Relational versus Non-Relational Database Systems for Data Warehousing

Carlos Ordonez
University of Houston
Houston, TX, USA

Il-Yeol Song
Drexel University
Philadelphia, PA, USA

Carlos Garcia-Alvarado
University of Houston
Houston, TX, USA

ABSTRACT

Relational database systems have been the dominating technology to manage and analyze large data warehouses. Moreover, the ER model, the standard in database design, has a close relationship with the relational model. Recently, there has been a surge of alternative technologies for large scale analytic processing, most of which are not based on the relational model. Out of these proposals, distributed file systems together with MapReduce have become strong competitors to relational database systems to analyze large data sets, exploiting parallel processing. Moreover, there is progress on using MapReduce to evaluate relational queries. With that motivation in mind, this panel will compare pros and cons of each technology for data warehousing and will identify research issues, considering practical aspects like ease of use, programming flexibility and cost; as well as technical aspects like data modeling, storage, hardware, scalability, query processing, fault tolerance and data mining.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; H.2.7 [**Database Management**]: Database administration—*Data Warehousing*

General Terms

Algorithms, Design, Languages, Performance

1. INTRODUCTION

For a long time, relational database systems have been the best technology to manage and analyze large data warehouses. On the other hand, there has been an explosion of semi-structured data on the Internet represented by documents, web pages, images and diverse files, which are stored and queried by search engines. Relational database systems offer extensive functionality to efficiently and reliably update OLTP databases with transactions and to analyze large data warehouses with queries using SQL [2]. However, relational database systems are not flexible and efficient enough to manage and analyze large collections of semi-structured data files, although they offer some basic text processing capabilities. Thus there have been many alternative database technology proposals to solve such limitations [2]. (e.g. XML, streams, column stores, keyword search, content-based image retrieval, and MapReduce, among others). Out of all technologies, MapReduce [1, 5] has gained significant attention, especially to analyze large data sets. MapReduce [1, 5] is a simple parallel programming model coming from functional programming, that enables fast processing on a cluster of computers. MapReduce is typically supported by an efficient distributed file system (DFS), being Hadoop [4, 5] the most popular. Initially, MapReduce was used to perform analysis on large collections of text files. Also, MapReduce has shown to be a good alternative to perform data mining [4] and it is being extended to perform database-like query processing (i.e. SPJA queries). Even further, the Hadoop DFS and MapReduce have been used to develop a data warehouse (e.g. Hive at Facebook [5]). Therefore, it is important to compare features, limitations, advantages and disadvantages of both technologies.

One of the main goals for the panel is to discuss the present and future of data warehousing. Thus the panel will carefully compare DBMS and MapReduce technologies, from practical and technical perspectives, considering modeling, management and analysis of large data warehouses. Also, the panel will identify research issues to further enhance and integrate relational DBMSs and MapReduce.

DOLAP'10, October 30, 2010, Toronto, Ontario, Canada.
ACM 978-1-4503-0383-5/10/10.

2. DISCUSSION SUMMARY

The panel discussion will focus on practical and technical aspects. Practical aspects include technology maturity, ease of programming (e.g. testing and code maintenance), integration with other existing technologies and cost of development. Technical aspects include database modeling, required hardware, system architecture, query evaluation, fault tolerance, OLAP and data mining.

A summary of practical issues follows. Relational DBMSs provide SQL as the standard language to update and query information [2]. SQL computing capabilities can be extended with UDFs [3] and other programming mechanisms, such as stored procedures, embedded SQL and ODBC (or JDBC). MapReduce is available in object-oriented libraries in C++ and Java, out of which Hadoop (Java) is now the most popular. It is fair to say C++/Java are preferred over SQL by most developers. Integrating data mining algorithms into a DBMS is difficult given its complex architecture, unavailability of source code and relational foundation [3]. Most DBMSs offer OLAP and data mining techniques. However, a significant portion of processing happens outside the DBMS. In contrast, MapReduce is an open programming platform, which can be easily extended. From a cost per-

spective, the best parallel DBMSs are commercial, require specialized hardware and are somewhat expensive, whereas MapReduce is open-source and can be implemented with any hardware. On a brighter side, commercial DBMSs offer much more functionality.

We now provide a summary of technical aspects. Most data warehouses are designed with the ER model, complemented by object-oriented software engineering methods like UML. The Extract-Transform-Load (ETL) processes efficiently update the data warehouse with batches of new records. Since ETL builds denormalized tables and transforms existing tables for integration, it is considered a part of data warehouse modeling. ETL is time-consuming and difficult in data warehousing, but easier and faster in MapReduce/DFS. Spatial and temporal dimensions are also incorporated to provide comprehensive database models. Modeling is well studied in relational databases, but remains not well understood in alternative information management technologies. Both parallel DBMSs and the DFS supporting MapReduce, are based on a shared-nothing architecture [5] in which each processing unit (node, process, thread) has its own memory and disk storage; data records are dynamically redistributed among processing units through message passing. Most DBMSs have row-based storage, where a disk block stores a set of rows together. Column-based stores are an innovative alternative to efficiently evaluate analytic queries based on different subsets of columns. In contrast, file systems in search engines (e.g. Yahoo Hadoop DFS, Google GFS) are designed to manage and analyze large collections of diverse semi-structured files (i.e. html web pages, text files, documents and so on). DBMSs provide fault tolerance for transaction and query processing whereas MapReduce provides better fault-tolerance, but only for job processing. A failure in a DBMS may require restarting the query, whereas given a failure in the cluster, MapReduce may continue working without interruption with automatic process migration. DBMSs offer extensive query capabilities, especially with joins (inner, outer) and sophisticated aggregations (e.g. group-by, cubes). There is extensive work on efficiently evaluating cube computations in a DBMS, where the input (fact) table is modeled a multidimensional data set to compute aggregations on multiple dimension combinations. In short, OLAP technology works well with a DBMS. MapReduce can easily evaluate group-by aggregations and recent research shows it is feasible to efficiently compute joins, which are fundamental relational query operators in a DBMS. However, cube computations are not well studied in MapReduce. It is important to emphasize aggregate UDFs accumulate (aggregation detail) and merge phases exhibit fundamental similarities with map() and reduce() functions. Finally, there is extensive work on efficient data mining algorithms, most of which work on flat files outside the DBMS. Moreover, most statistical packages work outside the DBMS, although some can push some demanding computations into the DBMS. We should mention commercial DBMSs offer various data mining algorithms, but they are internally integrated. Therefore, they cannot be customized or extended by the user. There have been alternative proposals to program and optimize data mining algorithms with SQL queries and UDFs [3]. But for the most part, processing data mining inside a DBMS remains a difficult problem.

3. CONCLUSIONS

Relational database systems represent a mature and standard technology in data warehousing, whereas DFS and MapReduce represent a relatively young technology. It is likely SQL will remain the standard language to update and query large databases in the near future. Most relational database systems are basically row stores with tables horizontally partitioned, whereas search engines are based on distributed file systems, where MapReduce uses key-value pairs for processing. MapReduce is simple and efficient to compute aggregations. UDFs represent a promising alternative to extend a DBMS with advanced analytical (OLAP and data mining) capabilities, whose features are similar to MapReduce. DFS/MapReduce are unlikely to substitute the DBMS for data warehousing, although its user base and research will keep growing since it is open-source. Instead, MapReduce can complement DBMSs with flexible and scalable parallel processing for analytic applications.

Given the wide use of DBMSs and the growth of information on the Internet there exist many research issues. SQL together with UDFs (and other DBMS programming mechanisms) will likely remain faster than MapReduce running on the same hardware, but fault-tolerance for large-scale parallel query processing should be improved in a DBMS. UDF support should be improved for more efficient OLAP and data mining processing. The DBMS should support processing of more complex data types (arrays, objects, UDTs). DFS and MapReduce have a simple and flexible parallel system architecture. DBMSs should have new modular architectures, where subsystems can be turned off for analytic applications, reducing system overhead and making management easier. It is likely MapReduce will efficiently evaluate complex relational queries with joins and aggregations, combining it with SQL, as recent research shows. SQL, UDFs and MapReduce should be better integrated to program OLAP and data mining techniques. Fault-tolerance in long-running queries should be improved in a DBMS to avoid restarts. Faster mechanisms for loading large tables of records are needed in MapReduce to match the loading speed provided by DBMSs. In the same vein, DBMSs should provide more flexibility to analyze large external files.

4. REFERENCES

[1] J. Dean and S. Ghemawat. MapReduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.

[2] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison/Wesley, 6th edition, 2010.

[3] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22, 2010.

[4] C. Ordonez and J. García-García. Database systems research on data mining. In *Proc. ACM SIGMOD Conference*, pages 1253–1254, 2010.

[5] M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and parallel DBMSs: friends or foes? *Commun. ACM*, 53(1):64–71, 2010.