Towards a Multi-engine Query Optimizer for Complex SQL Queries on Big Data

Evdokia Kassela, Ioannis Konstantinou, Nectarios Koziris
CSLab, National Technical University of Athens, Greece
{evie, ikons, nkoziris}@cslab.ece.ntua.gr

***Abstract*—In an era where big data analytics has become a first-class requirement for both the industrial and the academic community, multiple engines are built to execute distributed domain-specific analytics. SQL-based big data analytics is a very popular but also challenging domain due to its complexity that requires multiple runtime query optimizations. Popular frameworks, such as Presto and SparkSQL, commonly retrieve data from multiple sources and process them locally using domain-specific optimizers. However, recent work indicates that no single engine offers the optimal all-in-one solution for all types of SQL queries. Taking this into account, we envision building an optimizer to facilitate faster distributed SQL analytics over multiple engines, which will perform operator-level optimization using Machine Learning techniques and will exploit the sophisticated data-driven local engine optimizations.**

***Index Terms*—big data analytics, SQL, multi-engine, optimizer**

I. Introduction

The constantly growing need to analyze large amounts of data, has led to the development of multiple distributed execution engines that operate on top of 'big data'. Each engine usually targets a different analysis domain, e.g. SQL, Graph Analytics or Machine Learning, and employs data-driven optimizations. However, there also exist general-purpose engines that perform multi-domain analysis, with Spark [1] being the most prominent example of such an engine.

SQL-based analytics is a fundamental and particularly popular field for data analysts, but it is also the most challenging domain due to the complexity of certain SQL queries and the volume of data. This has led to the adoption of different architectures and approaches ([2], [3]) in order to optimize the execution of a query. At the same time, the 'one size fits all' paradigm has outgrown itself [4] as the data that need to be queried quite often come from different data sources and are stored in different formats. For example, limited structured data are usually stored in relational databases (RDBMS) and main memory SQL systems, while unstructured data from loggers are stored in NoSQL databases and distributed file systems as the Hadoop DFS [5].

In order to facilitate querying large amounts of data from different data sources, Facebook has developed its own distributed SQL execution engine, named Presto [6]. Presto moves data from various remote storage engines locally to process them, and thus the optimal setup for Presto is to be installed on all available data nodes to avoid excessive data movements. However, it does not employ the sophisticated data-driven local optimizations of the underlying storage engines, but uses its own cost-based optimizer which still operates suboptimally with complex analytics queries [7]. Spark's SparkSQL [8] module also provides a similar functionality, as it can retrieve data from other storage engines and process them locally with its own cost-based optimizer, but, similar to Presto, installation within the data nodes is required to achieve data locality and local engine optimizations are ignored.

In our latest work [7] we evaluated state-of-the-art general-purpose and specialized big data analytics frameworks like SparkSQL, Presto, and Hive on Tez ([9], [10]) with SQL workloads. The results indicate that no single specialized or general-purpose system offers the optimal all-in-one solution, as the optimizers' efficiency and the memory management play an important role in the performance of different SQL queries. In order to achieve the fastest query execution under these circumstances, it is crucial to take into account each engine's optimizations and operators performance before running a query, following a multi-engine execution approach.

Along these lines, polystore systems have been developed ([11]–[15]) to handle the different data models of various stores and allow multi-engine execution. These systems perform multi-domain analytics over different engines by pushing part of the execution workflow on the underlying most appropriate engine. Although this approach allows local optimizations to be performed by the engines themselves, the existing implementations of polystore systems treat the different engines' operator and cost models as black-boxes ignoring these optimizations. Moreover, the cost models are usually user provided and integrated into the system in order to perform optimized planning. It is therefore difficult to adopt such systems in a custom user environment since adding a new engine and its cost models seems a challenging procedure. It is also worth mentioning that in certain cases ([12]–[14]), the user may need to use a custom language to express his queries. Taking these aspects into account, it is clear that a much simpler solution could be developed to allow data analysts to perform SQL analytics in a more user-friendly and self-configured environment.

On the other hand, some of these optimization approaches ([13], [15]) delegate the query plan optimization problem to a problem of minimizing intermediate result movements. Although this is an interesting approach, as the cost of moving a large intermediate result between engines can seriously affect the query performance, with this approach a specific operator's performance over different engines is not properly taken into account and only data placement is optimized. Most

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

importantly, none of the existing polystore systems properly employ the local physical optimizations that are critical in the SQL domain.

II. Approach

Since no single engine can offer out-of-the-box the best performance and taking into account the complexity and inadequacy of existing polystore systems in such a domain, we propose a new framework that will optimize distributed SQL query execution on big data over multiple engines with the following features:

- **Multi-engine aware query transformation**. Users will be able to submit a complex SQL query which will be rewritten into multiple SQL statements that will be executed on different engines. Different statements will be created in case a cross-engine data transfer is required between them. Statements that have no data dependencies and can be executed in parallel in a different or even the same engine will be scheduled appropriately. The overhead of data transfers will be taken into account when building the rewritten query plan, and statements will be divided into multiple ones or grouped depending on the estimated costs. For cross-engine data transfers we will avoid materializing intermediate results, and instead in-memory views will be created on the selected engines. These views will be registered as in-memory tables which will then be fetched by other engines using 'SELECT *' statements.
- **Operator-level optimization**. Different operators will be scheduled for execution on different engines by creating the appropriate statement. The underlying engines' physical optimizations and operators' performance will be taken into account, as cost models of all the physical operators will be automatically built for this purpose using offline performance profiling. During this initial offline model training procedure, specific SQL queries (also containing hints) that correspond to basic SQL operators (e.g. GROUP BY, broadcast/repartition JOIN [16]) will be executed on available engines using different data sizes as a first step. At a second phase, JOIN queries with no hints using the same data sizes of the previous step will be executed to obtain an estimation of automatic physical optimizations performed by each engine. It is important to notice that, when building the cost models of the operators, statistics that are related to the input and output table size and selectivity will be taken into account apart from operator performance metrics in order to achieve minimal intermediate result movements between engines.

An initial high-level outline of the envisioned architecture is presented in Figure 1. In brief, our framework will operate as a cross-engine optimizer which will produce a rewritten query and will schedule operators execution over different engines. State-of-the-art big data and RDBM systems that support SQL semantics will be integrated with our framework, such as Apache Spark, Apache Hive, Presto, PostgreSQL [17] and Apache Ignite [18]. Minimal effort will be required to add a new engine by simply integrating the engine API that will be used to remotely submit SQL code. The data that the engines

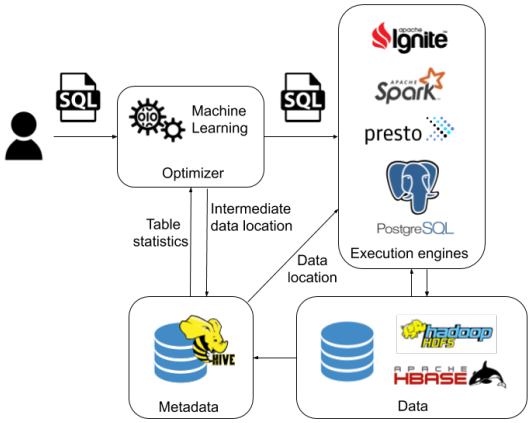

Fig. 1. High-level outline of the envisioned architecture

will operate on, will reside in big data stores such as HDFS and HBase, and relational databases as PostgreSQL.

Almost all the synchronous big data analytics engines such as Spark and Presto, rely on data statistics, known as metadata, to perform query cost-based optimization. The Hive metastore is the most popular metadata storage engine that can store metadata of tables residing in various stores including HDFS, HBase and any relational database and it will be a basic functional component in our design, where all tables will be registered and accessed. Hive will also be used to create temporary in-memory tables in order to store the intermediate results location that will be accessed by multiple engines. To sum up, our envisioned framework will include an optimizer which will use Machine Learning/Deep Learning techniques to build cost models of all the basic SQL operators and estimate a query's execution cost using these models and the Hive metastore table statistics.

III. Related Work

We divide existing work in two categories: the ones that offer SQL APIs to the users and the ones with custom APIs.

A. SQL API

With MuSQLe [19] framework, the authors optimize distributed SQL query execution over multiple engines. The framework optimization is performed on the logical plans, allowing local physical optimization to be performed by the engines themselves, and it also takes into account intermediate result movements. Although each engine operators and cost-models do not need to be integrated, but only specific simple API calls must be implemented for each one, the engines have all been modified to allow creating pseudo-tables and allow estimating SQL queries execution time. Unfortunately, the framework retrieves intermediate results and then sends them to other engines, which is prohibitive for using it with large datasets as required for big-data analytics and this is also proved by the experimental evaluation which is performed with relatively small datasets.

BigDAWG [11] is a polystore system that offers cross-engine query execution. Storage engines that can be queried with a specific query language and use a specific data model are considered as an island of information. A different island represents data that can be queried with a different query

language over the same or different storage backend(s). The primal target of the system is to identify semantic equivalencies among operators from different backend engines and examine optimization opportunities between islands for individual operators. Users can either submit queries to a specific island or create cross-engine queries, which involve transforming and transferring data between engines. BigDAWG relies on black box performance profiling of its underlying engines ignoring their local optimizations, hence it does not employ the sophisticated physical optimizations during query planning.

MISO [15] is another polystore system which primarily focuses on tuning the data placement on various stores with an aim to reduce data movements between them. An online algorithm is used to strategically place materialized views in the underlying stores using predefined cost functions and 'what-if' analysis is used to evaluate hypothetical physical placements. During the integration of a new store, it is therefore also required to manually define a new cost function. However, the actual performance of different physical operators on the underlying engines is omitted from cost estimation.

B. Non SQL API

CloudMdsQL [12] provides a SQL-like language for expressing queries and submits them to the underlying integrated SQL and NoSQL engines. It applies rule-based optimizations in sub-query level using user-defined cost models. AWESOME [13] is a similar platform developed to support analytics on social data. A language named ADIL is used to define original and derived data placement and computations over different data models and engines, but also native queries can be executed on single engines. Nevertheless, this system only performs data ingestion optimization, similar to [15]. In both of these approaches a custom language is used to express queries, therefore during the integration of a new engine a custom wrapper must also be created in order to transform CloudMdsQL/ADIL queries into engine specific operations. The same applies for the following scheduler that we describe below, named IReS.

IReS [14] is a multi-engine resource scheduler for big-data analytics workflows. Workflows of analytics tasks such as SQL queries, ML algorithms, etc. can be defined with a JSON-based metadata framework and the system uses performance and cost models (created after profiling) together with a user-defined policy to schedule the execution and resource allocation in each engine. The optimal plan for moving data between engines is also generated. However, the need to describe datasets, operators and workflows in a specific detailed JSON format renders the procedure to execute a single but complex SQL query over-complicated, and moreover no sub-query level optimization is possible in this case as the query is scheduled for execution as-is on a specific engine.

ACKNOWLEDGMENT

This work has been co-financed by the European Union and Greek national funds through the Operational Program "Competitiveness, Entrepreneurship and Innovation", under the call RESEARCH – CREATE – INNOVATE (project code: T1EDK-04605).

REFERENCES

[1] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: A Unified Engine for Big Data Processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/2934664

[2] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A Comparison of Approaches to Large-scale Data Analysis," in *SIGMOD*, 2009, pp. 165–178.

[3] A. Floratou, U. F. Minhas, and F. Özcan, "SQL-on-Hadoop: Full Circle Back to Shared-nothing Database Architectures," *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1295–1306, Aug. 2014.

[4] M. Stonebraker and U. Cetintemel, "One Size Fits All: An Idea Whose time has come and gone," in *ICDE*, 2005, pp. 2–11.

[5] D. Borthakur, *The hadoop distributed file system: Architecture and design*. Hadoop Project Website, 2007.

[6] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner, "Presto: SQL on Everything," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, April 2019, pp. 1802–1813.

[7] E. Kassela, N. Provatas, I. Konstantinou, A. Floratou, and N. Koziris, "General-Purpose vs Specialized Data Analytics Systems: A Game of ML & SQL Thrones," in *2019 IEEE International Conference on Big Data (Big Data)*, Dec 2019.

[8] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark SQL: Relational Data Processing in Spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 1383–1394. [Online]. Available: http://doi.acm.org/10.1145/2723372.2742797

[9] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive - a petabyte scale data warehouse using Hadoop," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, March 2010, pp. 996–1005.

[10] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications," *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1357–1369, 2015.

[11] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. Zdonik, "The BigDAWG Polystore System," *SIGMOD Rec.*, vol. 44, no. 2, pp. 11–16, Aug. 2015.

[12] B. Kolev, C. Bondiombouy, P. Valduriez, R. Jimenez-Peris, R. Pau, and J. Pereira, "The CloudMdsQL Multistore System," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: ACM, 2016, pp. 2113–2116. [Online]. Available: http://doi.acm.org/10.1145/2882903.2899400

[13] S. Dasgupta, K. Coakley, and A. Gupta, "Analytics-driven data ingestion and derivation in the AWESOME polystore," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 2555–2564.

[14] K. Doka, N. Papailiou, D. Tsoumakos, C. Mantas, and N. Koziris, "IReS: Intelligent, Multi-Engine Resource Scheduler for Big Data Analytics Workflows," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 1451–1456.

[15] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, and M. J. Carey, "MISO: Souping Up Big Data Query Processing with a Multistore System," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 1591–1602. [Online]. Available: http://doi.acm.org/10.1145/2588555.2588568

[16] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, "A comparison of join algorithms for log processing in mapreduce," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 975–986.

[17] "PostgreSQL." https://www.postgresql.org/.

[18] "Apache Ignite." https://ignite.apache.org/.

[19] V. Giannakouris, N. Papailiou, D. Tsoumakos, and N. Koziris, "MuSQLE: Distributed SQL query execution over multiple engine environments," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 452–461.