Radoop: Analyzing Big Data with RapidMiner and Hadoop

Zoltán Prekopcsák, Gábor Makrai, Tamás Henk,
Csaba Gáspár-Papanek*
Budapest University of Technology and Economics, Hungary

Abstract

Working with large data sets is increasingly common in research and industry. There are some distributed data analytics solutions like Hadoop, that offer high scalability and fault-tolerance, but they usually lack a user interface and only developers can exploit their functionalities. In this paper, we present Radoop, an extension for the RapidMiner data mining tool which provides easy-to-use operators for running distributed processes on Hadoop. We describe integration and development details and provide runtime measurements for several data transformation tasks. We conclude that Radoop is an excellent tool for big data analytics and scales well with increasing data set size and the number of nodes in the cluster.

1 Introduction

Working with big data sets has become increasingly common in many areas. A recent Gartner survey identified data growth as the largest challenge for enterprises [1]. Another study from EMC projects that there will be a 45-fold growth of annual data in the next 10 years [7]. Information and data overload has never been as problematic as nowadays. For many years, storing data was as easy as loading it into a relational database, and advanced data analytics solutions were able to find patterns and extract useful information from those databases.

However, in some areas, data growth has reached the point where a single relational database is not enough. This big data phenomenon has appeared in areas like meteorology, genomics, biological research, Internet search, finance, and many more [10]. The term Big Data is used to describe data sets from a

*prekopcsak, makrai, henk, gaspar @tmit.bme.hu few hundred gigabytes to tens of terrabytes or even more. The usual answer to data growth problems has been to scale up and put more storage and processing power in a single machine, but current computer architectures could not keep up with the growth of storage and processing needs. An alternative approach has appeared which is to scale out to more computers and create a distributed infrastructure of hundreds or even thousands of computers.

Distributed computing is a great promise for handling large data, but it lacks the toolset that we are familiar with on a single machine. It needs new programming paradigms and new tools that we can use for data analytics. There are many such projects which aim to solve efficient data access and provide different data analytics functions in a distributed environment, but they usually need complex command-line mechanisms or even programming to make them work.

In this paper, we present an extension to the RapidMiner data mining suite, which hides all the complexity of distributed data analytics and provides big data processing capabilities in the familiar analytics environment. In Section 2, we present Hadoop and its subprojects, and some related works with RapidMiner. Section 3 describes the integration and development details and Section 4 presents the user interface. We provide some performance measurements in Section 5 and describe future ideas at the end.

2 Background

The first step in the widespread use of distributed data processing was the publication of the Google File System [8] in 2003, and the MapReduce programming paradigm [6] in 2005. The Apache Hadoop project was inspired and built around these ideas and it contains both elements:

- Hadoop Distributed File System: fault-tolerant, scalable, simply expandable, highly configurable distributed storage system [4]
- 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 [11]

In the MapReduce programming paradigm, the input data is splitted into many pieces and these pieces are given to map processes running on different machines. Then the outputs of these map processes are given to many reduce processes which are the final stages of the execution. This model can be seen on Figure 1. All input and output data structures are key/value pairs and the framework is able to distribute data between processing nodes based on the key of these pairs. Working with key/value pairs does not imply any restrictions, because the key and the value can be any object types. MapReduce-based distributed systems ensure fast software development, because the developer only needs to care about map and reduce methods. Hadoop is able to run MapReduce algorithms on unlimited number of processing nodes and it optimizes task distribution in a way that data communication overhead is minimal between the machines. In case of hundreds or thousands of processing nodes, it needs to handle faulty machines and network problems, because these events occur quite often in big clusters.

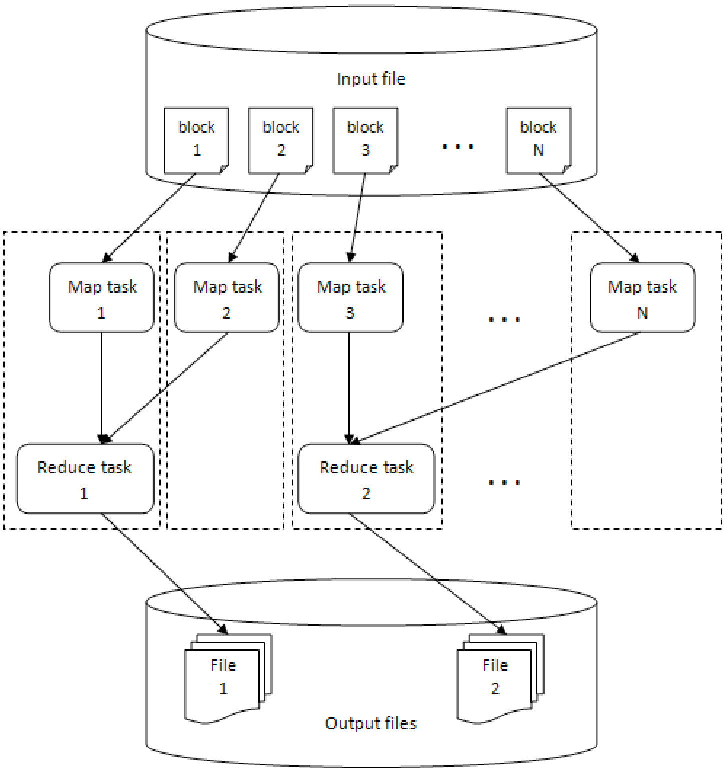


Figure 1: MapReduce parallel programming model

Apache Hive is one of the many Hadoop-related sub-projects. It is the Hadoop solution for data warehousing, and it was created to handle very large scale data (even hundreds of terabytes) efficiently. Hive has a SQL-like query language called HiveQL and a JDBC interface for standardized access, but the most common way of working with Hive is the command line.

There are several other Apache projects in the area of data analytics, and Mahout is the most important from these. Mahout is a collection of scalable machine learning algorithms that run on the Hadoop platform [14]. After the publication of the MapReduce programming paradigm, there has been a significant interest in the machine learning community, and numerous algorithms have been described according to the MapReduce approach [5, 9] and most of these are already implemented in Mahout.

All these Hadoop-related projects are in heavy use at companies working with big data, but they usually lack an easy-to-use interface and they are hard to learn. On the other side, data analytics on a single machine is available for everyone: there are excellent commercial and even open-source solutions like Weka, RapidMiner and KNIME. RapidMiner [13] is one of the most popular and it has a clean user interface that makes it easy to learn. Furthermore, RapidMiner is extendable, so developers can provide additional functionality to the basic software.

In this paper, we present the integration of Hadoop HDFS, Hive and Mahout functionalities in the RapidMiner environment, so these complicated distributed processes can be used with RapidMiner's simple operator flow interface. We are not aware of any general integration project similar to this, but there has been some previous work on integrating several specific Hadoop functions to RapidMiner [2, 3].

3 Integration of RapidMiner and Hadoop

To enable Hadoop integration in RapidMiner, we have created an extension called Radoop. This extension provides additional operators for RapidMiner and it communicates with the Hadoop cluster to run the jobs. We have decided to reuse certain data analytics functions of Hive and Mahout because they are highly optimized. The overall architecture can be seen on Figure 2.

We have designed the extension to achieve a close integration and provide functionalities from Hadoop that are commonly used in memory-based RapidMiner processes. For this, we received aggregated operator usage statistics from Rapid-I and implemented all important operators from the top of the list. We also wanted to keep the convenient RapidMiner features like metadata transformations and breakpoints which make the analysis design process much easier and more error-free.

The creation of a Radoop process begins with adding the RadoopNest meta-operator. It contains general settings for the cluster (like the IP address of the Hadoop master node), and all other Radoop operators can only be used inside this meta-operator. In the following, we first describe data handling and the several possibilities of uploading the data to the cluster. Afterwards, we discuss the data preprocessing and modeling operators of Radoop.

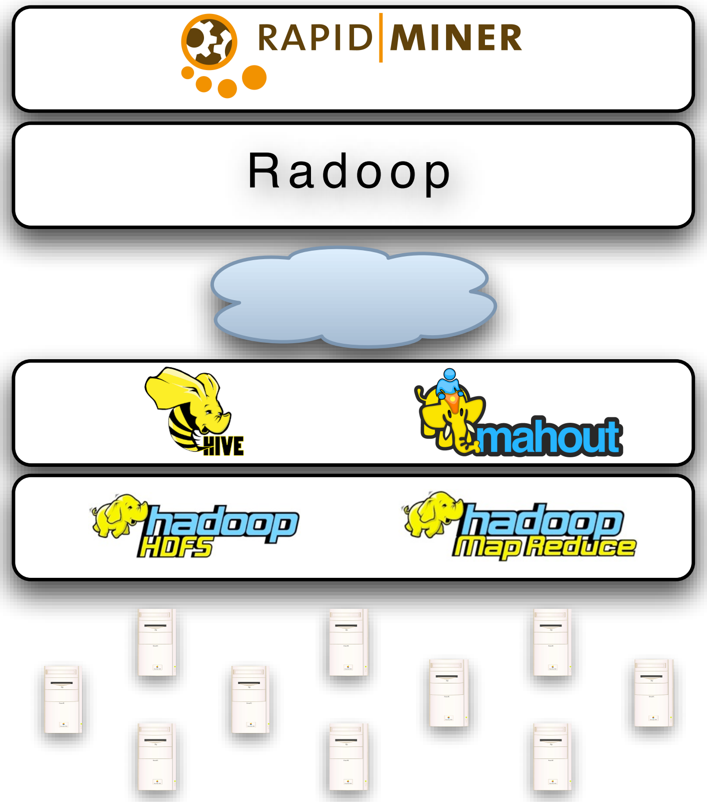


Figure 2: Overall architecture of the RapidMiner-Hadoop integration

3.1 Data handling and I/O integration

In RapidMiner, data tables are ExampleSet objects and normally stored in memory. In Radoop, we store data tables in Hive and we use the HadoopExampleSet object to describe it. It is very important to note that the HadoopExampleSet only stores several pointers and settings, but all data is stored in Hive on the distributed file system, so there is no significant memory consumption during Radoop processes.

We have implemented Reader and Writer operators to enable transferring large data files from right inside RapidMiner. We have overridden basic RapidMiner Reader operators, so the parsing of different data formats are done by the original parsers and then the data is loaded to a Hive table instead of a memory-based ExampleSet. It is the same with Writers, that the output format is created by RapidMiner and we only change the ExampleSet to a HadoopExampleSet as an input. Thanks to this modular process, we support most file formats for read and write operations that RapidMiner has built-in support for.

Reader and Writer operators work with parsing every row in the dataset which has an overhead, so big CSV files can also be uploaded to HDFS and loaded to Hive from the usual command line interface, and we have a Retrieve operator that can access them from RapidMiner. Store and Retrieve are extremely powerful operators in RapidMiner to write and read back intermediate results. We have the same operators in Radoop, so after you have uploaded your input file once over the network, you can use fast Retreive and Store operations to access and save the resulting Hive tables.

Again, all these operators work with a very limited memory footprint, so they can easily handle hundreds of gigabytes or even larger data sets.

3.2 Data transformations

All data miners know that data analytics projects need a lot of effort for preprocessing and cleaning the data. It is usually said that 80% of the work consists of preprocessing and only 20% is modeling and evaluation. RapidMiner has an excellent mechanism to support powerful data transformations by creating views during the process and only materializing the data table in memory when it is needed.

We do the same by using views in HiveQL and only doing expensive data materialization (writing the result table to the distributed file system) when it is needed. It ensures better performance as Hive will only materialize cascaded views and it can also optimize the query.

We support many data transformations that can be expressed as HiveQL scripts. It includes selecting attributes, generating new attributes, filtering examples, sorting, renaming, type conversions, and many more. We also support aggregations and joining tables and it is possible to add more advanced transformations to Hive by user-defined functions (UDF). These operators can be used with the same name and with similar settings to the normal data transformation operators, only the small Hive icons sign that they run on a distributed Hadoop cluster.

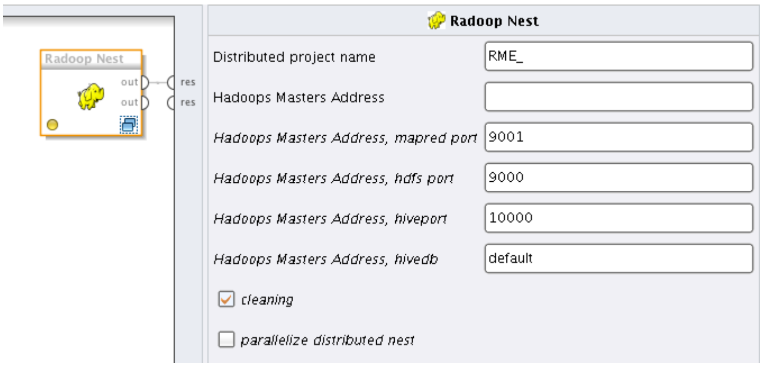
3.3 Data mining and machine learning

Mahout has an increasing number of machine learning algorithms already implemented for Hadoop, so our intention was to integrate it with RapidMiner and use these highly optimized implementations for modeling in Radoop. It turned out that Mahout has a unique way of handling input and output data

[14], so this integration process is more complicated, but we have already managed to integrate the k-means clustering algorithm and more clustering and classification methods (like decision trees and naive Bayes) will be added in the coming months.

4 User Interface

As we have described in Section 3, every Radoop process starts with adding a RadoopNest operator which can be seen on Figure 3. The global settings include a Hive table prefix for the project, the IP address of the Hadoop master node and other port settings. All other operators will run in the nest and use these settings for communicating with the cluster.



Radoop Nest

Distributed project name	RME_
Hadoops Masters Address	
Hadoops Masters Address, mapred port	9001
Hadoops Masters Address, hdfs port	9000
Hadoops Masters Address, hiveport	10000
Hadoops Masters Address, hivedb	default

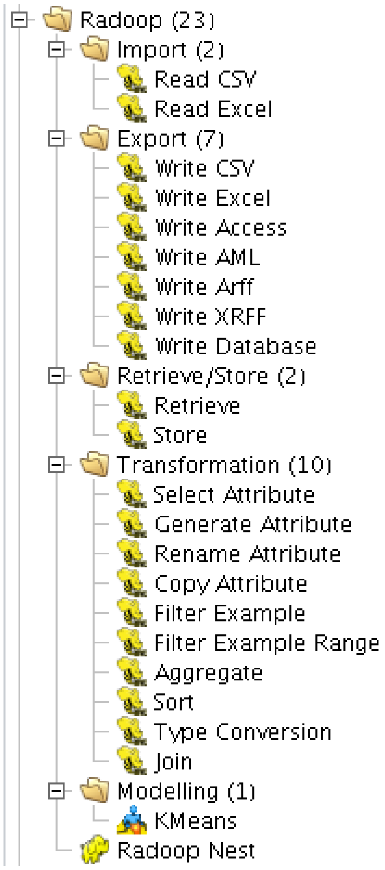
✓ cleaning

☐ parallelize distributed nest

Figure 3: Settings of the RadoopNest operator

All Radoop operators can be accessed from the usual operator panel of RapidMiner, under the Radoop category. These are operators implemented for this project and they can be seen on Figure 4. However, there are other built-in RapidMiner operators which work seamlessly with Radoop. For example, the Multiply operator can create branches, the Subprocess can group operators, or the macro operators can define global variables.

All the integration effort targeted that the user interface of Radoop should be identical to the built-in functionalities of RapidMiner. The process design is the same under the RadoopNest like it is in core RapidMiner, and there are similar operators that can be used on Hadoop.



- Radoop (23)
  - Import (2)
    - Read CSV
    - Read Excel
  - Export (7)
    - Write CSV
    - Write Excel
    - Write Access
    - Write AML
    - Write Arff
    - Write XRFF
    - Write Database
  - Retrieve/Store (2)
    - Retrieve
    - Store
  - Transformation (10)
    - Select Attribute
    - Generate Attribute
    - Rename Attribute
    - Copy Attribute
    - Filter Example
    - Filter Example Range
    - Aggregate
    - Sort
    - Type Conversion
    - Join
  - Modelling (1)
    - KMeans
  - Radoop Nest

Figure 4: Currently available operators of Radoop

5 Performance measurements

We have created measurements on how Radoop performs and scales with the size of the data set and the number of processing nodes. We have used 4 to 16 nodes in the cluster and experimented with 128 MB to 8 GB data sets. We have also included measurements with a single machine RapidAnalytics server solution for reference.

The first experiment retrieves the table from the (distributed) file system, filters out half of the rows, and writes the result back to the (distributed) file system. The same experiment has been run for data sizes of 128 MB to 1 GB with RapidAnalytics and 128 MB to 8 GB with Radoop. The RapidAnalytics experiment was limited to only 1 GB because 2 GB could not fit into the available memory. The runtime results can be seen on Figure 5. Single computer results are from RapidAnalytics and multiple node measurements are from Radoop.

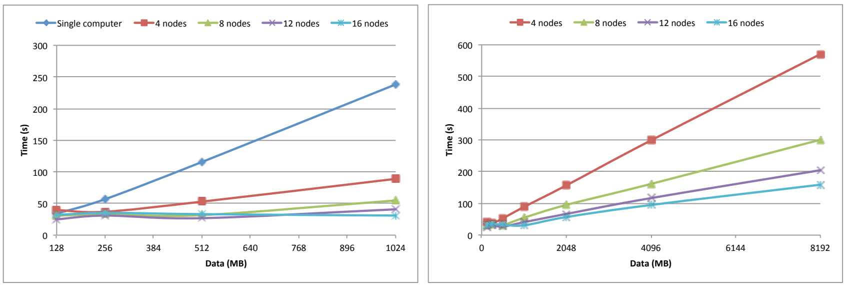


Figure 5: Filter runtime results for small data sets (on the left), and large data sets (on the right)

We can see that both RapidAnalytics and Radoop scales linearly with increasing data size, and Radoop finishes the job much faster, even with 4 processing nodes. The only exception is that Hive has a constant runtime for small data sets. This is caused by the 64 MB default block size in Hadoop and the fact that each block is processed by only one machine. In case of a 128 MB data set, only two nodes are working on the problem, so the constant time that we see is the time needed for one node to process one block. Hence, these results can be improved by changing the default block size.

This enlightens the fact that Hadoop and Hive are not intended for real-time use. Memory-based solutions might perform better on small data sets, but Hadoop has great scalability and the same process will also run in linear time for hundred or thousand times more data.

The second experiment included a more complex process. After loading the data set, we select several attributes, filter out some examples and aggregate according to an attribute. In a web analytics example, it could be a web log from which search robots are filtered out and the number of visits are aggregated for each page. After this, we rename newly created columns and write back the result. The whole process can be seen on Figure 6.

This process includes the creation of four HiveQL views and finally a result table, but the Hive query optimizer is able to express it as a single HiveQL query and the result is acquired with only one stage of map and reduce jobs. The runtime results on Figure 7 show that RapidAnalytics is quite slow on a single machine and Radoop can process 8 times more data on 8 nodes with approximately the same runtime. It shows that task distribution has a minimal

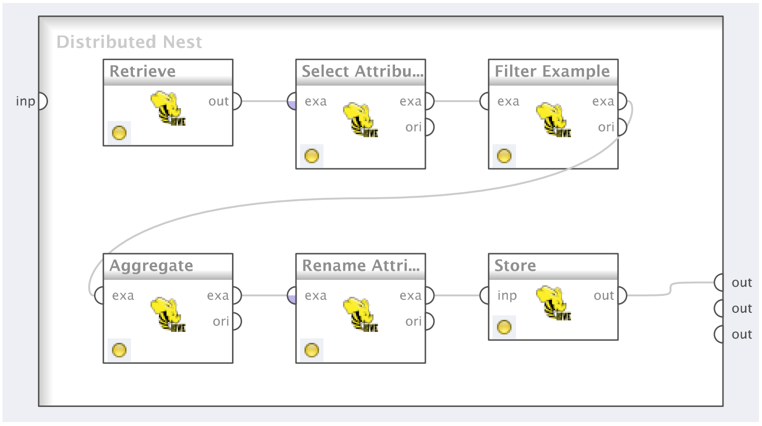


Figure 6: Operator flow of a Radoop process

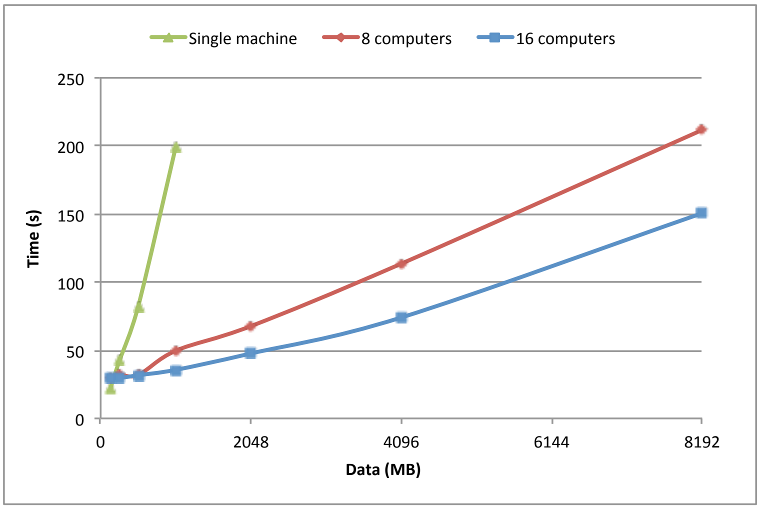


Figure 7: Runtime results for a complex data transformation process

overhead, and Radoop can be a good choice even for data sets of a few hundred megabytes.

Further measurements with Hive are available in our previous work [12].

6 Conclusion and future work

We have presented Radoop, a Hadoop extension for the popular RapidMiner data mining tool. The complexities of running jobs on a distributed cluster are covered and the close integration keeps the excellent process design and validation features of RapidMiner.

Our measurements show that Radoop scales well with increasing data size and increasing number of Hadoop processing nodes, so it is possible to effectively analyze data far beyond the size of main memory. However, performance gain over the single machine solution can be seen even for data sets of a few hundred megabytes and even with only 4-8 processing nodes.

Future work will include new data transformation operators, additional clustering and classification algorithms from Mahout, some improvements on the user interface, and a monitoring panel which will enable the investigation of long running Hadoop processes. The beta version of Radoop will be available at `http://radoop.eu`.

References

[1] A. Adams and N. Mishra. User Survey Analysis: Key Trends Shaping the Future of Data Center Infrastructure Through 2011. Technical report, Gartner, 2010.

[2] A. Arimond. A Distributed System for Pattern Recognition and Machine Learning. Master's thesis, TU Kaiserslautern and DFKI, 2010.

[3] A. Arimond, C. Kofler, and F. Shafait. Distributed Pattern Recognition in RapidMiner. In *Proceedings of RapidMiner Community Meeting And Conference*, 2010.

[4] D. Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 2007.

[5] C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, page 281. The MIT Press, 2007.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] J. Gantz and D. Reinsel. The Digital Universe Decade - Are You Ready? Technical report, EMC Corporation, 2010.

[8] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[9] D. Gillick, A. Faria, and J. DeNero. Mapreduce: Distributed computing for machine learning, 2006.

[10] A.J.G. Hey, S. Tansley, and K.M. Tolle. *The fourth paradigm: data-intensive scientific discovery.* Microsoft Research, 2009.

[11] C. Lam and J. Warren. *Hadoop in Action.* Manning Publications, 2010.

[12] G. Makrai and Z. Prekopcsák. Scaling out data preprocessing with Hive. In *Proceedings of the 15th International Student Conference on Electrical Engineering*, 2011.

[13] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940. ACM, 2006.

[14] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action.* Manning Publications, 2011.