Towards an Analytics Query Engine [*]

Nantia Makrynioti
Athens University of Economics and Business
Athens, Greece
makriniotik@aueb.gr

Vasilis Vassalos
Athens University of Economics and Business
Athens, Greece
vassalos@aueb.gr

ABSTRACT

This vision paper presents new challenges and opportunities in the area of distributed data analytics, at the core of which are data mining and machine learning. At first, we provide an overview of the current state of the art in the area and then analyse two aspects of data analytics systems, semantics and optimization. We argue that these aspects will emerge as important issues for the data management community in the next years and propose promising research directions for solving them.

Keywords

Data analytics, Declarative machine learning, Distributed processing

1. INTRODUCTION

With the rapid growth of world wide web (WWW) and the development of social networks, the available amount of data has exploded. This availability has encouraged many companies and organizations in recent years to collect and analyse data, in order to extract information and gain valuable knowledge. At the same time hardware cost has decreased, so storage and processing of big data is not prohibitive even for smaller companies. Topic classification, sentiment analysis, spam filtering, fraud and anomaly detection are only a few analytics tasks that gained considerable popularity over the past few years, along with more traditional warehouse queries that gather statistics from data. Hence, making the deployment of solutions for such tasks less tedious, adds value to the services provided by these companies and organizations, and encourages more people to enhance their work using information from data.

It is clear that the areas of data mining and machine learning are at the core of data analysis tasks. However, developing such algorithms needs not only expertise in software engineering, but also a solid mathematical background in order to interpret correctly and efficiently the mathematical computations into a program. Even when experimenting with black box libraries, evaluating various algorithms for a task and tuning their parameters, in order to produce an effective model, is a time-consuming process. Things become even more complicated when we want to leverage parallelization on clusters of independent computers for processing big data. Details concerning load balancing, scheduling or fault tolerance can be quite overwhelming even for an experienced software engineer.

Research in the data management domain recently started tackling the above issues by developing systems for large-scale analytics that aim at providing higher-level primitives for building data mining and machine learning algorithms, as well as hiding low-level details of distributed execution. MapReduce [12] and Dryad [16] were the first frameworks that paved the way. However, these initial efforts suffered from low usability, as they offered expressive but at the same time low-level languages to program data analysis tasks. Soon the need for higher-level programming languages on top of these frameworks became apparent. Systems, such as Hive [23], Pig Latin [20], DryadLINQ [25] and Scope [9], offer higher-level languages that enable developers to write part of their programs in declarative style. Then, these programs are automatically translated into MapReduce jobs or Dryad vertices that form a directed acyclic graph (DAG), which is optimized for efficient distributed execution. This paradigm is adopted by other systems, too. Stratosphere [5], Tupleware [11] and MLbase [17] also aim at hiding details of distributed execution, such as load balancing and scheduling, in order to give the impression to the user that she develops code for a single machine.

Apart from the programming model, optimization techniques is another important issue that these systems address. Big data need fast and efficient solutions and as a consequence frameworks should leverage any opportunity for code optimization. Query rewriting, exploitation of data locality and vectorization are concepts already known and widely explored in databases and compilers. These techniques are also applied in the aforementioned analytics systems along with more recent ideas.

In this paper, we study systems for large-scale data analysis in the context of these two directions: semantics and optimization techniques. We present an overview of the immense development of systems for large scale data analysis that made their appearance over the past few years and propose research topics that we argue will attract the interest of the data management community in the near future. The rest of the paper is organized as follows. Section 2 describes classes of systems for distributed data analytics and section

[*]This work was co-funded by the European Union and the General Secretariat of Research and Technology, Ministry Of Education, Research Religious Affairs under the project Asset Management Optimization & Risk Management Software (AMOR) of the Bilateral R&T Cooperation Program Greece - Israel 2013-2015.

3 analyses open challenges concerning the semantics and the optimization methods used in such systems. Finally, section 4 concludes the paper.

2. CURRENT STATE OF THE ART

Current work in the area of big data analytics focuses on proposing programming models for data mining and machine learning tasks, and developing optimizations that result to efficient execution of users' programs on a distributed execution engine. So, large-scale analytics frameworks can be divided into two main categories: libraries of data mining/machine learning algorithms or sets of primitives to develop such algorithms.

Libraries provide implementations of algorithms commonly used in data analysis tasks, such as SVMs and K-means, targeted to a specific distributed execution platform. The user is able to use these algorithms in her code by calling them as functions, in order to load data from files or databases, transform data or use machine learning algorithms to analyse them. The programming paradigm is the same as that of a developer writing code for a single machine and calling functions from a third-party library. The difference is that this code is then automatically compiled by the system in order to be executed on a distributed platform. Such libraries are available for all popular distributed execution engines. Apache Mahout [2] was initially implemented on Hadoop [1] and is gradually extended to Spark [26]. A similar example is MLlib [19], a scalable machine learning library on top of Spark, whereas MADlib [10] is a library of SQL-based machine learning and data analysis algorithms, which run on database engines. They include algorithms for classification, clustering, collaborative filtering, dimensionality reduction and other useful preprocessing tasks.

Systems that belong to the second category provide a set of primitives to the users, in order to simplify the development of distributed data analysis algorithms. These primitives can be also combined with UDFs (User Defined Functions) in many cases to allow for custom code in data analysis tasks. In this class of systems, algorithms are not ready to call, but the user can use these primitives to develop machine learning or data mining algorithms that run on a specific distributed execution engine. The provided primitives hide low-level details concerning distribution, such as load balancing and fault tolerance, for which the system provides solutions. Declarative style combined with imperative/procedural programming, is popular in this kind of systems, gaining ground for the application of the DBMS paradigm in data analysis platforms.

This trend is already evident in existing platforms, such as Stratosphere, Jaql [6] and Pig Latin that provide a hybrid of procedural and declarative programming, as well as DryadLINQ, Tupleware and Spark that attempt to incorporate relational and other operators to a host language. Stratosphere provides three programming models organized as layers of a larger stack that comes down to an execution engine and a data storage system. Sopremo is the top layer of the stack and trades expressiveness for declarativity. It includes a considerable number of high-level operators, such as relational or domain-specific operators which offer more advanced functionality (e.g. duplicate detection, named entity recognition). Each Sopremo plan is translated to the programming model below Sopremo, PACT, and is finally executed on Nephele, Stratosphere's execution engine. Similar to Stratosphere's Sopremo layer, Jaql and Pig Latin define each workflow as a sequence of steps, but with each step performing a high-level transformation, such as SQL and ETL operations. Users are able to integrate custom code in their data analysis tasks by writing their own UDFs either in external languages, such as Java and Python, or by using operators of the system. Eventually, Jaql and Pig Latin scripts are automatically compiled to MapReduce jobs. A variant of Pig Latin, called MyriaL, extends its programming model with looping constructs and is used in Myria [15], a big data management service. Iterative processes are also a limitation for Jaql and Sopremo, as the developer is not able to define in these programming models that a given workflow will be repeated for a number of iterations or as long a condition holds.

Moving on to the incorporation of relational operators to a high-level language, we find DryadLINQ that transforms LINQ programs into distributed processes running on an execution engine called Dryad. The LINQ model incorporates constructs for manipulating data items into a host language, such as C# and other .NET languages. Iteration is expressed using loop constructs of the host language. Spark exposes a similar functional programming interface implemented in Scala, but provides greater support for iterative processes and expression of shared state among iterations. Finally, Tupleware follows the same approach as Stratosphere's PACT model by proposing operators that are second-order functions and take as argument a user defined first-order function. However, Tupleware's set of operators is more extended than the one provided by PACT. These operators are used to define workflows inside a host language. Then, the system transforms user's code into a distributed program, which is deployed and executed on a cluster of machines.

On the other hand, MLI API [22] and SystemML [13] that run on Spark and MapReduce respectively, aim to imitate the style of statistical computing languages, such as R and MATLAB, which are very popular among machine learning researchers and help them build software prototypes quickly. As a result, MLI includes interfaces for three main concepts, Optimizers, Algorithms and Models, as well as APIs for two data structures MLTable and LocalMatrix, which supports linear algebra operations. In SystemML, programs are sequences of statements written in DML, a language which includes constructs for input/output, control structures and assignments, as seen in R, on matrices or scalars. Nevertheless, more advanced features of R, such as objects and lists, are not currently supported by DML.

Finally, MLbase provides a declarative language above a layer of a library of algorithms, by which the user is able to define the type of task she wants to execute, e.g. classification, and the data to be used. Then the system tests various machine learning algorithms from its library and parameter values on the data provided, and determines an effective combination based on quality and time performance. Apart from the declarative language, MLbase also offers high-level primitives, such as gradient and stohastic gradient descent, that make development of distributed machine learning algorithms easier for researchers.

Despite the abundance of available systems and the variety of approaches that these follow, programmers are still expected to write a considerable amount of code in most of these platforms. The idea of providing greater degree

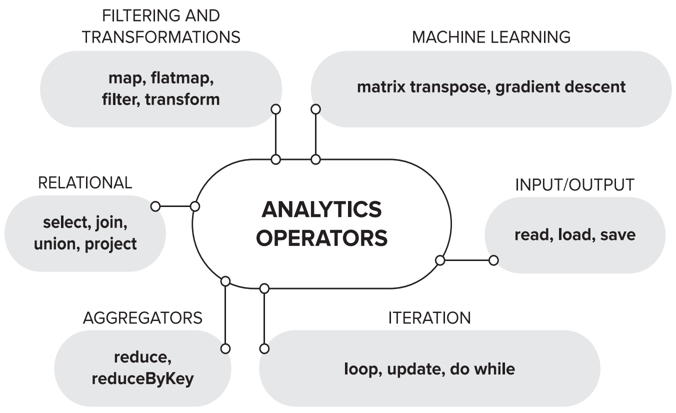

Figure 1: The main categories of analytics operators in current systems.

of declarativity by extending the set of operators that data analysis systems support, and minimize any glue code needed between operation calls would make the development and maintenance of programs much easier. The description of an algebra that would form the basis of these operators and model their semantics is a primary goal at this direction. We explore this open problem in the following section, as well as the challenges that are presented by optimizing the execution of tasks written in this set of operators.

3. OPEN CHALLENGES

3.1 Data Analytics Semantics

For some time now, the data management community compares the current situation in data analytics with the beginning of database systems. Currently, data mining and machine learning tasks in distributed platforms are done in ad hoc ways and developers have to use various systems, each targeted to a specific class of tasks. Data models and operators are different among systems, each exposing its own semantics, and there is no notion of an algebra that could form a basis for data analytics languages. Figure 1 displays the main categories of operators included in the current systems.

While relational queries are easily defined with the aforementioned operators, common concepts in machine learning, such as models and optimization algorithms, are not represented in a declarative way in the systems described above and their implementation still involves a lot of custom code. Hence, apart from the distributed execution, the development of machine learning algorithms is not simplified compared to more traditional languages such as R and MATLAB.

In an attempt to present declarative solutions for machine learning tasks, recent work [8] proposes Datalog as a suitable declarative foundation for analytics systems. As each system is usually targeted to a specific ML class of tasks (e.g. graph analytics), Datalog can serve as a logical layer where all these different programming models will be translated. The purpose is two-folded. First, to avoid developing optimizations for each new system and second to separate the user's program from its logical interpretation. The second argument is very important as it prevents changes in logical and physical layer from affecting user's code, whereas improvements in these layers can increase the efficiency of execution overall. An extension of Datalog could also cover more requirements of data analysis tasks and be exposed to the users as a programmable language. The distinction between the user's program and the logical layer is also supported by Hyracks [7], which serves as a parallel-platform for compiling higher-level declarative data-processing languages, such as Pig Latin and Hive.

We also consider the sets of operations provided by MLI API, MLbase and Spark ML [3] good attempts, as except from relational operators, they are also closer to the semantics of machine learning area by providing declarative operators for some frequent components of machine learning algorithms, such as linear algebra operations and gradient descent. The Spark ML package is also built around the key concepts of learning and data transformation algorithms, in order to standardize multiple APIs for machine learning on top of a data structure called DataFrame. Thus, a key ingredient in the development of optimizable, massively scalable analytics is the creation of an analytics query engine supporting a language with clear semantics that incorporates both relational operators and high-level operators for common components of machine learning algorithms, e.g. optimization functions and preprocessing tasks [1]. The purpose of this language is to model various data analysis tasks as plans consisting solely or mostly of operation calls. Another important aspect of an efficient analytics query engine is the implementation of optimizations, as described in the next section.

3.2 Optimizations in Data Analytics Systems

The optimization techniques adopted by large-scale analytics systems are mainly borrowed from two areas: databases and compilers.

As in database systems, query rewriting is also used in data analysis systems, either depending on heuristics or on a cost model. Heuristics are rules that are fired based on specific properties, but it is not examined whether these rules produce a faster plan in any case. DryadLINQ is currently applying heuristics in query rewriting, although in the future the group plans to design and implement a cost model for query optimization similar to the one used in DBMS. Stratosphere and Jaql have already moved on to cost-based optimizations, handling also difficulties that appear due to the large amount of user-defined code. Given that many operators of data analysis systems take as input user defined first-order functions, semantics of these operators cannot be known. This makes a big difference from traditional query optimizations. Static code analysis is one way to tackle unknown semantics. Through code analysis, it is possible to determine which data are read and written from each operator and separate them into read and write sets. Applying compiler optimizations to UDFs is also useful. Function and variable inlining, as well as SIMD vectorization are the most popular compiler optimization techniques we see in these systems. Jaql and Tupleware already exploit these ideas.

Concerning optimization in the context of the analytics model proposed above, we describe crucial issues that come into the picture. The first decision to be made regards the

[1]Part of these elements are also covered by the PMML format [14], a data exchange standard for sharing predictive models produced by data mining and machine learning algorithms.

definition of the form of the logical plan for the programs expressed with this model. Another important concern is the computation of appropriate cost metrics for evaluating these plans. Traditional cost models used in query optimization are not designed to cover every dimension of query performance, such as execution time. Machine learning has already made its way into prediction of execution latency and resource usage [4, 18] for queries running on DBMSs with relevant papers proposing the training of models on previous query instances, whose features are based on cardinalities estimated by the optimizer, the count of occurrences for each operator in the query plan and other statistics. The application of similar machine learning techniques for predicting performance metrics of data analysis programs is an interesting direction to pursue, which becomes even more challenging if we consider concurrent workloads [24]. Finally, given the sheer size of data that are analysed nowadays, the proposed optimization techniques should address challenges that arise in a distributed environment [21] and efficiently translate these logical plans to programs that would run in a distributed execution engine, such as Spark.

4. CONCLUSION

In this paper, we presented open challenges in the research area of big data analytics and we specifically focused on the aspects of semantics and optimization techniques. Given the variety of systems that are used for data analysis, we believe that the consolidation of semantics in an appropriate algebra and the evolvement of optimization methods applied in an analytics query engine are of great importance and will attract even more interest in the next few years.

5. REFERENCES

[1] Apache hadoop. `https://hadoop.apache.org/`.

[2] Apache mahout. `http://mahout.apache.org/`.

[3] Spark ml. `http://spark.apache.org/docs/latest/ml-guide.html`.

[4] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. Learning-based query performance modeling and prediction. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 390–401, 2012.

[5] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964, Dec. 2014.

[6] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Y. Eltabakh, C. Kanne, F. Özcan, and E. J. Shekita. Jaql: A scripting language for large scale semistructured data analysis. *PVLDB*, 4(12):1272–1283, 2011.

[7] V. Borkar, M. Carey, R. Grover, N. Onose, and R. Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1151–1162, 2011.

[8] Y. Bu, V. R. Borkar, M. J. Carey, J. Rosen, N. Polyzotis, T. Condie, M. Weimer, and R. Ramakrishnan. Scaling datalog for machine learning on big data. *CoRR*, abs/1203.0160, 2012.

[9] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow.*, 1(2):1265–1276, Aug. 2008.

[10] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. Mad skills: New analysis practices for big data. *Proc. VLDB Endow.*, 2(2):1481–1492, Aug. 2009.

[11] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, U. Çetintemel, and S. B. Zdonik. Tupleware: Redefining modern analytics. *CoRR*, abs/1406.6667, 2014.

[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation*. USENIX Association, 2004.

[13] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 231–242, 2011.

[14] A. Guazzelli, M. Zeller, W.-C. Lin, and G. Williams. PMML: An open standard for sharing models. *The R Journal*, 1(1):60–65, 2009.

[15] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, S. Xu, M. Balazinska, B. Howe, and D. Suciu. Demonstration of the myria big data management service. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 881–884, 2014.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, 2007.

[17] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*. www.cidrdb.org, 2013.

[18] J. Li, A. C. König, V. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for sql queries using statistical techniques. In *38th International Conference on Very Large Databases*. Very Large Data Bases Endowment Inc., August 2012.

[19] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in apache spark. *CoRR*, abs/1505.06807, 2015.

[20] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1099–1110, 2008.

[21] M. T. Ozsu. *Principles of Distributed Database Systems*. Prentice Hall Press, 3rd edition, 2007.

[22] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. E. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska. Mli: An api for distributed machine learning. *CoRR*, abs/1310.5426, 2013.

[23] A. Thusoo, J. S. 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. *Proc. VLDB Endow.*, 2(2):1626–1629, Aug. 2009.

[24] W. Wu, Y. Chi, H. Hacígümüş, and J. F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *Proc. VLDB Endow.*, 6(10):925–936, Aug. 2013.

[25] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and J. Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 1–14, 2008.

[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, 2010.