Can We Analyze Big Data inside a DBMS?

Carlos Ordonez
University of Houston
Houston, TX 77204, USA

ABSTRACT

Relational DBMSs remain the main data management technology, despite the big data analytics and no-SQL waves. On the other hand, for data analytics in a broad sense, there are plenty of non-DBMS tools including statistical languages, matrix packages, generic data mining programs and large-scale parallel systems, being the main technology for big data analytics. Such large-scale systems are mostly based on the Hadoop distributed file system and MapReduce. Thus it would seem a DBMS is not a good technology to analyze big data, going beyond SQL queries, acting just as a reliable and fast data repository. In this survey, we argue that is not the case, explaining important research that has enabled analytics on large databases inside a DBMS. However, we also argue DBMSs cannot compete with parallel systems like MapReduce to analyze web-scale text data. Therefore, each technology will keep influencing each other. We conclude with a proposal of long-term research issues, considering the "big data analytics" trend.

1. INTRODUCTION

DBMS are capable of efficiently updating, storing and querying large tables. Relational DBMSs have been the dominating technology to manage and query structured data. Nowadays there is an explosion of data on the Internet (big data) represented by web pages, documents, diverse files and so on. In both worlds the main challenge is to efficiently analyze large data sets. Research on analyzing large data sets is extensive, and has proposed many efficient algorithms, data structures, and optimizations to analyze large data sets. Data mining is concerned with knowledge discovery, represented by models and patterns in large databases. The position of this paper is that, despite existing progress, it is necessary to study the integration of statistical methods and matrix computations with database systems in more depth. We present a survey of system infrastructure, programming mechanisms, algorithms and optimizations that enable analytics on large data sets inside a DBMS, avoiding exporting data outside.

There is significant progress on efficient data mining algorithms, but most of them work outside a DBMS on flat files. Previous research has shown it is slow to move large volumes of data due to double I/O, change of file format and network speed [24]. Therefore, exporting data sets is a bottleneck to analyze large data sets outside a DBMS [24]. The data set storage layout has a close relationship to algorithm performance. Reducing the number of passes over a large data set resident on secondary storage, developing incremental algorithms with faster convergence, sampling for approximation and creating efficient data structures for indexing or summarization remain fundamental algorithmic optimizations. DBMSs [18] and MapReduce [35] are currently two competing technologies for data mining on large data sets, both based on automatic data parallelism on a shared-nothing architecture. DBMSs can efficiently process transactions and SQL queries on relational tables. There exist parallel DBMSs for data warehousing and large-scale analytic applications, but they are difficult to expand and they are expensive. On the other hand, MapReduce automatically distributes analytic operations across a cluster of computers and it can work on top of a distributed file system, providing great flexibility, efficiency, expandability and low cost. Nevertheless, we will explain why there are many benefits for the end-user when performing data mining inside a DBMS: querying, reducing data inconsistency, security, fault-tolerance, but mostly avoiding the export bottleneck if data originates from an OLTP database or a data warehouse.

The problem of integrating statistical and machine learning methods and models with a DBMS has received moderate attention [18, 4, 25, 32, 33, 24]. In general, such problem is considered difficult due to a relational DBMS architecture, the matrix-oriented nature of statistical techniques, lack of access to the DBMS source code, and the comprehensive set of techniques already available in mathematical languages like Matlab. statistical languages like R, and numerical libraries like LAPACK. As a consequence, in modern database environments, users generally export data sets to a statistical or parallel system, iteratively build several models outside the DBMS, and finally deploy the best model. In practice, exploratory cube analysis does most of query processing inside the DBMS, but the computation of statistical models (e.g. regression, PCA, decision trees, Bayesian classifiers, neural nets) and pattern search (e.g. association rules) is more commonly performed outside, despite the fact that DBMSs indeed offer some data mining algorithms. The importance of pushing statistical and data mining computations into a DBMS is recognized in [8]; this work emphasizes exporting large tables outside the DBMS is a bottleneck and it identifies SQL queries and MapReduce [30] as two complementary mechanisms to analyze large data sets.

http://dx.doi.org/10.1145/2513190.2513198

2. PRELIMINARIES

2.1 Definitions

The goal is to analyze a data set X with machine learning and statistical techniques to compute a model or find patterns (in the case of association rules). In general, X has n records and d attributes (numeric or categorical). In many cases, all d attributes are numeric and then X becomes a set of n points in d-dimensional space. For predictive models, X has an additional "target" attribute. This attribute can be a numeric dimension Y (used for regression) or a categorical discrete attribute G (used for classification). In the case of cubes, the data set is a fact table F with d discrete attributes (called dimensions) to aggregate by group and m numeric attributes called measures.

Machine learning and statistical techniques make extensive use of vectors and matrices. As mentioned above, n is the total number of records (observations, points), d is the total number of dimensions (attributes, features) and k is a common additional size specified in models (e.g. k clusters, k components). Our system uses the following conventions for subscripts: $i = 1 \ldots n$, $h = 1 \ldots d$, $j = 1 \ldots k$.

In the relational DBMS the data set X is stored on a table with an additional column i identifying the point (e.g. a customer id), which is not used for statistical purposes. This leads to table X being defined as $X(\underline{i}, X_1, X_2, \ldots, X_d)$ with primary key i. When there is a predicted numeric dimension Y, X is defined as $X(\underline{i}, X_1, X_2, \ldots, X_d, Y)$ and when there is a categorical attribute G, X is defined as $X(\underline{i}, X_1, X_2, \ldots, X_d, G)$. Statistical models are stored in tables as well. On the other hand, the fact table has the generic definition $F(i, D_1, D_2, \ldots, D_d, A_1, A_2, \ldots, A_m)$.

2.2 Big Data Analytics Overview

We now give an overview of the entire analytic process going from data preparation to actually deploying and managing models. Each task is explained below.

In general, the first task is to process, transform and clean several tables to build a data set for analysis. Data transformation involves denormalization (pivoting [9], horizontal aggregations [27, 22]), binning (discretizing), sampling [7, 6], rescaling and null value imputation, among other transformations. The primary goal of data transformation is to create clean data sets that can be used as input for a data mining algorithm.

After a data set has been created, the next task is to explore its statistical properties. Exploratory analysis includes ad-hoc queries (i.e., each query leading to a further query), data quality assessment [11, 26] (i.e. to detect and solve data inconsistency and missing information), OLAP cubes, and spreadsheets [38] (which provide an interactive environment to define formulas among data elements). In general, no models are computed. Thus exploratory analysis reveals the "what", "where" and "when", but not "why".

The last task is to compute statistical models or discover patterns hidden in the data set. Statistical and machine learning models include both descriptive and predictive models such as regression, clustering classification, PCA, statistical tests, time series and so on. On the other hand, pattern search is represented by cubes [15], association rules [2], and their generalizations, like sequences and graphs. Finally, model management is a subtask of model computation, which includes scoring (model deployment, inference [17]), model exchange (with other tools) and versioning (i.e., a history of models).

3. SYSTEM INFRASTRUCTURE

3.1 Storage and Indexing

Most systems use three fundamental data set storage layouts, which influence I/O efficiency:

- Row storage: This is the most common storage layout, in which X is stored on a tabular format which has n rows, each having d columns. Several rows are generally stored together in a single block.
- Column storage: This is a novel layout tailored to analytical query processing [36]. In this case the data set is stored by column across several blocks, which enables efficient projection of subsets of columns. There is separate storage for the d columns, each having the primary key (or address) of each record and the corresponding column value. In the transaction layout, for efficiency purposes, only a subset of dimensions for each record is actually stored (e.g. excluding binary dimensions equal to zero).
- Key-value storage: This layout is the so-called transaction data set, stored on a normalized table with one attribute value per row and up to dn rows.

Three row storage layouts stand out: a standard horizontal layout having one row per input point, a long pivoted table with one attribute value per row or a transposed table having records as columns. Each layout provides tradeoffs among I/O speed, space and programming flexibility, depending on the required mathematical computation, data set dimensionality and the existence of null/zero values. Column stores provide limited flexibility in layouts. On the other hand, a vertical row layout is the most common for key-value stores like MapReduce [35], but column stores are being explored.

Storage in DBMSs is built around blocks of rows or columns of simple data types (i.e. numbers and strings) with similar size. On the other hand, key-value stores are designed to store strings of highly variable size. There has been a surge on array database systems, like SciDB [5] and ArrayStore [34], but with different storage and query evaluation mechanisms from SQL. Such systems are making some impact on the academic world, complementing Matlab and R, but their future impact is uncertain for corporate databases.

3.2 Parallel Processing

Currently, the best parallel architecture for DBMSs and cloud systems is shared-nothing, having N processing units. These processing units can be N threads on a single CPU server (including multicore CPUs) or N physical nodes (each one with its own RAM and secondary storage). In row stores tables are horizontally partitioned into sets of rows for parallel processing. Thus there exist sequential and parallel versions of database physical operators: table scan, external merge sort and joins. Similarly, in column stores columns are partitioned into blocks, which are in turn distributed among processing units. MapReduce [12, 35] trails behind DBMSs to evaluate such parallel physical operators on equivalent hardware, although the gap gradually shrinks with RAM-based processing [40].

3.3 Programming Alternatives

We distinguish three solutions to program data mining algorithms to analyze large data sets stored on a DBMS: (1) with SQL queries and UDFs without modifying the DBMS source code; (2) with an efficient systems language like C++ extending internal DBMS source code. (3) with programming languages in external systems, including parallel systems. The first alternative allows processing data "in-situ", avoiding exports outside the DBMS. The second one is feasible for open-source DBMSs and developers of industrial DBMSs. The last alternative is the most flexible, allowing many customized optimizations, has shortest development time and therefore the most popular.

Performing data mining with SQL queries and UDFs has received moderate attention. A major SQL limitation is its lack of ability to manipulate non-tabular data structures requiring pointer manipulation. When data mining computations are evaluated with SQL queries a program automatically creates queries combining joins, aggregations and mathematical expressions. The methods to generate SQL code are based on the input data set, the algorithm parameters and tuning query optimizations. SQL code generation exploits relational query optimizations including forcing hash-joins [25], clustered storage for aggregation [25, 24], denormalization to avoid joins and reduce I/O cost [23], pushing aggregation before joins to compress tables and balancing row distribution among processing units. This alternative covers the scenario where most processing on large tables is done inside the DBMS, but partial processing is done on small tables exported outside the DBMS [32] with an external mathematical library or statistical program. Given DBMS overhead and SQL lack of flexibility to develop algorithmic optimizations SQL is generally slower than a systems language like C++. But as hardware accelerates and memory grows such gap in speed becomes less important. SQL has been enriched with object-relational extensions like User-Defined Functions (UDFs), stored procedures and User-Defined Types, with aggregate UDFs being the most powerful for data mining. SQL extensibility mechanisms include User-Defined Functions (UDFs), stored procedures, and table-valued functions UDFs represent a compromise between modifying the DBMS source code, extending SQL with new syntax and generating SQL code. UDFs represent an important extensibility feature, now widely available (although not standardized) in most DBMSs. UDFs include scalar, aggregate and table functions with aggregate UDFs being the most powerful, providing a simplified parallel programming interface. UDFs can take full advantage of the DBMS processing power when running in the same address space as the query optimizer (i.e. unfenced [32] or unprotected mode). UDFs do not increase the computation power of SQL, but they make programming easier and many optimizations feasible since they are basically C code fragments, which allow pushing more mathematical processing into main memory. However, query optimization techniques need to be adapted because UDFs work in main memory, parallel execution needs customized optimizations UDFs are fed by table scan operators, UDFs cannot call relational operators and cost functions are different. Data structures, incremental computations and matrix operations are certainly difficult to program with SQL and UDFs due to the relational DBMS architecture, limited flexibility of the SQL language and DBMS subsystems overhead, but they can be optimized and they can achieve a tight integration with the DBMS. SQL can achieve linear scalability on data set size and dimensionality to compute several models. It has been shown that SQL is somewhat slower (not an order of magnitude slower) than C++ (or similar language) [23], with relative performance getting better as data set size increases (because overhead decreases). There is research on adapting and optimizing numerical methods to work with large disk-resident matrices [29, 39], but only a few work in the DBMS realm [24, 18].

Developing fast algorithms in C++ or Java was the most popular alternative in the past. This was no coincidence as a high-level programming language provides freedom to manipulate RAM and minimize I/O. Modifying and extending the DBMS C (or C++) source code (open source code or proprietary), yields algorithms tightly coupled to the DBMS. This alternative has been the most common for commercial DBMSs and it includes extending SQL with new syntax. This alternative includes low-level APIs to directly access tables through the DBMS file system (cursors [32], blocks of records with OLEDB [21]). This alternative provides fast processing, but also has disadvantages. Given the DBMS complex architecture and lack of access to DBMS source code it is difficult for end users and researchers to incorporate new algorithms or change existing ones. Nowadays, this alternative is not popular for parallel processing.

The most popular alternative to process large data sets in parallel outside a DBMS is currently MapReduce [35], helped by the fact that the Hadoop file system is open-source and robust. For the most part a cloud computing model is the norm, where the cloud offers hardware and software as service. Even though some DBMSs have achieved some level of integration between SQL and MapReduce (e.g., Greenplum [18], HadoopDB [1], Teradata AsterData [14]), in most cases data must be moved from the DBMS storage to the MapReduce file system. Some recent systems that have enabled SQL capabilities on top of the distributed Hadoop file system, integrating some information retrieval capabilities into SQL, include Hive [40], ODYS [37] and AsterixDB [3]. In statistics and numerical analysis the norm is to develop program in non-systems languages like R, Matlab, SAS and Fortran, which do not scale to large data sets. Thus analysts generally split computations between such packages and MapReduce [10], pushing demanding computations with heavy I/O on large data sets into MapReduce. In general, the integration of mathematical packages like the R language, Matlab or scientific computing programming languages remains limited. The RIOT [41] system introduces a middleware solution to provide transparent I/O in the R statistical package with large matrices (i.e. the user does not need to modify R programs to analyze large matrices). RIOT does address the scalability problem, but outside the DBMS. Unfortunately, RIOT cannot efficiently analyze data inside a DBMS.

3.4 Algorithm Design and Optimizations

Efficient algorithms are central on database systems. Their efficiency depends on the data set layout, iterative behavior, numerical method convergence and how much primary storage (RAM memory) is available. There are two major goals: decreasing I/O and accelerating convergence. Most statistical algorithms have iterative behavior requiring one or more passes over the data set per iteration [24]. The

most common approach to accelerate an algorithm is to reduce the number of passes scanning the large data set, without decreasing (or even increasing) model quality (e.g. log-likelihood, squared error and node purity. The rationale is to decrease I/O cost, but also to reduce CPU time. Many algorithms attempt to reduce the number of passes to one, processing the data set in sufficiently large blocks. Nevertheless, additional iterations on the entire data set are generally needed to guarantee convergence to a stable solution. Reducing the number of passes is highly related to incremental algorithms, which can update the model as more records (points) are processed. Incremental algorithms with matrices on secondary storage are an important solution. Algorithms updating the model online based on gradient-descent methods have shown to benefit several common models [18] and they can be integrated as a UDF into the DBMS architecture. Stream processing represents an important variation of incremental algorithms to the extreme that each point is analyzed once, or even skipped. Building data set summaries that preserve its statistical properties is an essential optimization to accelerate convergence. Most algorithms build summaries based on sufficient statistics, most commonly for the Gaussian and Bernoulli densities. Sufficient statistics have been extended and generalized considering independence and multiple subsets from the data set, benefiting a big family of linear models. Data set squashing keeps high order moments (sufficient statistics beyond power 2) on binned attributes to compress a data set in a lossy manner; statistical algorithms on squashed data need to be modified considering weighted points.

Sampling (including randomized algorithms and Monte Carlo methods) is another acceleration mechanism that can accelerate processing by computing approximate solutions (only when an efficient sampling mechanism exists), but which always requires additional samples (e.g. bootstrapping) or extra passes over the entire data set to reduce approximation error: skewed probabilistic distributions and extreme values impact sampling quality. Data structures tailored to minimize I/O cost using RAM efficiently are another important mechanism. The most common data structures are balanced trees, tries, followed by hash tables. Trees are used to index transactions (FP-tree), itemsets (with a hash-tree combo) or to summarize points.

4. MODELS AND PATTERNS

Research on analyzing big data sets can be broadly classified into the following categories:

1. Statistical models, which involve matrices, vectors and histograms. Such models are computed by a combination of numerical and statistical methods.

2. Patterns, which involve exploratory search on a discrete combinatorial space (cubes, association rules, itemsets, sequences, graphs). Such patterns are commonly discovered by search algorithms exploring a lattice or a graph.

4.1 Models: Supervised and Usupervised

Here we present models in two major groups: unsupervised models (clustering, dimensionality reduction) and supervised models (classification, regression, feature selection, variable selection).

We start with clustering, one of the most well-known unsupervised data mining techniques. The data set is partitioned into subsets, so that points in the same subset are similar to each other. Clustering is commonly used as a building block for more complex models. Techniques that have been incorporated internally in a DBMS include K-means clustering and hierarchical clustering. Scalable K-means can work incrementally in one pass performing iterations on weighted points in main memory. This algorithm is also based on sufficient statistics. The algorithm relies on a low-level direct access to the file system which allows processing the data set in blocks. SQL has been successfully used to program clustering algorithms in SQL, such as K-means [23] and the more sophisticated EM algorithm [25]. Distance is the most demanding computation in these algorithms. It is feasible to develop an incremental version [23] with SQL queries, that requires a few passes (even two) on the data set; this approach is somewhat similar to iterated sampling (bootstrapping), with the basic difference that each record is visited at least once. UDFs further accelerate distance computation [24] by avoiding joins altogether. The EM algorithm represents an important maximum likelihood estimation method. Reference [25] represents a first attempt to program EM clustering (solving a Gaussian mixture problem) in SQL, proving it was feasible to program a fairly complex algorithm entirely with SQL code generation. In [25] several alternatives for distance computation are explored pivoting the data set and distances to reduce I/O cost. Mixture parameters are updated with queries combining aggregations and joins. This SQL solution requires multiple iterations and two passes per iteration. Deriving incremental algorithms, exploiting sufficient statistics and accelerating processing for EM with UDFs are open problems. Subspace clustering is a grid-based approach, which incorporates several optimizations similar to frequent itemset mining (discussed below). The data set is binned with user-defined parameters, which is easily and efficiently accomplished with SQL. SQL queries traverse the dimension lattice in a level-wise manner making one pass of each size. Acceleration is gained with aggressive pruning of dimension combinations. PCA, an unsupervised technique, is the most popular dimensionality reduction method. PCA computation can be pushed into a DBMS by computing the sufficient statistics for the correlation matrix with SQL or UDFs in one table scan. It is possible to solve PCA with a complex numerical method like SVD entirely in SQL. Such approach enables fast analysis of very high dimensional data sets (e.g. microarray data). Moreover, this solution is competitive with a state of the art statistical system (R package). PCA has also been solved summarizing the data with an aggregate UDF and computing SVD with the LAPACK library [28], giving two orders of magnitude performance improvement.

We now turn our attention to supervised techniques. Decision trees is a fundamental technique that recursively partition a data set to induce predictive rules. Sufficient statistics for decision trees are important to reduce number of passes numeric attributes are binned and row counts per bin or categorical value are computed. The splitting phase is generally accelerated using a condensed representation of the data set. Table-Valued Functions have been exploited to implement primitive decision tree operations. UDFs implement primitives to perform attribute splits, reducing the number of passes over the data set; this work also proposes a

prediction join, with new SQL syntax, to score new records. Reducing the number of passes on the data set to two or one using SQL mechanisms is an open problem. Exploiting MapReduce [13] to learn an ensemble of decision trees on large data sets using a cluster of computers is studied in [30]. This work introduces optimizations to reduce the number of passes on the data set.

We now discuss regression models, which are popular in statistics. Sufficient statistics for linear models (including linear regression) are generalized and implemented as a primitive function using aggregate UDFs, producing a tightly coupled solution. Basically sufficient statistics for clustering (as used by K-means) are extended with dimension cross-products to estimate correlations and covariances. Matrix equations are mathematically transformed into equivalent equations based on sufficient statistics as factors. The key idea is to compute those sufficient statistics in one pass, with SQL queries or even faster with aggregate UDFs. The equations solving each model can be efficiently evaluated with a math library, exporting small matrices whose size independent from data set size. Linear regression has been integrated with the DBMS with UDFs, exploiting parallel execution of the UDF in unfenced mode. Moreover, it is feasible to perform variable selection exploiting sufficient statistics [24]. Bayesian statistics are particularly challenging because MCMC methods require thousands of iterations. Recent research [20] shows it is feasible to accelerate a Gibbs sampler to select variables in linear regression under a Bayesian approach, exploiting data summarization. Database systems have been extended with model-based views in order to manage and analyze databases with incomplete or inconsistent content, to incrementally build regression and interpolation models, extending SQL with syntax to define model views.

We close discussing Support Vector Machines (SVMs), which are a mathematically sophisticated classifier. SVMs achieve a perfect separation of classes in a higher dimensional space, but have quadratic time complexity. Two fundamental SVM aspects need to be considered: kernel functions (mapping points to a higher dimensional space) and solving quadratic programming, a mathematically difficult problem. For instance, kernel functions have been integrated with a DBMS [19]. SVMs have been accelerated with hierarchical clustering and incremental matrix factorization.

4.2 Patterns: Cubes, Itemsets and Graphs

Pattern search is a different problem from statistical models because algorithms work in a combinatorial manner on a discrete space. Patterns include cubes, itemsets (association rules) [2], as well as their generalization to sequences (considering an order on items) and graphs. Given their generality and close relationship to social networks mining large graphs are currently the most important topic.

Cubes represent a mature research topic, but widely used in practice in BI tools. Most approaches avoid exploring the entire cube by pruning the search space. Methods increase the performance of multidimensional aggregations, by combining data structure for sparse data and cell precomputation at different aggregation levels of the lattice. The authors of [16] puts forward the plan of creating smaller, indexed summary tables from the original large input table to speed up aggregating executions. In [31] the authors explore tools to guide the user to interesting regions in order to highlight anomalous behavior while exploring large data cubes. Large lattices are explored with greedy algorithms.

Association rules were by far the most popular data mining topic in the past. Frequent itemset is generally regarded as the fundamental problem is association rule mining, solved by the well-known level-wise A-priori algorithm (exploiting the downward closure property). The common approach to mine association rules is to view frequent itemset search as optimizing multiple GROUP-BYs. From a DBMS perspective, exploiting SQL queries, UDFs and local caching are explored in [32]. This work studies how to write efficient SQL queries with k-way joins and nested subqueries to get frequent itemsets, but did not explore UDFs deep enough as they were more limited back then. SQL is shown to be competitive, but the most efficient solution (local caching) is based on creating a temporary file inside the DBMS, but apart from relational tables. We believe the slightly slower solution in SQL is more valuable in the long term. Recursive queries can search frequent itemsets, yielding a compact piece of SQL code, but unfortunately, this solution is slower than k-way joins. Relational division and set containment can solve frequent itemsets search, providing a complementary solution to joins. The key issue is that relational division requires different query optimization techniques, compared to SPJ operators. There has been an attempt to adapt data structures, like the FP-tree, to work in SQL, but experimental evidence over previous approaches is inconclusive. Mining frequent itemsets over sliding windows has been achieved with aggregate UDFs, interfacing with external programs. Table UDFs can update an entire cube in RAM in one pass for low dimensional data sets. Thus the dimension (or itemset) lattice can be maintained in main memory. The drawback is that TVFs require a cursor interface that disables parallel processing. The lattice is the main hindrance to perform processing with SQL queries because it cannot be easily represented as a tabular structure and because SQL does not have pointers and requires joins instead. There exists work on mining graphs (which represents a more general problem) with SQL, but solving narrow issues. Instead most work has focused on studying algorithms with MapReduce. For the most part, mining graphs with queries or UDFs remains unexplored.

5. THE FUTURE: RESEARCH ISSUES

Analyzing large volumes of relational and diverse text data together (i.e., big data analytics) rapidly growing adds a new level of difficulty and thus it nowadays represents a major research direction. The default storage mechanism in most DBMSs will likely remain row-based, but column stores [36] show promise to accelerate statistical processing, especially for feature selection, dimensionality reduction and cubes. Solid state memory may be an alternative for small volumes of data, but it will likely remain behind disks on price and capacity. Sharing RAM among multiple nodes in fast network can help loading large data sets in RAM. Shared disk is not a promising alternative due to the I/O bottleneck.

In parallel processing the major goal will remain the efficient evaluation of mathematical equations independently on each table partition, balancing workload, minimizing row redistribution. With the advances in fast network communication, larger RAM, multicore CPUs and larger disks the future default system will probably remain a shared-nothing parallel architecture with N processing units, each one with

its main memory and secondary storage (disk or solid state memory). Matrices are used extensively in mathematical models, computed with parallel algorithms. Thus we believe it is necessary to extend a DBMS with matrix data types, matrix operators and linear algebra numerical methods. In particular, it is necessary to study efficient mechanisms to integrate SQL with mathematical libraries like LAPACK. Array databases go in that direction, but they are incompatible with SQL and traditional relational storage mechanisms, resulting in multiple copies of the same data. Most likely, SQL will incorporate array storage mechanisms with a new data type and primitive operators.

Some common simple data mining algorithms may make their way to DBMS source code, whereas others may keep growing as SQL queries or UDFs. But most new data mining and statistical algorithms will be developed in languages like C++ or R. Given the similarity between MapReduce jobs and aggregate UDFs it is likely DBMSs will support automated translation from one into the other. Also, there will be faster interfaces to move data from one platform into the other. Extending SQL with new clauses is not a promising alternative, because SQL syntax is overwhelming and DBMS source code is required (many proposals have been simply forgotten). Combining SQL code generation and UDFs needs further study to program and optimize complex statistical and numerical methods. State-of-the-art data mining algorithms generally have or are forced to achieve linear time complexity in data set size. Can SQL queries callings UDFs match such time complexity exploiting relational physical operators (scan, join, sort)? There is evidence it can.

Future research will keep studying incremental model computation, block-based processing, data summarization, sampling and data structures for secondary storage under sequential and parallel processing. Developing incremental algorithms, beyond clustering and decision trees, is definitely challenging due to the need to reconcile parallel processing and incremental learning. Incorporating sampling as a primitive acceleration mechanism is a promising alternative to analyze large data sets, but approximation error must be controlled. Model selection is about selecting the best model out of several competing models; this task requires building several models with slightly different parameters or exploring several models concurrently. From the mathematical side the list is extensive. Overall, models involving time remain difficult to compute with existing DBMS technology (they are better suited to mathematical packages). Hidden Markov Models (HMMs) are a prominent example. Non-linear models (e.g. non-linear regression, neural networks) also deserve attention, but their impact is more limited. We anticipate recursive queries and aggregate UDFs will play a central role to analyze large graphs in a DBMS, whereas the state of the art for large graphs in social networks will be MapReduce.

Acknowledgments

This research work was partially supported by National Science Foundation grant IIS 0914861.

6. REFERENCES

[1] A. Abouzied, K. Bajda-Pawlikowski, J. Huang, D.J. Abadi, and A. Silberschatz. HadoopDB in action: building real world applications. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 1111–1114. ACM, 2010.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Conference*, pages 207–216, 1993.

[3] A. Behm, V.R. Borkar, M.J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, and V.J. Tsotras. ASTERIX: towards a scalable, semistructured data platform for evolving-world models. *Distributed and Parallel Databases (DAPD)*, 29(3):185–216, 2011.

[4] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. ACM KDD Conference*, pages 9–15, 1998.

[5] G.P. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *Proc. ACM SIGMOD Conference*, 2010.

[6] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *Proc. ACM SIGMOD Conference*, pages 287–298, 2004.

[7] J. Clear, D. Dunn, B. Harvey, M.L. Heytens, and P. Lohman. Non-stop SQL/MX primitives for knowledge discovery. In *ACM KDD Conference*, pages 425–429, 1999.

[8] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton. MAD skills: New analysis practices for big data. In *Proc. VLDB Conference*, pages 1481–1492, 2009.

[9] C. Cunningham, G. Graefe, and C.A. Galindo-Legaria. PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS. In *Proc. VLDB Conference*, pages 998–1009, 2004.

[10] S. Das, Y. Sismanis, K.S. Beyer, R. Gemulla, P.J. Haas, and J. McPherson. RICARDO: integrating R and hadoop. In *Proc. ACM SIGMOD Conference*, pages 987–998, 2010.

[11] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *ACM SIGMOD Conference*, pages 240–251, 2002.

[12] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[13] Eric Friedman, Peter Pawlowski, and John Cieslewicz. SQL/MapReduce: a practical approach to self-describing, polymorphic, and parallelizable user-defined functions. *Proc. VLDB Endow.*, 2(2):1402–1413, August 2009.

[14] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H. Jacobsen. Bigbench: towards an industry standard benchmark for big data analytics. In *Proc. ACM SIGMOD Conference*, pages 1197–1208. ACM, 2013.

[15] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-total. In *ICDE Conference*, pages 152–159, 1996.

[16] H. Gupta, V. Harinarayan, A. Rajaraman, and J.D.

Ullman. Index selection for OLAP. In *IEEE ICDE Conference*, 1997.

[17] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, New York, 1st edition, 2001.

[18] J. Hellerstein, C. Re, F. Schoppmann, D.Z. Wang, and et. al. The MADlib analytics library or MAD skills, the SQL. *Proc. of VLDB*, 5(12):1700–1711, 2012.

[19] B.L. Milenova, J. Yarmus, and M.M. Campos. SVM in Oracle database 10g: Removing the barriers to widespread adoption of support vector machines. In *VLDB Conference*, 2005.

[20] M. Navas, C. Ordonez, and V. Baladandayuthapani. On the computation of stochastic search variable selection in linear regression with UDFs. In *Proc. IEEE ICDM Conference*, pages 941 – 946, 2010.

[21] A. Netz, S. Chaudhuri, U. Fayyad, and J. Berhardt. Integrating data mining with SQL databases: OLE DB for data mining. In *Proc. IEEE ICDE Conference*, pages 379–387, 2001.

[22] C. Ordonez. Vertical and horizontal percentage aggregations. In *Proc. ACM SIGMOD Conference*, pages 866–871, 2004.

[23] 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.

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

[25] C. Ordonez and P. Cereghini. SQLEM: Fast clustering in SQL using the EM algorithm. In *Proc. ACM SIGMOD Conference*, pages 559–570, 2000.

[26] C. Ordonez and J. García-García. Referential integrity quality metrics. *Decision Support Systems Journal*, 44(2):495–508, 2008.

[27] C. Ordonez, J. García-García, and Z. Chen. Dynamic optimization of generalized SQL queries with horizontal aggregations. In *Proc. ACM SIGMOD Conference*, pages 637–640, 2012.

[28] C. Ordonez, N. Mohanam, C. Garcia-Alvarado, P.T. Tosic, and E. Martinez. Fast PCA computation in a DBMS with aggregate UDFs and LAPACK. In *Proc. ACM CIKM Conference*, 2012.

[29] F. Pan, X. Zhang, and W. Wang. CRD: fast co-clustering on large datasets utilizing sampling-based matrix decomposition. In *SIGMOD*, pages 173–184, New York, NY, USA, 2008. ACM.

[30] B. Panda, J. Herbach, S. Basu, and R.J. Bayardo. PLANET: Massively parallel learning of tree ensembles with MapReduce. In *Proc. VLDB Conference*, pages 1426–1437, 2009.

[31] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In *EDBT*, pages 168–182. Springer-Verlag, 1998.

[32] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. In *Proc. ACM SIGMOD Conference*, pages 343–354, 1998.

[33] K. Sattler and O. Dunemann. SQL database primitives for decision tree classifiers. In *Proc. ACM CIKM Conference*, pages 379–386, 2001.

[34] E. Soroush, M. Balazinska, and D. Wang. ArrayStore: A storage manager for complex parallel array processing. In *Proc. ACM SIGMOD Conference*, pages 253–264, 2011.

[35] M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and parallel DBMSs: friends or foes? *Commun. ACM*, 53(1):64–71, 2010.

[36] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB*, pages 1150–1160, 2007.

[37] K.Y. Whang, T.S. Yun, Y.M. Yeo, I.Y. Song, H.Y. Kwon, and I.J. Kim. ODYS: an approach to building a massively-parallel search engine using a DB-IR tightly-integrated parallel dbms for higher-level functionality. In *Proc. ACM SIGMOD Conference*, pages 313–324, 2013.

[38] A. Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian. Spreadsheets in RDBMS for OLAP. In *Proc. ACM SIGMOD Conference*, pages 52–63, 2003.

[39] G. Wu, E. Chang, Y.K. Chen, and C. Hughes. Incremental approximate matrix factorization for speeding up support vector machines. In *ACM KDD*, pages 760–766, New York, NY, USA, 2006. ACM.

[40] R.S. Xin, J.R., M. Zaharia, M.J. Franklin, S.S., and I. Stoica. Shark: SQL and rich analytics at scale. In *Proc. ACM SIGMOD Conference*, pages 13–24, 2013.

[41] Y. Zhang, W. Zhang, and J. Yang. I/O-efficient statistical computing with RIOT. In *Proc. ICDE*, 2010.