BrackitMR: Flexible XQuery Processing in MapReduce

Caetano Sauer, Sebastian Bächle, and Theo Härder

University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
{csauer,baechle,haerder}@cs.uni-kl.de

Abstract. We present BrackitMR, a framework that executes XQuery programs over distributed data using MapReduce. The main goal is to provide flexible MapReduce-based data processing with minimal performance penalties. Based on the Brackit query engine, a generic query compilation and optimization infrastructure, our system allows for a transparent integration of multiple data sources, such as XML, JSON, and CSV files, as well as relational databases, NoSQL stores, and lower-level record APIs such as BerkeleyDB.

1 Introduction

The success of query interfaces for MapReduce (MR), such as Hive [4] and Pig [1], indicates the importance of having simple distributed data-processing middlewares, like MR, combined with the expressive power and ease of use of query-processing languages. A breakthrough of such systems is the flexibility provided in comparison with traditional data warehouses and parallel databases. The need for flexibility is particularly strong in the *Big Data* domain, where data is frequently processed in an ad-hoc manner and usually does not follow a strictly relational, previously known schema.

Our approach, based on XQuery, takes the flexibility of MR-based query processing one step further. The starting point is the Brackit query engine, which compiles and optimizes XQuery programs in a storage-independent manner. Its data model extends the original XQuery data model, allowing the system to natively process not only XML structures, but also JSON and relational data. This allows us to plug-in various data sources that reuse the generic compilation and optimization steps performed by the engine. Thanks to a well-defined interface between query processing and storage layers, storage-related optimizations such as predicate and projection push-down are also exploited, thereby minimizing the performance penalty that usually comes as a trade-off for flexibility.

BrackitMR is a component which extends the Brackit query engine to produce MR jobs, transparently benefiting from the optimization techniques and the versatile storage integration. As we demonstrate, such capabilities go beyond the simple raw-file approach of related systems like Pig and Hive, providing wider applicability and better integration with existing infrastructures like DBMSs and data-warehouse systems.

J. Wang et al. (Eds.): WAIM 2013, LNCS 7923, pp. 806–808, 2013.

2 System Overview

Our system is composed of three main components: (i) the Brackit engine, which not only performs query compilation and optimization, but is also responsible for the execution of physical query plans within a single node; (ii) the BrackitMR component, which translates query plans into equivalent MR job specifications; and (iii) the Collection component, which manages multiple data sources that implement a unified interface for retrieving data, mapping into our extended data model, and performing push-down of operations like filters and projections.

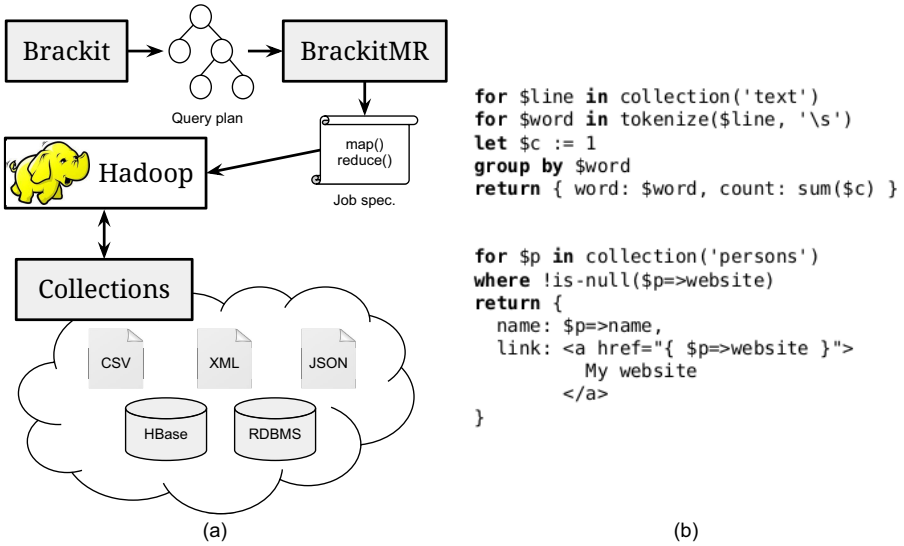

Fig. 1. (a) BrackitMR architecture and (b) a sample query accessing multiple stores

Figure 1a illustrates the three components of our system and how they interact with each other and with the Hadoop system. A query is first compiled and optimized by the Brackit engine, generating a logical query plan which is processed by the BrackitMR component. It splits the query plan into blocks of non-blocking operators which are assigned to Map and Reduce functions, as described in [2]. The technique applied is the same in systems like Hive [4] and Pig [1]. The generated job specifications are then submitted to the Hadoop cluster. When the Map and Reduce functions are executed, the parts of the logical plan contained within them are finally translated into physical operators by a local instance of the Brackit engine. When executed, these operators fetch data from the Collections component, which transparently handles a variety of data sources, producing either XML, JSON, or relational data. Figure 1b shows two sample queries. The top one implements the classical *word count* problem, while

the second one reads data from a JSON data source and produces records containing embedded XML, which is made possible by our extended data model. Both queries produce JSON records as output, which can be either printed on the screen or written to a data format specified by the user in the configuration. Note the use of the `=>` operator, which extracts a field from a JSON record.

3 Demo Description

The system we demonstrate consists of a Web interface which submits queries to a BrackitMR server. The attendees will be able to select from a predefined set of queries as well as to write their own queries. The interface will display the query plans and the steps of transforming it into a MR job. Execution will be monitored and the user will have the option to activate tracing capabilities and debug flags to investigate how data is transformed and shipped across nodes in the Hadoop cluster. The data sources provided will include plain text files, CSV, JSON, relational databases, native XML databases, BerkeleyDB container files, HBase tables, MongoDB collections, and so on. The unified interface of the Collection component allows such sources to be transparently processed within the same query, allowing data from heterogeneous sources to be processed and combined into a single dataset.

Another feature of our interface is benchmarking, which gives the possibility to compare query execution times with optimization features turned on and off. Furthermore, it is possible to compare the performance of the same query running on data stored in different formats, as well as compare the performance of BrackitMR with that of Pig and Hive.

Besides the ability to process data from various sources, BrackitMR is based on the XQuery language, which has a higher expressive power than PigLatin and HiveQL. As discussed in [3], XQuery also fulfills a wider set of flexibility requirements—particularly in the Big Data scenario—than SQL and the mentioned related approaches. Our extended data model, which adds support for JSON, also makes BrackitMR a perfect fit for scenarios in which XML is too cumbersome, but a flexible nested data model is still desired.

References

1. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig Latin: A Not-So-Foreign Language for Data Processing. In: SIGMOD Conference, pp. 1099–1110 (2008)
2. Sauer, C.: XQuery Processing in the MapReduce Framework. Master's thesis, University of Kaiserslautern, Germany (2012)
3. Sauer, C., Härder, T.: Compilation of Query Languages into MapReduce. Datenbank-Spektrum 13(1), 5–15 (2013)
4. Thusoo, A., Sen Sarma, J., Jain, N., Shao, Z., Chakka, P., Zhang, N., Anthony, S., Liu, H., Murthy, R.: Hive – A Petabyte Scale Data Warehouse using Hadoop. In: ICDE Conference, pp. 996–1005 (2010)