Query Processing in Cloud Multistore Systems

Carlyna Bondiombouy

Query Processing in Cloud Multistore Systems

Carlyna Bondiombouy *
†
Ph.D. thesis start date: 01/10/2014
Zenith team, Inria and LIRMM, University of Montpellier, France
carlyna.bondiombouy@inria.fr

1. INTRODUCTION

In the context of the cloud, we are witnessing a proliferation of data management solutions, including NoSQL data stores (e.g. Hbase, MongoDB, Neo4J), file systems (e.g. HDFS) and parallel processing frameworks (e.g. MapReduce, Spark). This makes it very hard for a user to access and analyze efficiently her data sitting in different data stores, e.g. RDBMS, NoSQL and HDFS. Processing queries against heterogeneous data sources has long been studied in the context of multidatabase systems and data integration systems [9]. However, these solutions no longer apply in the cloud, primarily because of the wide variety of models and languages of cloud data stores (key-value, document, table, graph, etc.). To address this problem, multistore systems,[4, 5, 6, 8, 10] have been recently proposed to provide integrated access to multiple, heterogeneous data stores through a single query engine. However, most multistore systems trade data store autonomy for performance and work only for certain categories of data stores, typically with RDBMS.

In the CoherentPaaS project [2], we are developing the Cloud Multidatastore Query Language (CloudMdsQL) [7]. CloudMdsQL is a functional SQL-like language, capable of querying multiple heterogeneous data stores within a single query that may contain embedded invocations to each data store's native query interface. The query engine has a fully distributed architecture, which allows query engine nodes to be collocated with data store nodes in a computer cluster. Compared to current multistore systems, CloudMdsQL is more general as it can be used to access any kind of data store, while respecting their autonomy. The major innovation of CloudMdsQL is that a query can exploit the full power of the local data stores, by simply allowing some local data store native queries (e.g. a breadth-first search query against a graph database) to be called as functions, integrated in SQL-like statements.

*Work partially funded by the European Commission under the Integrated Project CoherentPaaS [2].
†Ph.D. thesis advised by Patrick Valduriez (Inria).

2. PROBLEM DEFINITION

In this thesis, we address the problem of efficient query processing of multiple data stores with CloudMdsQL. What makes this problem difficult is the capability of CloudMdsQL to allow function calls to the data stores to be expressed in the native source languages. The problem is the optimization and efficient execution of nested queries, where a query on a data source may use inputs or parameters produced by a function call to another data source. In particular, the materialization of intermediate results to be exchanged between different data sources needs to be carefully studied as it may require specific data transfer and storage techniques.

The query optimization problem can be expressed as follows : Let $Q(S1, S2, Sn)$ be a nested query of the form $Q1(S1) -> Q2(S2) -> ...Qn(Sn)$, over n data stores, each with a different data model and query language, and in some cases (e.g. a document store, a graph data store) a different API, propose an approach to translate Q into an optimized query execution plan (QEP), with efficient management of intermediate results.

The fully distributed architecture of the query engine provides important opportunities for optimization, in particular, minimizing data shipping between nodes, select pushdown, bind join, and join ordering. However, compared to traditional distributed query processing, we must address several issues to fully exploit this architecture. First, the lack of cost models in some NoSQL data stores and the limited capability to build database statistics do not allow for defining a precise cost model for query optimization. Second, native functions (expressed in the native data store languages) can be long running and need specific attention. Third, CloudMdsQL supports specific operators, such as Map, Filter and Reduce, to be able to use parallel processing frameworks. The use of these operators in conjunction with SQL-like statements requires new rewrite rules for optimization.

3. RELATED WORK

We can divide multistore systems between loosely-coupled, tightly-coupled and hybrid. Loosely-coupled multistore systems provide support for autonomous data stores, much like multidatabase systems, e.g. DISCO [11]. For instance, BigIntegrator [13] integrates data from NoSQL big data, such as Google's Bigtable, and relational databases. The system relies on mapping a limited set of relational operators to native queries expressed in GQL (Google Bigtable query

language). Since GQL represents a subset of SQL, this is easy but only works for Bigtable-like systems.

Tightly-coupled multistore systems trade autonomy for performance. For instance, Odyssey [6] enables storing and querying data within HDFS and RDBMS, using opportunistic materialized views. JEN [12] allows joining data from two data stores, HDFS and RDBMS, with specific parallel join algorithms, in particular, an efficient zigzag join algorithm, and techniques to minimize data movement. As the data size grows, executing the join on the HDFS side appears to be more efficient.

Hybrid systems are similar to tightly-coupled systems, e.g. integrating HDFS and RDBMS in a shared-nothing cluster, except that the HDFS data is accessed through a data processing framework like MapReduce. For instance, HadoopDB [3] provides Hadoop MapReduce/HDFS access to multiple single-node RDBMS servers (e.g. PostgreSQL or MySQL) deployed across a cluster, as in a shared-nothing parallel DBMS. It interfaces MapReduce with RDBMS by database connectors that execute SQL queries to return key-value pairs.

Our work fits in the hybrid system category. However, it is not limited to MapReduce to access HDFS data, as we can support other frameworks like Spark. Furthermore, it does not give up data store's autonomy, yet allowing for optimization through the use of native subqueries and operator ordering.

4. APPROACH

To address the problem defined in Section 2, we exploit the declarativity of CloudMdsQL and the distributed architecture of the query engine [1]. In particular, the fact that the query engine nodes can directly communicate with each other, by exchanging code (QEPs) and data, allows important optimizations (e.g. minimizing data transfers by moving the smallest intermediate data for subsequent processing by one particular node).

As for declarative query languages (e.g. SQL), a QEP can be abstracted as a tree or DAG of operators and communication (send/receive) operators to exchange data and commands between query engine nodes. This allows us to reuse and extend the techniques from distributed query processing, which we adapt to our fully distributed architecture. In particular, we strive to:

- Minimize local execution time in the data stores, by pushing down select operations in the data store subqueries and exploiting bind join by query rewriting;
- Minimize global execution time by operator ordering;
- Minimize communication cost and network traffic by reducing data transfers between nodes.

To compare alternative rewritings of a query, we start with a simple catalog, which is replicated at all nodes in primary copy mode. The catalog provides basic information about data store collections such as cardinalities, attribute selectivities and indexes, and a simple cost model. Such information can be given with the help of the data store administrators. The query language provides the possibility for the user to define cost and selectivity functions whenever they cannot be derived from the catalog, mostly in the case of using native subqueries.

The search space explored for optimization is the set of all possible rewritings of the initial query, by pushing down select operations, expressing bind joins and ordering operators. For subqueries represented by sequences of map/filter/reduce (MFR) operations, we apply MFR rewrite rules to determine the optimal place for inclusion of pushed down operations within the MFR operator chain.

Unlike in traditional query optimization where many different permutations are possible, this search space is not very large, so we can use a simple exhaustive search strategy.

References

[1] Cloudmdsql project. cloudmdsql.gforge.inria.fr.

[2] Coherentpaas project. coherentpaas.eu.

[3] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. Hadoopdb: An architectural hybrid of mapreduce and DBMS technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[4] F. Bugiotti, D. Bursztyn, A. D., I. Ileana, and I. Manolescu. Invisible glue: Scalable self-tuning multistores. In *CIDR Int. Conf.*, 2015.

[5] D. J. DeWitt, A. Halverson, R. V. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling. Split query processing in polybase. In *SIGMOD Int. Conf.*, pages 1255–1266, 2013.

[6] H. Hacigümüs, J. Sankaranarayanan, J. Tatemura, J. LeFevre, and N. Polyzotis. Odyssey: A multi-store system for evolutionary analytics. *PVLDB*, 6(11):1180–1181, 2013.

[7] B. Kolev, P. Valduriez, C. Bondiombouy, R. Jiménez-Peris, R. Pau, and J. Pereira. Cloudmdsql: Querying heterogeneous cloud data stores with a common language. In *Distributed and Parallel Databases*, 2015. Extended version of BDA 2014. Under revision.

[8] J. LeFevre, J. Sankaranarayanan, H. Hacigümüs, J. Tatemura, N. Polyzotis, and M. J. Carey. MISO: souping up big data query processing with a multistore system. In *SIGMOD Int. Conf.*, pages 1591–1602, 2014.

[9] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

[10] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing analytic data flows for multiple execution engines. In *SIGMOD Int. Conf.*, pages 829–840, 2012.

[11] A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE TKDE*, 10(5):808–823, 1998.

[12] T. Yuanyuan, T. Zou, F. Özcan, R. Gonscalves, and H. Pirahesh. Joins for hybrid warehouses: Exploiting massive parallelism and enterprise data warehouses. In *EDBT/ICDT Int. Conf.*, pages 373–384, 2015.

[13] M. Zhu and T. Risch. Querying combined cloud-based and relational databases. In *Int. Conf. on Cloud and Service Computing CSC*, pages 330–335, 2011.