherwise (i.e. fewer passes than traditional methods). Our system can analyze data sets faster than R even when they fit in RAM and it also eliminates memory limitations in R when data sets exceed RAM size. On the other hand, it is an order of magnitude faster than Spark (a prominent Hadoop system) and a traditional row-based DBMS.**

I. Introduction

There is significant research progress on efficient algorithms to analyze large data sets, but most of them work outside a DBMS on flat files. Currently, R package is one of the most popular open-source systems to perform statistical analysis due to its extensive library of models and techniques, intuitive syntax and interpreted language (i.e. interactive). Unfortunately, as well noted in the literature, R is slow to analyze large data sets, especially when they do not fit in RAM. On the other hand, DBMSs and the Hadoop eco-system are currently the two competing technologies to analyze big data, both based on automatic data parallelism on a shared-nothing architecture. Nevertheless, previous research [3] has shown it is slow to move large volumes of data outside a DBMS due to double I/O, the need to change blocks to an external file format and network latency. Moreover, the export operation needs to be repeated if the database is refreshed with new records or data sets are rebuilt with new variables (dimensions). On the other hand, traditional DBMSs are highly efficient for SQL query processing, but SQL is considered slow and inadequate for matrix manipulation. Reasons supporting such perception include the following: There is more flexibility to develop algorithms in a traditional programming language like C++; relational tables and arrays are incompatible with each other; the complex parallel DBMS internal architecture. Therefore, the problem of performing statistical analysis with SQL has received scant attention. User-defined functions are a powerful programming mechanism that can extend SQL math capabilities, but at the cost of leaving optimization and memory management to the developer.

Given its wide adoption and standards, SQL will likely remain the language of DBMSs. Research in the last decade has proposed specialized DBMSs to improve SQL performance. Specifically, column stores [1], [5] provide orders of magnitude of acceleration for analytical SQL queries combining joins and aggregations. With that motivation in mind, we present COLUMNAR, an analytic system that can work on top of a columnar DBMS evaluating optimized SQL queries and user-defined functions (UDFs) for such architecture. The main contribution of our research is that key matrix equations involving large matrices are computed with SQL queries and UDFs instead of a traditional programming language. In this manner, we have been able to integrate methods without disrupting the DBMS architecture and without the need to modify the DBMS internal source code. Our algorithms can process a large data set in either one pass or a few passes. Moreover, all our algorithms have linear scalability on data set size and they exhibit linear speedup.

COLUMNAR has unique features and advantages compared to other analytic systems. Unlike most research prototypes and existing tools, our system directly processes relational tables, truly performing "in-database" analytics. Our system can analyze large data sets faster than most external data mining tools, including the R package and Hadoop systems (including Spark, a prominent system that has subsumed MapReduce). From an R perspective, our system eliminates the RAM limitations in R. Even though there exist R libraries, like bigmem and snow, to remove the RAM limitations, they still cannot work in parallel. From a parallel processing perspective, our system is much faster than Hadoop systems (like Spark) to analyze large tables running on the same hardware. In other words, it requires fewer nodes in a parallel cluster. There exist other systems integrating R and a DBMS (Vertica, SQL Server etc.) or R and Hadoop (IBM Ricardo, Teradata Aster), but in general they require transferring data in RAM back and forth between the DBMS and R.

Our system has promise of wide application considering DBMSs with ACID properties and relational query capabilities remain the standard data management platform and column-based DBMSs gain wider adoption. As time goes by most data warehouses will be supported by a columnar DBMS

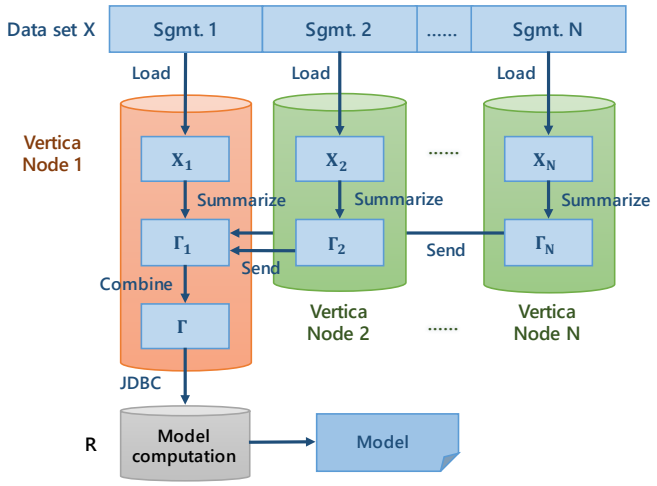

Fig. 1. System architecture.

behind to process ad-hoc queries. This scenario will lead to asking "why" questions once interesting trends and facts are discovered. Therefore, there will be many data sets stored in the DBMS that will require analysis with machine learning models, where the R language or similar statistical systems come into play. The user will benefit from analyzing such data sets inside the DBMS: avoiding export bottlenecks, enforcing secure access and eliminating redundant copies of data in external computers.

II. Definitions

The input data set $X = \{x_1, x_2, \ldots, x_n\}$ has n records (or points) and d attributes (dimensions, features) plus an optional "supervised" attribute (class G or dependent variable Y). That is, n is the data set size and when all attributes are numeric, d represents dimensionality.

At a basic level, we compute descriptive statistics on the data set X, which include the global mean μ, the variance-covariance matrix V (also called Σ) and the correlation matrix ρ. These descriptive statistics can be computed on subsets of X as well (e.g. filtering X with some selection predicate). From a statistical model perspective, our system offers two kinds of models: supervised and unsupervised models. In supervised models, as noted above, X has an additional "target" attribute: a numeric dimension Y (for regression) or a discrete (categorical) "class" attribute G for classification. Unsupervised models include PCA (solved with SVD), and clustering with mixtures of distributions (solved with K-means). On the other hand, supervised (predictive) models include: linear regression (solved by least squares) and the Naïve Bayes classifier (estimating joint class probability as a product of marginal probabilities).

III. System Overview

A. System Architecture

Our system is based on a client-server architecture between R and the DBMS, shown in Figure 1. The DBMS is a parallel cluster with N processing nodes under a shared-nothing architecture (no shared RAM or disk). The R package commonly runs on a separate machine or on one of the N nodes (e.g. the master node), under the assumption that any computation involving n rows is always evaluated with SQL (possibly calling a UDF). Numerically intensive matrix factorizations to solve SVD and least squares are solved by R, which calls the LAPACK library. The data set, model matrices and model parameters, are all stored as relational tables. An R program at the client computer generates optimized SQL queries, connects to the DBMS (via JDBC)submits dynamically generated SQL queries (possibly calling UDFs) and at the end of DBMS processing, it retrieves results and loads them into R data structures (matrices, data frames). R is responsible for two main tasks: (1) computing the model exploiting data summarization computed by the DBMS (train phase); (2) testing the model by applying the model to produce the "estimated" attribute (test phase). The iterative method convergence and stopping are managed by an R program.

B. Big Data Analytics with SQL Queries and R

We start by discussing storage in the columnar DBMS, where indexing is not required. The data set and model matrices are stored in tables, having a primary key with a specific combination of matrix subscripts, depending on their storage layout. Our system uses two fundamental data set layouts for X: horizontal and vertical. The first layout is a standard tabular representation which has n rows, each having d columns. The second layout is based on a pivoted table with one attribute value per row and up to dn rows, which enables efficient processing of sparse matrices and which also removes DBMS limitations on the maximum number of columns. The I/O efficiency of each layout depends on matrix sparsity and the specific query physical operator (scan, join, sort). The data set will be sorted by the matrix subscripts, and will be segmented and distributed to all the N processing nodes evenly according to the value of a hash function on the row number i. This process will be automatic in the columnar DBMS during the data loading phase once we properly define the table at creation time.

Computing matrix equations with SQL queries has proven to be hard to optimize [2]. To get started, it is not possible to create fixed queries that can efficiently evaluate mathematical equations because the table definition for the input data set is not fixed (especially for the horizontal layout with d columns). Second, evaluating matrix equations with pure SQL queries usually involves table joins, which are always quite expensive for DBMSs especially when they are evaluated in parallel. Therefore, SQL code generation at run-time from R and developing specialized mathematical UDFs to bypass table joins are necessary. Statistical methods and data mining algorithms are also programmed with R in order to take full advantage of its rich library of models and techniques.

The R program is based on two sets of parameters. The first set of parameters controls SQL code generation (i.e. table name and layout etc.) and database systems optimizations,

which are tailored to each algorithm (with some essential math optimizations across multiple models). The second set of parameters controls the mathematical behavior of each technique. Examples include the number of clusters, tolerance threshold for convergence, numeric stability issues, and so on. We must emphasize that SQL code generated by R represents a Turing-complete language. That is, we can evaluate any equation with matrices, no matter how complex it is.

C. Algorithmic Optimizations

We discuss algorithmic optimizations, which are general and therefore can be applied on any programming languages like C++ or Java (i.e. beyond SQL). The sufficient statistics of the multivariate normal distribution are an essential ingredient to accelerate data mining computations [3]. Our system makes extensive use of sufficient statistics to reduce the number of passes on the data set, reaching one pass in most cases. Our sufficient statistics are basically: the number of data records n, the linear sum of points L (a d-dimensional vector) and the quadratic (with cross-products) sum of points Q (a $d \times d$ matrix) [2], [3]: $L = \sum_{i=1}^{n} x_i$, $Q = \sum_{i=1}^{n} x_i \cdot x_i^T$. In order to reduce data set summarization to a a single matrix multiplication, we build an extended vector z_i in RAM as $z_i = [1, x_i, y_i]$. Then we compute a fundamental summarization matrix $\Gamma = Z \cdot Z^T$ [4]. Therefore, Γ becomes a comprehensive summary of X containing L, Q as sub-matrices. Then Γ is computed by a single SQL query using self-joins or calling a user-defined transform function that builds vector z_i and updates Γ in parallel and incrementally. Γ is then used to substitute X. The global mean and covariance are $\mu = L/n$ and $V = Q/n - (L/n)(L/n)^T$. Correlations ρ_{ab} are efficiently computed with $\rho_{ab} = (nQ_{ab} - L_a L_b)/(\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2})$. To compute the global mean, global covariance, PCA, Naïve Bayes and linear regression sufficient statistics enable computation of the model in one table scan. For PCA the correlation matrix ρ is passed to the SVD function in R (which calls LAPACK). Linear regression is solved using $\beta = Q^{-1}XY^T$ calling R as well (which also calls LAPACK). Sufficient statistics are independently computed on k subsets of the data set for K-means clustering, where the cluster means C_j and cluster variances are efficiently computed with queries on small tables. For K-means clustering, we developed an incremental algorithm that obtains an approximate, but highly accurate, solution in a few iterations, being much faster than the standard K-means algorithm.

D. Query Processing Optimizations

From a query processing perspective, we carefully analyzed each query plan, identifying key optimizations for a columnar architecture. The following discussion is mainly about computing data set summarization to later compute the model in R (i.e. train phase). An important distinction with cube queries is that statistical analysis generally requires all columns from the base table, resulting in a so-called super-projection [1]. We should emphasize that despite the fact that there exist similarities between row and column DBMS, there are many important, sometimes subtle, differences in query optimization between both. A significant difference with previous research based on row storage [2] is that there are no indexes: column values are internally maintained sorted at all times. Columnar DBMSs have two internal storage managers: a Write-Optimized Storage (WOS) to enable fast loading and a Read-Optimized Storage (ROS) for query processing. There is an incremental conversion of new records between the WOS and the ROS. We will show the WOS to ROS conversion is a bottleneck to load large data sets. However, since the WOS runs concurrently as a background process the DBMS can incrementally consume batches of new records. In this manner, the DBMS can incrementally analyze high-velocity data. We now explain query optimization queries on the ROS. Since equations are evaluated with SQL queries and UDFs, there are temporary tables and we control how each temporary table is stored, by choosing an appropriate layout for the data set, which improves compression and provides ordering of column values for faster join and partition processing in UDFs. Query rewriting is essential. A join operation is the main operation that can degrade performance, if not carefully managed. Denormalization is used to compute sufficient statistics together, eliminating joins in a query. Alternatively, a user-defined transform function pushes the computation of the summarization matrix Γ to RAM. The second major relational operator is a GROUP BY aggregation, which in general requires sorting by a different set of columns from the projection. Whenever possible, aggregations are pushed before joins. There are two choices for join algorithms: merge joins and hash joins. Notice joins are not sort-merge joins since they skip a sorting operation. When two tables have the same primary key they are already sorted. Therefore, we can use merge joins, resulting in linear performance. On the other hand, if a merge join is not possible, when one table is small, whereas as the other table is large, we exploit hash joins. If both of these cases fail then we define a projection on the join column to enable a merge join. In short, unlike previous research [2], merge joins are preferred over hash joins and nested loop joins are out of consideration. To conclude we briefly discuss the model test phase, which is efficiently done by sending data blocks from the DBMS to R to compute the "estimated" attribute in a new column and then sending such column back to the DBMS. Such block-based algorithm is efficiently implemented via a UDF that build R data frames by block and calls R on each block: data transfer back and forth happens exclusively in RAM.

IV. Experimental Validation

We present experiments to evaluate the performance of our system compared to Spark and R. Since our algorithms do not change the accuracy of matrices and final results, it is unnecessary to measure statistical or numeric accuracy.

A. Experimental Setup

We ran all the systems on a machine that has a Quad Core CPU, 4GB of RAM and 1TB SATA disk running Ubuntu

Server 14.04. The columnar DBMS was HP Vertica 7.2. We ran only one parallel system at a time, while the rest were shut down.

The data set we used was the network intrusion data set obtained from the KDD Cup repository (KDD 99). The original KDDnet data set has $n = 10M$ data records and $d = 38$ attributes. We sampled and replicated the KDDnet to get varying n (data set size) and d (dimensionality), without altering its statistical properties. All the data sets were available as CSV files on disk for R, on HDFS for Spark, and as relational tables in a vertical layout (i, j, v) for HP Vertica. In Vertica, the relational tables for the data sets were ordered by the matrix subscripts i, j, and were segmented across the nodes by $modularhash(i)$, a hash function that helps distribute the data evenly. The ordering and segmentation were specified in the table definition, and the data was segmented and sorted at loading time (i.e. No other separate projection is created).

B. Performance Comparison

We now compare the performance of our system computing the data set summarization against SQL queries with joins, Spark and R. Since R cannot work in parallel, in this comparison we only use $N = 1$ (number of the processing nodes) and vary n and d.

Table I shows the result of the comparison. If the computation still cannot finish after 20 minutes, we report "stop". We only compared the time to compute the summaries of the data set Γ because Γ can be consumed by R and the time to compute PCA or LR is 1 second for $d \leq 100$. From the result we can see the approach using SQL queries calling UDFs is the fastest for all data sets. In particular, it is one order of magnitude faster than SQL with joins because our UDF can bypass the expensive self-join. Meanwhile, our UDF approach is much faster than R and Spark. The table showed that when n grows from 100K to 1M, the time for R becomes more than 10 times slower. This indicates that the scalability of R degrades a lot due to the limit of the RAM as the size of the data set continue to grow. The comparison between our UDF and Spark shows that although Spark scales well as the data set gets larger, it cannot get as efficient as our UDF due to the CSV parsing and JVM overhead.

TABLE I
PERFORMANCE COMPARISON: COLUMNAR VS. SQL QUERIES WITH JOINS, R AND SPARK (TIME IN SECS.)

		COLUMNAR			
n	d	UDF	join query	R	Spark
10K	10	0.05	0.29	0.11	0.27
	20	0.08	0.74	0.23	0.31
	40	0.14	1.86	0.49	0.63
	80	0.25	5.45	0.98	1.10
100K	10	0.33	3.48	1.16	1.36
	20	0.52	7.64	2.38	2.70
	40	1.03	15.59	5.04	4.82
	80	1.97	40.86	10.75	10.00
1M	10	2.8	28.31	16.46	11.30
	20	4.68	63.86	29.65	23.00
	40	9.54	163.17	61.78	47.47
	80	22.36	stop	273.86	95.69

REFERENCES

[1] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandier, L. Doshi, and C. Bear. The Vertica analytic database: C-store 7 years later. *PVLDB*, 5(12):1790–1801, 2012.

[2] C. Ordonez. Integrating K-means clustering with a relational DBMS using SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(2):188–201, 2006.

[3] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

[4] C. Ordonez, Y. Zhang, and W. Cabrera. The Gamma operator for big data summarization on an array DBMS. *Journal of Machine Learning Research (JMLR): Workshop and Conference Proceedings (BigMine 2014)*, (36):61–96, 2014.

[5] M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E.J. O'Neil, P.E. O'Neil, A. Rasin, N. Tran, and S.B. Zdonik. C-Store: A column-oriented DBMS. In *Proc. VLDB Conference*, pages 553–564, 2005.