Distributed Query Processing on the Cloud: the Optique Point of View (Short Paper)*

Herald Kllapi[2], Dimitris Bilidas[2], Ian Horrocks[1], Yannis Ioannidis[2], Ernesto Jimenez-Ruiz[1], Evgeny Kharlamov[1], Manolis Koubarakis[2], Dmitriy Zheleznyakov[1]

[1] Oxford University, UK
[2] University of Athens, Greece

Abstract. The Optique European project [3] [6] aims at providing an end-to-end solution for scalable access to Big Data integration, where end users will formulate queries based on a familiar conceptualization of the underlying domain. From the users' queries the Optique platform will automatically generate appropriate queries over the underlying integrated data, optimize and execute them on the Cloud. In this paper we present the distributed query processing engine of the Optique platform. The efficient execution of complex queries posed by end users is an important and challenging task. The engine aims at providing a scalable solution for query execution in the Cloud, and should cope with heterogeneity of data sources as well as with temporal and streaming data.

1 Introduction

The Optique Project aims at providing end users with the ability to access Big Data through queries expressed using familiar conceptualization of the underlying domain. This approach is usually referred to as *Ontology Based Data Access* (OBDA) [12, 2].

In Figure 1 we present the architecture of the Optique OBDA approach. The core elements of the architecture are an *ontology*, which describes the application domain in terms of user-oriented vocabulary of classes (usually referred as concepts) and relationships between them (usually referred as roles), and a set of *mappings*, which relates the terms in the ontology and the schema of the underlying data source. End-users formulate queries using the terms defined by the ontology, which should be intuitive and correspond to their view of the domain, and thus, they are not required to understand the data source schemata. The main components of the Optique's architecture are

- *the Query Formulation component* that allows end users to pose queries to the system,
- *the Ontology and Mapping Management component* that allows for bootstrapping of ontologies and mappings during the installation of the system and for their subsequent maintenance,

* This research was financed by the Optique project with the grant agreement FP7-318338.

[3] http://www.optique-project.eu

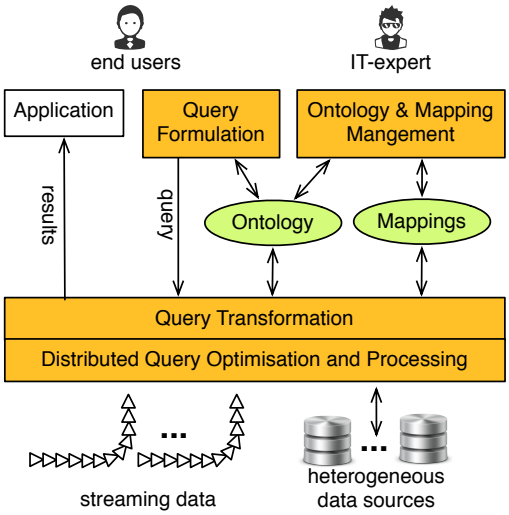

Fig. 1. The general architecture of the Optique OBDA system

- *the Query Transformation component* that rewrites users' queries into queries over the underlying data sources,
- *the Distributed Query Optimisation and Processing component* that optimises and executes the queries produced by the Query Transformation component.

All the components will communicate through agreed APIs.

In order for the Optique OBDA solution to be practical, it is crucial that the output of the query rewriting process can be evaluated effectively and efficiently against the integrated data sources of possibly various types, including temporal data and data streams. This efficiency for Big Data scenarios is not an option – it is a necessity. We plan to achieve the efficiency by both *massive parallelism*, i.e., running queries with the maximum amount of parallelism at each stage of execution, and *elasticity*, i.e., by allowing a flexibility to execute the same query with the use of resources that depends on the the resource availability for this particular query, and the execution time goals. The role of the Distributed Query Optimisation and Processing component is to provide this functionality and we will focus on the component in this paper.

An important motivation for the Optique project are two demanding use cases that will give to the project the necessary test-bed. The first one is provided by Siemens [4] and encompasses several terabytes of temporal data coming from sensors, with an increase rate of about 30 gigabytes per day. The users need to query these data in combination with many gigabytes of other relational data that describe events. The second use case is provided by Statoil [5] and concerns more than one petabyte of geological data. The data are stored in multiple databases which have different schemata and the user has to access many of them in order to get results for a single query. In general, in the oil and gas industry IT-experts spend 30–70% of their time gathering and assessing the

[4] http://www.siemens.com
[5] http://www.statoil.com

quality of data [3]. This is clearly very expensive in terms of both time and money. The Optique project aims at solutions that reduce the cost of data access dramatically. More precisely, Optique aims at reducing the running times of the queries for these use cases from hours to minutes and from days to hours. A bigger goal of the project is to provide a platform[6] with a generic architecture that can be easily adapted to any domain that requires scalable data access and efficient query execution for OBDA solutions.

The rest of this paper is organized as follows. In Section 2 we first give an overview of the system architecture and then we present a more detailed description of the basic components. In Section 3 we present some uses cases. In Section 4 we present some related work and in Section 5 we conclude.

2 System Architecture

The distributed query execution is based on the ADP [14], a system for complex dataflow processing in the cloud. ADP has been developed and used successfully in several European projects. The initial ideas came from Diligent [5]. Then ADP was adapted and used in project Health-e-Child as a Medical Query Processing Engine [10]. Subsequently, it was refined to support more execution environments, more operators, and a more query processing and optimization algorithms. ADP has been used successfully at the University of Athens for large scale distributed sorting algorithms, large scale database processing, and also for distributed data mining problems.

The general architecture of the distributed query answering component within the Optique platform is shown in Figure 2. The system utilizes state-of-the-art database techniques: *(i)* a declarative query language based on data flows, *(ii)* the use of sophisticated optimization techniques for executing queries efficiently, *(iii)* operator extensibility to bring domain specific computations into the database processing, and *(iv)* execution platform independence to insulate applications from the idiosyncrasies of the execution environments, such as local clusters, private clouds, or public clouds.

The query is received through the gateway using JDBC API (Java Database Connectivity). This communication mainly involves interaction with the Query Transformation component. The Master node is responsible for initialization and coordination of the process. The Optimization Engine produces the execution plan for the query using techniques described in [11]. Next, the execution plan is given to the Execution Engine which is responsible for reserving the necessary resources, sending the operators of the graph to the appropriate workers, and monitor the execution.

The system uses two different communication channels between the different components of the system. Data from the relational data sources, streams, and federated sources is exchanged between the workers using lightweight TCP connections and compression for high throughput. All the other communications (e.g., signals denoting that a node is connected, execution is finished, etc.), is done through a peer-to-peer network (P2P Net). For the time being, this network is a simple master-slaves using Java-RMI (Remote Method Invocation).

[6] Optique's solutions are going to be integrated via the Information Workbench platform [8].

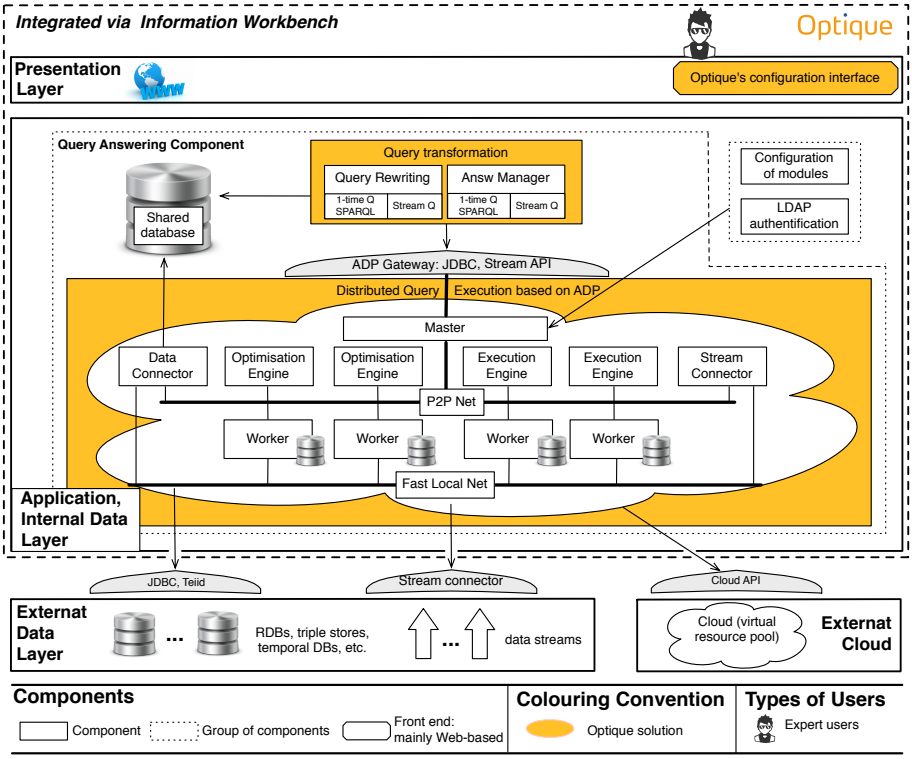

Fig. 2. General architecture of the ADP component within the Optique System

Language and Optimization: The queries are expressed in SQL. Queries are issued to the system through the gateway. The SQL query is transformed to a data flow language allowing complex graphs with operators as nodes and with edges representing producer-consumer relationships. The first level of optimization is planning. The result of this phase is an SQL query script. We enhanced SQL by adding the table partition as a first class citizen of the language. A table partition is defined as a set of tuples having a particular property (e.g., the value of a hash function applied on one column is the same for all the tuples in the same partition). A table is defined as a set of partitions. The optimizer produces an execution plan in the form of a directed acyclic graph (DAG), with all the information needed to execute the query. The following query is an example.

```
DISTRIBUTED CREATE TABLE lineitem_large TO 10 ON l_orderkey AS
SELECT * FROM lineitem WHERE l_quantity = 20
```

The query creates 10 partitions of a table with name lineitem_large with rows based on a selection condition. The partitioning is based on the column l_orderkey.

Execution Engine: ADP relies on an asynchronous execution engine. As soon as a worker node completes one job, it is sending a corresponding signal to the execution engine. The execution engine uses an asynchronous event based execution manager,

which records the jobs that have been executed and assigns new jobs when all the prerequisite jobs have finished.

Worker Pool: The resources needed to execute the queries (machines, network, etc.) are reserved or allocated automatically. Those resources are wrapped into containers. Containers are used to abstract from the details of a physical machine in a cluster or a virtual machine in a cloud. Workers run queries using a python wrapper of SQLite [7]. This part of the system, which is available [8], can also be used as a standalone single node DB. Queries are expressed in a declarative language which is an extension of SQL. This language facilitates considerably the use of user-defined functions (UDFs). UDFs are written in Python. The system supports row, aggregate, and virtual table functions.

Data / Stream Connector: Data Connector and Stream Connector are responsible for handling and dispatching the relational and stream data through the network respectively. These modules are used when the system receives a request for collecting the results of executed queries. Stream Connector uses an asynchronous stream event listener to be notified of incoming stream data, whereas Data Connector utilizes a table transfer scheduler to receive partitions of relational tables from the worker nodes.

3 Use Cases

Now we present some of the use cases of the distributed query processing component.

Data Import: The system provides the possibility to import data from several heterogenous sources. These data can be of many different types, including relational data, data in file formats like comma-separated values files or XML and streams. When the data is in the form of streams, the procedure is initiated through the Stream API in the ADP Gateway, otherwise the JDBC API is used. In the first case, Master Node employs one or more Optimization Engines which produce a plan defining which worker nodes should be receiving each data stream. In the second case, the Optimization Engines also define how the data should be partitioned (number of partitions, partitioning column, etc.) and where each partition should be stored. The Master Node is notified when the execution plan is ready and then it employs one or more Execution Engines.

Query Execution: In a similar manner, when ADP Gateway receives a query, one or more Optimization Engines produce an execution plan which contains the resulted sequence of operators and the data partition upon which they should be applied. The Optimization Engines report back to the Master Node which then utilizes the Execution Engines who communicate with the Worker Nodes to execute the query. In the case of federated data, some Worker Nodes need to communicate with external databases. They ask queries and get back their results which, depending on the plan, need to be combined with the data that they have locally.

[7] `http://www.sqlite.org`
[8] `https://code.google.com/p/madis/`

When the execution of the query has finished, the Master Node is notified and through the Gateway it can send a message to the external components. The results stay in the Worker Nodes, because the volume of data in the results may be prohibitive for them to be transfered in a single node. When an external component want to access the results, then it must do so by sending an extra request. When receiving such a request, the Master Node uses the Data Connector to collect the results or apply to them some aggregation functions (for example sum, average, etc.).

4 Related Work

The most popular big data platforms today are based on the MapReduce paradigm. MapReduce was introduced by Google [4] as a simplified big data processing platform on large clusters. The intuitive appeal of MapReduce and the availability of platforms such as Hadoop, has also fueled the development of data management platforms that aim at the support of SQL as a query language on top of MapReduce, or are hybrid systems combining MapReduce implementations with existing relational database systems. These platforms attempt to compete with the well-known shared-nothing parallel database systems available from relational DBMS vendors such as Oracle. For example, Hive [13] is a data warehousing system built on top of Hadoop. The Hive query language, HiveQL, is a subset of SQL. For example, it does not support materialized views, and allows subqueries only in the FROM clause. Furthermore, only equality predicates are supported in joins and it only supports UNION ALL (bag union) i.e., duplicates are not eliminated. HadoopDB [1] is a very recent proposal which integrate single-node database functionality with Hadoop in order to provide a highly scalable and fault tolerant distributed database with full SQL support. The U.S. startup Hadapt [9] is currently commercializing HadoopDB. Greenplum by EMC [7] is another commercial platform for big data analysis that is based on a massively parallel database system (shared-nothing architecture) that supports in-database MapReduce capabilities.

In the Semantic Web world, the emphasis recently has been on building scalable systems that offer expressive querying and reasoning capabilities over ontologies expressed in RDFS or OWL 2 and its profiles (EL, QL and RL) and data in RDF. These systems include database platforms for RDF offering the query language SPARQL (Sesame, Jena, Virtuoso, Quest [12], OWLIM, AllegroGraph etc.) and OWL 2 reasoners (Pellet, HermiT etc.) Although recent RDF stores have been shown to scale to billions of triples, the scalability of Semantic Web systems in general is lacking compared with the scalability of more traditional systems such as parallel databases, or newer approaches such as NoSQL databases and parallel databases/MapReduce hybrids. Recent Semantic Web research is also focusing on the use of MapReduce for querying RDF data, but also for forward and backward reasoning with RDFS/OWL 2 ontologies.

To summarise, we believe that the benefits of solutions based on MapReduce are limited and cannot be efficiently extended to more general workloads and more expressive SQL queries such the ones needed in Optique. We believe that by using ADP and the holistic optimization framework of Optique, provide us with a solid foundation upon which to build and go beyond current state of the art platforms for Big Data processing.

5 Conclusions

The efficient execution of SQL queries on big data is an open research problem and initial results achieved by research prototypes such as HadoopDB are encouraging. In the Optique project we will push the barrier and provide massively parallel and elastic solutions for query optimisation and execution over Big Data integration. Our solutions based on ground breaking research will be deployed and evaluated in our use cases. This will provide valuable insights for the application of semantic technologies to Big Data integration problems in industry.

References

1. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D.J., Rasin, A., Silberschatz, A.: HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. PVLDB 2(1), 922–933 (2009)
2. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web 2(1), 43–53 (2011)
3. Crompton, J.: Keynote talk at the W3C Workshop on Semantic Web in Oil & Gas Industry: Houston, TX, USA, 9–10 December (2008), available from http://www.w3.org/2008/12/ogws-slides/Crompton.pdf
4. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008), http://doi.acm.org/10.1145/1327452.1327492
5. DILIGENT: A digital library infrastructure on grid enabled technology (2004), http://diligent.ercim.eu/, http://diligent.ercim.eu/
6. Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Möller, R., Özep, O., Rodriguez Muro, M., Rosati, R., Schlatte, R., Schmidt, M., Soylu, A., Waaler, A.: Scalable End-user Access to Big Data. In: Rajendra Akerkar: Big Data Computing. Florida : Chapman and Hall/CRC. To appear. (2013)
7. Greenplum: "greenplum, http://www.greenplum.com/" (2011), http://www.greenplum.com/
8. Haase, P., Schmidt, M., Schwarte, A.: The information workbench as a self-service platform for linked data applications. In: COLD (2011)
9. Hadapt: "hadapt analytical platform, http://www.hadapt.com/" (2011), http://www.hadapt.com/
10. Health-e-Child: Integrated healthcare platform for european paediatrics (2006), http://www.health-e-child.org/, http://www.health-e-child.org/
11. Kllapi, H., Sitaridi, E., Tsangaris, M.M., Ioannidis, Y.E.: Schedule optimization for data processing flows on the cloud. In: Proc. of SIGMOD. pp. 289–300 (2011)
12. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over dl-lite ontologies. In: KR (2012)
13. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Anthony, S., Liu, H., Murthy, R.: Hive - a petabyte scale data warehouse using Hadoop. pp. 996–1005 (2010)
14. Tsangaris, M.M., Kakaletris, G., Kllapi, H., Papanikos, G., Pentaris, F., Polydoras, P., Sitaridi, E., Stoumpos, V., Ioannidis, Y.E.: Dataflow processing and optimization on grid and cloud infrastructures. IEEE Data Eng. Bull. 32(1), 67–74 (2009)