Scaling out data preprocessing with Hive

Gábor MAKRAI[1], Zoltán PREKOPCSÁK[2]

[1,2]Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, Magyar Tudósok körútja 2., H-1117, Budapest, Hungary

makrai@tmit.bme.hu[1], prekopcsak@tmit.bme.hu[2]

Abstract.

We introduce a user-friendly graphical data preprocessing application based on Hive, one of the well known open-source distributed warehouse systems. It is comfortable, and easy to use for preprocessing purposes, but to prove usability of this application, we created measurement a framework to ensure precise results. These results show that our application has outstanding scaling capability in the case of increasing data amount and increasing number of applied computers. We conclude that this is a good solution for medium and large scale data preprocessing.

Keywords

Hadoop, Hive, large-scale data processing, RapidMiner

1. Introduction

Google [1] encountered a serious problem when its services were becoming worldwide. At the same time with this amazing growth, it had to ensure continuity of its own services, so it needed a reliable, scalable, distributed computational and storage system. It decided to build own system from everyday components, because the cost of maintenance is smaller than the unique ones and in this case reliability is the most important issue. After 2005, Google decided to open the source of its system, and one of the successors is Hive [2], which is a perfect solution for handling very-large scale data. Not only Google encountered this problem [3]. Many companies were suffering from this, especially where historical data has to be stored (telecommunication and financial sector) and where huge amount of data is created through the experiments (bioinformatics, astronomy, and chemical sciences).

Companies need scalable data storing and processing solutions to handle this amount of data [2, 4]. Commercial solutions are very good in optimization, but they are implemented in a single computer approach. This means, if a company want to handle larger database, it has to buy a larger machine. This ”scale-up” philosophy leads to evolution of very expensive high-end server computers. As we mentioned before, Google solved the problem another way [5]: it uses large number of computers to handle data. In that case, if a problem needs larger database, it is enough to add more computers to the current cluster. This is the ”scale-out” philosophy, which gives much cheaper solutions than the previous one.

As Larose mentioned in his book [6], one of the base paradigm of data mining is processing large data. The size of data often larger than a single computer's capacity. In those situations, scale-out solutions can be very useful, because they can use the overall capacity of the given cluster. Often used data mining algorithms can be parallelized (not all of them and sometimes they do not have perfect parallel capabilities).

Today's data mining tools and environments have graphical interface [7] to easily create a complex data mining process. Our goal is to create an user-friendly graphical interface for our application.

In this paper, we introduce our user-friendly graphical application, which can use the strength of Hive to achieve very good performance in data preprocessing. The rest of the paper is structured as follows. In Section 2, we briefly review the history of Hadoop and Hive. Then in Section 3, we describe the precise details of implementation decisions, while in Section 4, we give the results of the measurement. Finally, we conclude and outline some ideas for future work.

2. Background

As previously mentioned, Google developed a distributed system to handle the continuously growing amount of data. It introduced the Google File System [1] in 2003. After that in 2005, it introduced MapReduce [5], which was an innovative approach of distributed computation philosophy. When the implementation of GFS and MapReduce reached the desired level, source codes of MapReduce and GFS were given to Apache Foundation [2], and the Apache Hadoop was born. It contains both previous elements:

- Hadoop Distributed File System: fault tolerant, scalable, simple expandable, highly configurable distributed storage system [2]
- Hadoop MapReduce: software framework for easily writing applications which process vast amounts of data in-parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner [2]

In the world of MapReduce [5, 8, 9], the problem (large input data) splits into many pieces and these pieces are given to map processes. The outputs of these map processes are given to many reduce processes. The reduce processes are the final stage of the execution. They have to create the final results of the problem. Data in input and output of both, map and reduce processes has to be key/value pairs. The framework is able to distribute to data between processing nodes based on value of key field of these pairs. These distributed systems ensure fast software development, because the programmers have to care for only map and reduce programs. Hadoop systems are ready for created map and reduce programs, and they will use every processing node in the cluster to utilize the available computational performance.

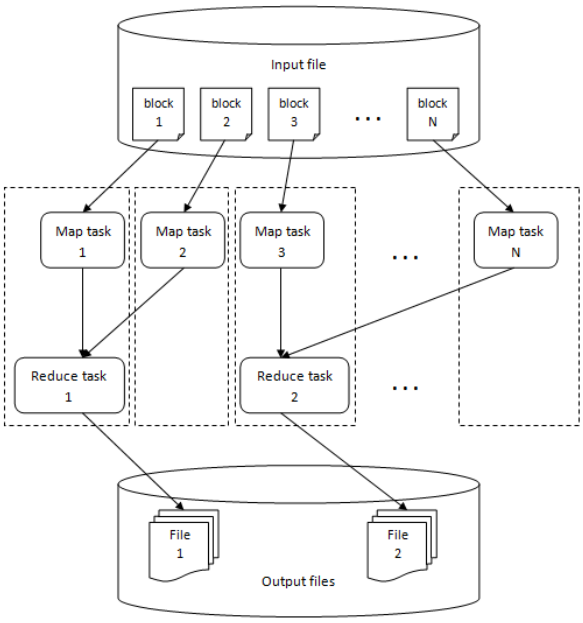

Fig. 1. MapReduce parallel programming model

Hadoop Hive [2] is one of the many Hadoop-related sub-projects which were started by Apache Foundation. It is Hadoop's solution for data warehouses, and it was created to handle very large scale data (hundreds of terabytes). This is a noSQL solution, which means it stores the data different way from the relational database management systems: data tables are stored in plain text file format on the distributed file system (of course, there are many optimization opportunities: compressed file storing method, sequence file in place of text file, etc.). Users of Hive has to work in poor circumstances, because Hive has no graphical user interface.

3. Implementation

In this chapter we introduce implementation details of our application. Main goal of this was to enable user-friendly graphical scale out data preprocessing. Preprocessing part of data mining processes is very important, because data mining algorithms require clean and error-free data. Usually data preprocessing takes more than 60 percent of the job. [6] As Larose mentioned in his book [6], this task consists of the following sub-tasks:

- Filtering fields (table columns) and records (table entries): avoiding obsolete and redundant data
- Handling missing values: replacing NULL values
- Detecting outlier data, which lead to wrong result
- Transforming data to the desired format for applied data mining algorithms

It is an important thing to realize, that most of these functions can be implemented with standard SQL queries, so basically they can be implemented with Hive as it has a query language similar to SQL.

We decided to integrate our extension to an existing data mining environment, because users will be able to use our application in a similar way as they have used this environment before. They do not have to use a new, unfamiliar application to do their work. One of our goal was to suit our application to the existing environment, so much so that the users do not realize the differences between the existing environment, and our application.

We chose RapidMiner[1] for our programming environment, because it is one of the best known open-source data mining tools and it has a clean graphical user interface. It is written in Java, and the source code is well documented, so extensions can be easily created. In RapidMiner, the atomic element is the "operator". There are several kinds of operator groups (import, data transformation, modelling, etc.). Users can create a process in operator flow approach: they can put operators into the process and connect them to each other, so they can describe a complex task with them and their connections. The key of the success of RapidMiner is the professional meta data description of the connections which allows the average user to understand precisely each step of the process. Figure 2 shows a sample process in RapidMiner.

We have implemented the basic transform functions for preprocessing tasks:

- Column filter: select the desired columns
- Data filter: select the desired rows (users can enter expression)

[1] http://rapid-i.com/

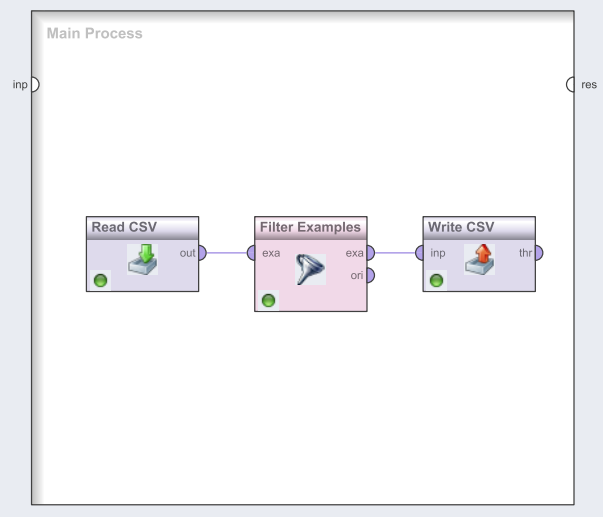

Fig. 2. Sample clustering process in RapidMiner

- Column generation: generate newly calculated columns (users can enter expression)
- Aggregate: aggregate functions (like GROUP BY in SQL)

Finally, we created the test data sets. All of them had the same structure:

- ID column: unique integer identifier
- integer columns: 10 randomly generated integer type columns
- real columns: 10 randomly generated double type columns

We created 7 different size data sets: 128MB (0.5 million records), 256MB (1 million records), 512MB (2 million records), 1GB (4 million records), 2GB (8 million records), 4GB (16 million records) and 8GB (32 million records).

4. Results

In this section, we present our measurement results. This includes performance comparison results between our user-friendly application and the built-in RapidMiner solution, and includes our scalability measurement results of our application. We created three different test scenarios:

- Filtering: select all columns with filtering expression (filter out approximately 50 percent of records)
- Data generation: select four columns (two integer, two double), and create both sum columns
- Aggregate: select two columns (one integer, and one double), and create min, max, and average value grouped by a different integer column

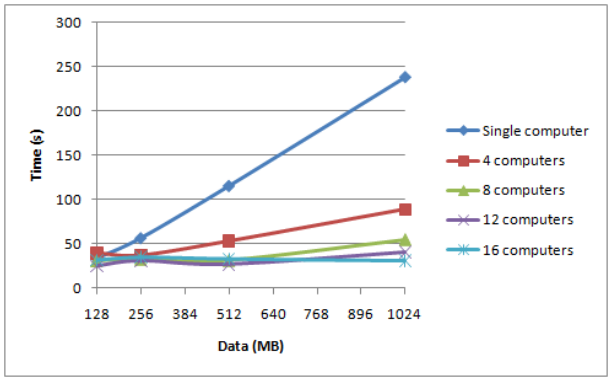

Fig. 3. Filtering on small data sets

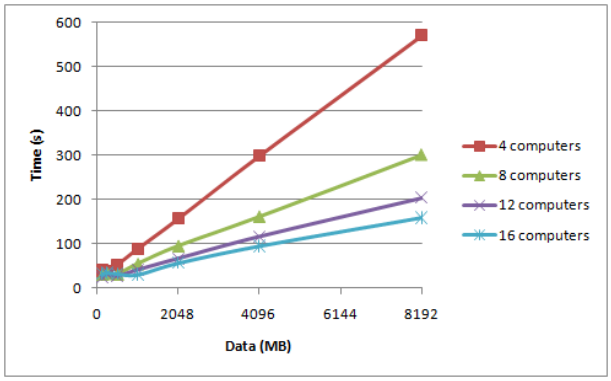

Fig. 4. Filtering on large data sets

Every scenario started with reading data from the available storage system (hard disk or distributed file system), and ended with saving data to the same place.

We used the following test environments to create these results:

- Single computer: this was a server that had large computational and storage performance (Sun Fire X4100 Server), in these cases we used the basic RapidMiner to run the scenarios
- Hadoop clusters: components were computers with medium processing capabilities (AMD desktop computers), in these cases we used our application

Figure 3 and Figure 4 show the results of filtering scenario. As we expected, processing time of our application is inversely proportional to the number of the applied processing computers. Result of the single computer proves, that the processing time strongly depends on the speed of the storage system, because the read and write operations cost much more time than processing the data in memory. We discovered a limitation of RapidMiner in this test scenarios: it cannot handle data over 1 GB, because it tries to fit the whole dataset in the memory.

On the other hand, results of Hadoop clusters show the expected scaling performance (in the meaning of increasing

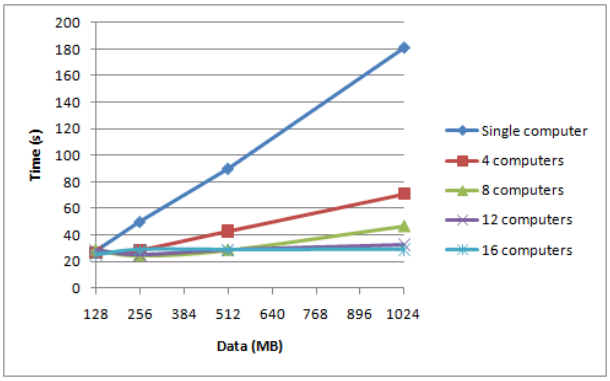

Fig. 5. Data generation on small data sets

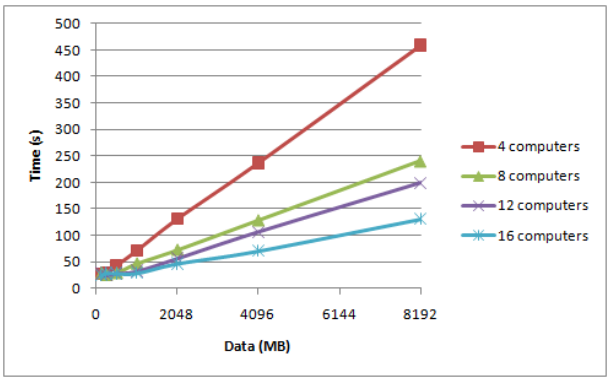

Fig. 6. Data generation on large data sets

amount of data and increasing applied performance capacity) except the cases of operations with small data. In these cases our extension shows constant results. The reason of this behaviour is based on the steps of MapReduce applications: usual inputs of MapReduce (and Hive) tasks are much bigger than these data, and input data splitting algorithm is not optimal for this amount of data, which resulted less distributed tasks than the number of processing units. The size of the smallest test data is 128MB, the default block (this is the atomic element of file storage, the blocks read from and write to the storage in one piece) size in the distributed storage system is 64MB. This means that the splitting algorithm created two tasks for the whole cluster. The smallest cluster contains 4 processing node, so in this case only 50 percent of the computers worked on the problem. With the growth of data amount, this constant processing time changed to the expected linear one. Furthermore, we expected constant initialization time, which takes approximately 10-15 seconds before each MapReduce job. Because of this initialization time, our application is not suitable for real-time data preprocessing, but perfect for off-line data analysis.

Figure 5 and Figure 6 show the results of the next scenario, which is data generation. We experimented the same performance anomaly as the previous scenario: using our solution to solve the small problems resulted constant processing time.

Finally, Figure 7 and Figure 8 show the result of last scenario,which is aggregation. We did not expected any anomaly. Our application had the same performance advantage as we had experienced previously. But we realized a Hadoop specific performance feature. In the first scenario, the output data was approximately 50 percent of the whole data, because of the select statement. In the second scenario this quantity was smaller, because we selected only four columns, and their calculated columns. In the last scenario, the output data was minimal, because of the behaviour of aggregate functions. We expected relatively increasing performance with the decrease of output data compared to the single node performance. This behaviour interconnected with the data storage method of the distributed file system: it uses replicated file storing, so when a file is being saved to it, it will spread the file's blocks in the cluster, which is a slow process, because of the speed of the network.

These test scenarios have proved that our application has good scalable performance and it is usable even if small number of processing computers is used for preprocessing.

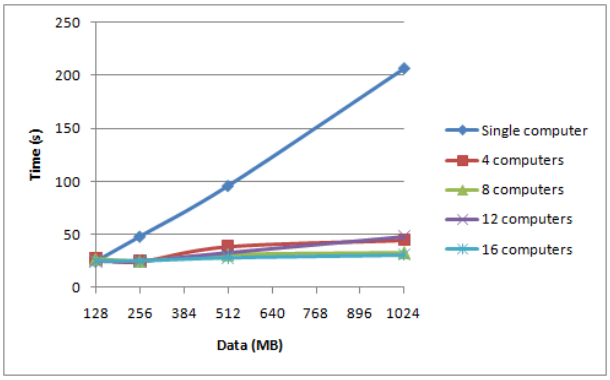

Fig. 7. Aggregate on small data sets

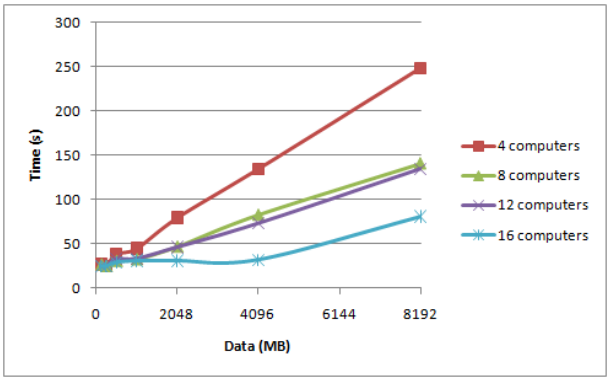

Fig. 8. Aggregate on large data sets

5. Conclusion

We have presented a user-friendly graphical data preprocessing application, which solves the problem in a distributed way. The results show that it can produce good per-

formance in the case of few applied computers and small amount of data, in contempt of the original purpose of Hive. We implemented import, preprocessing and export functions for Hive. Our next task is to extend this application with distributed data mining algorithms, which leads to a fully distributed data mining tool.

Acknowledgements

We would like to thank the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, especially Robert Szabó, PhD, for accessing us to one of the laboratory in our department for measurement purposes. Without this, our comprehensive measurement would have not been possible.

References

[1] S. GHEMAWAT, H. GOBIOFF, S. LEUNG *The Google file system*, SIGOPS Oper. Syst. Rev. 37, 2003.

[2] C. LAM *Hadoop in Action*, Manning, 2010.

[3] T. HEY, S. TANSLEY, K. TOLLE *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Redmond, 2009.

[4] S. OWEN, R. ANIL, T. DUNNING, E. FRIEDMAN *Mahout in Action*, Manning, 2011.

[5] J. DEAN, S. GHEMAWAT *MapReduce: simplified data processing on large clusters*, Communications of the ACM, 2008, 107-113.

[6] D. T. LAROSE *Discovering Knowledge in Data: An Introduction to Data Mining*, Wiley-Interscience, 2004

[7] I. MIERSWA, M. WURST, R. KLINKENBERG, M. SCHOLZ, T. EULER *YALE: Rapid Prototyping for Complex Data Mining Tasks*, Proceedings of the 12th ACM SIGKDD International Conference, 2006

[8] C. RANGER, R. RAGHURAMAN, A. PENMETSA, G. BRADSKI, C. KOZYRAKIS *Evaluating MapReduce for Multi-core and Multiprocessor Systems*, in Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, 2007.

[9] D. GILLICK, A. FARIA, J. DENERO *Mapreduce: Distributed computing for machine learning*, 2008., [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary10.1.1.111.9204

About Authors...

Gábor MAKRAI is a second year software engineering MSc student at Budapest University of Technology and Economics, specializing in infocommunication systems. He got involved in research works of data mining group of his department.

Zoltán PREKOPCSÁK is a PhD student at Budapest University of Technology and Economics. His research topic is pattern classification in time series with applications in human-computer interaction and ubiquitous computing.