Efficient Big Data Processing in Hadoop MapReduce

Jens Dittrich Jorge-Arnulfo Quiané-Ruiz

Information Systems Group
Saarland University
http://infosys.cs.uni-saarland.de

ABSTRACT

This tutorial is motivated by the clear need of many organizations, companies, and researchers to deal with big data volumes efficiently. Examples include web analytics applications, scientific applications, and social networks. A popular data processing engine for big data is Hadoop MapReduce. Early versions of Hadoop MapReduce suffered from severe performance problems. Today, this is becoming history. There are many techniques that can be used with Hadoop MapReduce jobs to boost performance by orders of magnitude. In this tutorial we teach such techniques. First, we will briefly familiarize the audience with Hadoop MapReduce and motivate its use for big data processing. Then, we will focus on different data management techniques, going from job optimization to physical data organization like data layouts and indexes. Throughout this tutorial, we will highlight the similarities and differences between Hadoop MapReduce and Parallel DBMS. Furthermore, we will point out unresolved research problems and open issues.

1. INTRODUCTION

Nowadays, dealing with datasets in the order of terabytes or even petabytes is a reality [24, 23, 19]. Therefore, processing such big datasets in an efficient way is a clear need for many users. In this context, Hadoop MapReduce [6, 1] is a big data processing framework that has rapidly become the de facto standard in both industry and academia [16, 7, 24, 10, 26, 13]. The main reasons of such popularity are the ease-of-use, scalability, and failover properties of Hadoop MapReduce. However, these features come at a price: the performance of Hadoop MapReduce is usually far from the performance of a well-tuned parallel database [21]. Therefore, many research works (from industry and academia) have focused on improving the performance of Hadoop MapReduce jobs in many aspects. For example, researchers have proposed different data layouts [16, 9, 18], join algorithms [3, 5, 20], high-level query languages [10, 13, 24], failover algorithms [22], query optimization techniques [25, 4, 12, 14], and indexing techniques [7, 15, 8]. The latter includes HAIL [8]: an indexing technique presented at this VLDB 2012. It improves the performance of Hadoop MapReduce jobs by up to a factor of 70 — *without* requiring expensive index creation phases. Over the past years researchers have actively studied the different performance problems of Hadoop MapReduce. Unfortunately, users do not always have a deep knowledge on how to efficiently exploit the different techniques.

In this tutorial, we discuss how to reduce the performance gap to well-tuned database systems. We will point out the similarities and differences between the techniques used in Hadoop with those used in parallel databases. In particular, we will highlight research areas that have not yet been exploited. In the following, we present the three parts in which this tutorial will be structured.

2. HADOOP MAPREDUCE

We will focus on *Hadoop MapReduce*, which is the most popular open source implementation of the *MapReduce framework* proposed by Google [6]. Generally speaking, a Hadoop MapReduce job mainly consists of two user-defined functions: *map* and *reduce*. The input of a Hadoop MapReduce job is a set of key-value pairs (k, v) and the map function is called for each of these pairs. The map function produces zero or more intermediate key-value pairs (k', v'). Then, the Hadoop MapReduce framework groups these intermediate key-value pairs by intermediate key k' and calls the reduce function for each group. Finally, the reduce function produces zero or more aggregated results. The beauty of Hadoop MapReduce is that users usually only have to define the map and reduce functions. The framework takes care of everything else such as parallelisation and failover. The Hadoop MapReduce framework utilises a distributed file system to read and write its data. Typically, Hadoop MapReduce uses the *Hadoop Distributed File System (HDFS)*, which is the open source counterpart of the Google File System [11]. Therefore, the I/O performance of a Hadoop MapReduce job strongly depends on HDFS.

In the first part of this tutorial, we will introduce Hadoop MapReduce and HDFS in detail. We will contrast both with parallel databases. In particular, we will show and explain the static physical execution plan of Hadoop MapReduce and how it affects job performance. In this part, we will also survey high level languages that allow users to run jobs even more easily.

3. JOB OPTIMIZATION

One of the major advantages of Hadoop MapReduce is that it allows non-expert users to easily run analytical tasks over big data. Hadoop MapReduce gives users full control on how input datasets are processed. Users code their queries using Java rather than SQL. This makes Hadoop MapReduce easy to use for a larger number of developers: no background in databases is required; only a basic knowledge in Java is required. However, Hadoop MapReduce jobs are far behind parallel databases in their query processing efficiency. Hadoop MapReduce jobs achieve decent performance

through scaling out to very large computing clusters. However, this results in high costs in terms of hardware and power consumption. Therefore, researchers have carried out many research works to effectively adapt the query processing techniques found in parallel databases to the context of Hadoop MapReduce.

In the second part of this tutorial, we will provide an overview of state-of-the-art techniques for optimizing Hadoop MapReduce jobs. We will handle two topics. First, we will survey research works that focus on tuning the configuration parameters of Hadoop MapReduce jobs [4, 12]. Second, we will survey different query optimization techniques for Hadoop MapReduce jobs [25, 14]. Again, we will highlight the differences and similarities with parallel databases.

4. DATA LAYOUTS AND INDEXES

One of the main performance problems with Hadoop MapReduce is its physical data organization including data layouts and indexes.

Data layouts: Hadoop MapReduce jobs often suffer from a row-oriented layout. The disadvantages of row layouts have been thoroughly researched in the context of column stores [2]. However, in a distributed system, a pure column store has severe drawbacks as the data for different columns may reside on different nodes leading to high network costs. Thus, whenever a query references more than one attribute, columns have to be sent through the network in order to merge different attributes values into a row (*tuple reconstruction*). This can significantly decrease the performance of Hadoop MapReduce jobs. Therefore, other, more effective data layouts have been proposed in the literature for Hadoop MapReduce [16, 9, 18].

Indexes: Hadoop MapReduce jobs often also suffer from the lack of appropriate indexes. A number of indexing techniques have been proposed recently [7, 15, 17, 8].

In the third part of this tutorial, we will discuss the above data layouts and indexing techniques in detail.

5. CONCLUSION

We conclude by providing a holistic view on how to leverage state-of-the-art approaches (presented in the first three parts) to significantly improve the performance of Hadoop MapReduce jobs. In particular, we will identify open challenges. In addition, we will sketch a vision on how future Hadoop MapReduce platforms may look like. Finally, we will open the floor for questions on dedicated topics presented in this tutorial.

Target Audience. The first part of this tutorial covers the basics of Hadoop MapReduce. Therefore this part is interesting for all VLDB attendees that want to learn how Hadoop MapReduce can be used for big data analytics. The second and third part of this tutorial are designed for attendees — researchers as well as practitioners — with an interest in performance optimization of Hadoop MapReduce jobs.

Acknowledgments. We would like to thank all students of the seminar *Efficient Parallel Data Processing in MapReduce Workflows* at Saarland University, summer term 2012, for the fruitful discussions. We would like to thank the entire Hadoop++/HAIL team for their feedback and support.

Biographical Sketches

Jens Dittrich (@jensdittrich) is an Associate Professor of Computer Science/Databases at Saarland University, Germany. Previous affiliations include U Marburg, SAP AG, and ETH Zurich. He received an Outrageous Ideas and Vision Paper Award at CIDR 2011, a CS teaching award for database systems, as well as several presentation and science slam awards. His research focuses on fast access to big data.

Jorge-Arnulfo Quiané-Ruiz is a postdoctoral researcher at Saarland University, Germany. Previous affiliations include INRIA and University of Nantes. He was awarded with a Ph.D. fellowship from the Mexican National Council of Technology (CONACyT). He obtained, with highest honors, a M.Sc. in Computer Science from the National Polytechnic Institute of Mexico. His research mainly focuses on big data analytics.

6. REFERENCES

[1] Hadoop, http://hadoop.apache.org/mapreduce/.
[2] D. Abadi et al. Column-Oriented Database Systems. *PVDLB*, 2(2):1664–1665, 2009.
[3] F. N. Afrati and J. D. Ullman. Optimizing Joins in a Map-Reduce Environment. In *EDBT*, pages 99–110, 2010.
[4] S. Babu. Towards automatic optimization of MapReduce programs. In *SOCC*, pages 137–142, 2010.
[5] S. Blanas et al. A Comparison of Join Algorithms for Log Processing in MapReduce. In *SIGMOD*, pages 975–986, 2010.
[6] J. Dean and S. Ghemawat. MapReduce: A Flexible Data Processing Tool. *CACM*, 53(1):72–77, 2010.
[7] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). *PVLDB*, 3(1):519–529, 2010.
[8] J. Dittrich, J.-A. Quiané-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad. Only Aggressive Elephants are Fast Elephants. *PVLDB*, 5, 2012.
[9] A. Floratou et al. Column-Oriented Storage Techniques for MapReduce. *PVLDB*, 4(7):419–429, 2011.
[10] A. Gates et al. Building a HighLevel Dataflow System on Top of MapReduce: The Pig Experience. *PVLDB*, 2(2):1414–1425, 2009.
[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP*, pages 29–43, 2003.
[12] H. Herodotou and S. Babu. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. *PVLDB*, 4(11):1111–1122, 2011.
[13] M. Isard et al. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *EuroSys*, pages 59–72, 2007.
[14] E. Jahani, M. J. Cafarella, and C. Ré. Automatic Optimization for MapReduce Programs. *PVLDB*, 4(6):385–396, 2011.
[15] D. Jiang et al. The Performance of MapReduce: An In-depth Study. *PVLDB*, 3(1-2):472–483, 2010.
[16] A. Jindal, J.-A. Quiané-Ruiz, and J. Dittrich. Trojan Data Layouts: Right Shoes for a Running Elephant. In *SOCC*, 2011.
[17] J. Lin et al. Full-Text Indexing for Optimizing Selection Operations in Large-Scale Data Analytics. *MapReduce Workshop*, 2011.
[18] Y. Lin et al. Llama: Leveraging Columnar Storage for Scalable Join Processing in the MapReduce Framework. In *SIGMOD*, pages 961–972, 2011.
[19] D. Logothetis et al. Stateful Bulk Processing for Incremental Analytics. In *SoCC*, pages 51–62, 2010.
[20] A. Okcan and M. Riedewald. Processing Theta-Joins Using MapReduce. In *SIGMOD*, pages 949–960, 2011.
[21] A. Pavlo et al. A Comparison of Approaches to Large-Scale Data Analysis. In *SIGMOD*, pages 165–178, 2009.
[22] J.-A. Quiané-Ruiz, C. Pinkel, J. Schad, and J. Dittrich. RAFTing MapReduce: Fast Recovery on the RAFT. *ICDE*, pages 589–600, 2011.
[23] A. Thusoo et al. Data Warehousing and Analytics Infrastructure at Facebook. In *SIGMOD*, pages 1013–1020, 2010.
[24] A. Thusoo et al. Hive – A Petabyte Scale Data Warehouse Using Hadoop. In *ICDE*, pages 996–1005, 2010.
[25] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi. Query Optimization for Massively Parallel Data Processing. In *SOCC*, 2011.
[26] M. Zaharia et al. Improving MapReduce Performance in Heterogeneous Environments. In *OSDI*, pages 29–42, 2008.