2009 IEEE International Conference on Data Mining Workshops

Toolkit-based high-performance Data Mining of large Data on MapReduce Clusters

Dennis Wegener, Michael Mock, Deyaa Adranale, Stefan Wrobel
Fraunhofer Institute Intelligent Analysis and Information Systems IAIS
Schloss Birlinghoven, 53754 Sankt Augustin, Germany
{*dennis.wegener, michael.mock, stefan.wrobel*}*@iais.fraunhofer.de, d.adrnalee@gmail.com*

***Abstract*—The enormous growth of data in a variety of applications has increased the need for high performance data mining based on distributed environments. However, standard data mining toolkits per se do not allow the usage of computing clusters. The success of MapReduce for analyzing large data has raised a general interest in applying this model to other, data intensive applications. Unfortunately current research has not lead to an integration of GUI based data mining toolkits with distributed file system based MapReduce systems. This paper defines novel principles for modeling and design of the user interface, the storage model and the computational model necessary for the integration of such systems. Additionally, it introduces a novel system architecture for interactive GUI based data mining of large data on clusters based on MapReduce that overcomes the limitations of data mining toolkits. As an empirical demonstration we show an implementation based on Weka and Hadoop.**

***Keywords*-MapReduce; Hadoop; Weka; Data Mining Toolkit;**

I. Introduction

The ever-increasing amount of electronically available data offers tremendous new opportunities for automated analysis. Data mining tasks will have to deal with increasingly large datasets, which exceed the capacity of familiar data mining toolkits with respect to memory consumption and performance. There is a clear need for high performance and distributed data mining for speeding up data mining processes as well as enabling scenarios with large data at all. On one hand, the Google File System [1] and the MapReduce [2] approach have proved to be very successful for handling the analysis of large data in parallel on huge clusters of computers. On the other hand, data mining toolkits such as Weka [3] or RapidMiner [4] facilitate and accelerate the interactive data mining process. They provide a convenient user interface with ready to use access to a library of implemented data mining algorithms which can be freely combined, and the possibility for extension with new algorithms.

Unfortunately, common toolkits are designed to run on single-computer systems as sequential applications on small to medium size datasets, and struggle with large datasets due to large memory consumption and long execution times. Although extensions to some of these toolkits allow them to work on clusters [5], [6], [7], [8], there is currently no GUI supported toolkit that allows for the analysis of very large data in an interactive way.

Previous research has left the question unanswered on how to integrate data mining toolkits such as Weka with MapReduce frameworks based on distributed file systems such as Hadoop [9]. In face of a multitude of contrasting design principles of these two systems, the integration of the user interface, the storage model and the computational model are challenging. Thus, new principles need to be defined on how to model the control and data flow regarding the contrasting user interfaces, how to design the data representation and management that allows access from the data mining toolkit side as well as from the MapReduce framework side, and how to modify the algorithms of data mining toolkits to allow the usage of MapReduce for data access and processing.

The contribution of this paper is a novel system architecture for interactive GUI based data mining of large data on MapReduce clusters that is defined according to new principles for addressing the above mentioned problems. Our architecture maintains the features of the toolkit character, namely a ready to use environment, including commonly known algorithms with a GUI users are used to, and the support of extensibility. The latter allows for the easy integration of further parallelized algorithms, supporting the reuse of the code of the data mining toolkit as far as possible. To empirically demonstrate our architecture, we show how the Weka data mining toolkit and the Hadoop system can be integrated and present first performance results.

The paper is organized as follows: Section II gives an overview over related work. In Section III, our approach to the integration, the design decisions and the overall architecture are presented. Section IV shows how data mining algorithms can be integrated and parallelized with our approach and gives examples. We present the experimental evaluation in Section V and our conclusions in Section VI.

II. Related Work

In recent years, there have been a lot of attempts to parallelize individual data mining algorithms. In contrast, our work focuses on a general toolkit framework for parallel data mining processes that simplifies using and adding new parallel algorithms based on MapReduce. Several attempts have tried to integrate Weka in a distributed environment and to parallelize some of Weka's operations like model evaluation and cross validation. Weka Parallel [5] extends

978-0-7695-3902-7/09 $26.00 © 2009 IEEE
DOI 10.1109/ICDMW.2009.34

Weka by parallel cross validation on multiple machines. In contrast, our approach provides parallelization in term of asynchronous job submission, for different types of model evaluation including cross validation, and algorithm internal parallelism. GridWeka [6] extends Weka by the capability of distributing tasks on multiple computing resources. Similarly, WekaG [8] aims also at adapting Weka to a grid environment. In our case, w.r.t. the algorithm internal parallelization, load balancing, resource planning, fault-tolerance etc. are managed by the Hadoop system, which is superior in many of these topics and outperforms many custom approaches. Weka4WS [7] aims at integrating Weka in a grid environment. Parallelization is only reached by asynchronous job submission, while we aim at parallelization in terms of job submission, model evaluation and algorithm internal parallelization. In addition, individual Weka processes in Weka4WS are running on a single machine which becomes problematic for large data.

However, among these adaptations of Weka, we have not found any successful attempt to support background job submission, model evaluation and parallel algorithms for very large data. All these attempts seem to work with the full-in-memory dataset and thus have not solved the memory consumption problem of Weka with large datasets. Apart from that, these approaches do not support MapReduce and Hadoop, and thus cannot profit from the advantages the MapReduce programming model and Hadoop provide.

On the other hand, parallelizing data mining algorithms using the MapReduce model has received significant attention from the research community since the introduction of the model by Google. In [10], the MapReduce model based on Hadoop is examined for applicability in the field of data mining. It is shown that several classes of data mining algorithms fit very well with the MapReduce programming model. The work in [11] shows, based on a shared-memory implementation for multi-core machines, how data mining algorithms can be applied to the MapReduce model if they are written as a *summation form*, a sum over expressions of the data instances. The Apache Mahout Project [12], based on [11], is aiming at providing a collection of machine learning algorithms for Hadoop.

However, all these attempts do not aim at toolkit integration and thus lack in advantages provided by a toolkit such as a ready to use environment supported by a GUI and extensibility mechanisms well known to data miners.

III. Architecture

In this section, we describe the design challenges and introduce how our system design addresses the integration of such data mining toolkits and MapReduce frameworks regarding their different models of user interaction and data storage. The computational model will be addressed in Section IV.

Integrating data mining toolkits such as Weka with MapReduce frameworks based on distributed file systems such as Hadoop is a challenge that requires well-balanced modeling and design decisions regarding different user interface, storage and computational models. The basic approach of our system design is to let a Data Mining Client process start Data Mining Server processes asynchronously on the cluster or cloud and to let these Data Mining Server processes have access to a MapReduce framework. For integrating the divergent computational models of data mining toolkits and the MapReduce programming model, we design our system to support parallelization in terms of background job submission, parallel model evaluation and parallelism inside of data-mining algorithms. In the data mining algorithm executed by the Data Mining Server, parallelizable subtasks are identified for different classes of data-mining algorithms that are executed in parallel with MapReduce. As the Data Mining Server also contains a complete instance of the data mining toolkit library, a re-use of the toolkit code for non-parallelizable parts is possible.

The overall architecture of our approach to integrate data mining toolkits with MapReduce frameworks is depicted in Fig. 1. The key element of our approach is to run several (extended) instances of the data mining toolkit library. One instance runs on the client side (in the following denoted as ”Data Mining Client”), offering the familiar GUI. For each job submission, another instance is started on the cluster side (denoted as ”Data Mining Server”), that communicates with the Data Mining Client and serves as entry point to the cluster. In our implementation, the data mining toolkit client interface (Weka Explorer) is extended to handle background job submission, i.e., the interface does not block when a job is submitted and results and messages of ongoing background job executions are captured and visualized asynchronously in the interface. Each submitted job creates a Data Mining Server process in the cluster, which executes the data mining toolkit library extended with access to a MapReduce framework (Hadoop). When the user submits a job from the Data Mining Client's graphical interfaces, the job is sent as a command to the cluster, e.g. using an SSH connection. The created Data Mining Server process will run the sequential part of algorithm code and submit jobs to the MapReduce framework for the parallel part. It can make use of the built-in data mining library whenever appropriate, allowing for potential reuse of the toolkit code for the sequential part. After the job is finished, its results are stored as a file in the master's (local) file system and transferred back to the client's file system, e.g. using a SCP connection, where it can be read and displayed.

For a smooth integration of the **user interface model**, we face the problem that data mining toolkit has a synchronous GUI while the MapReduce framework provides command-line and web-based interfaces only. In our solution, we extend the known Weka Explorer in the Data Mining Client

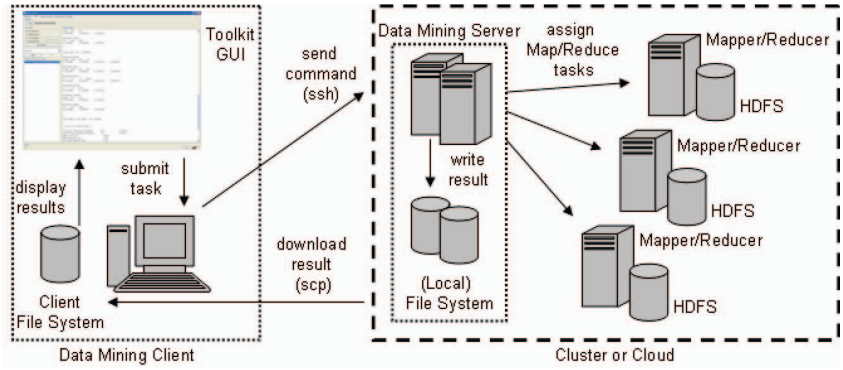


Figure 1. System architecture for integrating data mining toolkits with MapReduce frameworks on clusters and clouds.

with facilities for job submission to the Data Mining Server, hence bridging the gap between the sequential GUI on the client side and the computing cluster running Hadoop on the other side. In addition, the GUI in the Data Mining Client is extended to allow the user to select data from the distributed file system and to run a MapReduce based analysis on Hadoop on server side. As in the original version of Weka, the same interfaces of Weka's Explorer are used to allow the user to select datasets and submit learning jobs for building and evaluating models.

Regarding the **storage model** of our system, we have to define a principle on how to integrate RAM-based, random access to data on the toolkit side with sequential, file based access on the MapReduce framework side, while allowing a re-use of toolkit code for the sequential part of the algorithms. Our approach is to replace the instances representing data in the Data Mining Server with toolkit-conformant references to data instances in the MapReduce framework, such that data stored in the distributed file system on the cluster can be accessed from the Data mining Server. Furthermore, additional data management functions for transferring data to/from the cluster are integrated in the GUI in the Data Mining Client.

In our implementation, the user can select data from the same data sources of the original Weka Explorer: the local's file system, a database query, or a remote URL location. However, the data is not physically loaded into main memory, but transferred to the cluster's HDFS and then opened in the Explorer without loading all the instances in memory. In addition, data stored directly on the HDFS can be selected. In the original Weka Explorer, when the user opens a dataset, all of its attributes and instances are loaded into a single object of the class *Instances*. The object contains two lists, one for information on the attributes and one for the instances of the dataset. Each instance (of the type *Instance*) carries the values of the attributes. The list of *Instance* objects will be very large for large datasets and will be the cause for the memory consumption problem. In our solution, we store the dataset on the cluster's HDFS and we accomplish this by instantiating an object of the type *HadoopInstances* which extends the class *Instances*, stores the path of the dataset on the HDFS, and overrides the access to the list of instances by accessing the HDFS dataset instead. As *HadoopInstances* is a subclass of *Instances*, the generic Weka functions for dealing with instances such as for example *Instances.numClasses* can also be applied to *HadoopInstances*, as long as they do not access the values of the data items. Hence, the original Weka code can remain unchanged in the sequential part of the algorithm.

IV. Parallelization of the Data Mining Process

As motivated before, we designed our system to support parallelization in terms of background job submission, model evaluation and inside data mining algorithms. In this section, we focus on the principles for the integration of the **computational model** of data mining toolkits and MapReduce frameworks and show how the data mining process and the algorithms can be parallelized in our approach.

A. General Approach

One issue in the integration of the computational models are the different concepts of data processing. The data structures in Weka can contain arbitrary dependencies, while Hadoop only supports independent instance passing. In our approach, we split the algorithms into non-parallel parts that still can work with dependencies in memory and parallel parts that process data independently. Another issue is the way algorithms are represented. Algorithms in Weka do not have any constraints in terms of the type of the algorithm, but all of them are sequential. In Hadoop, algorithms have to follow the MapReduce principle and thus are parallel by design. In the data mining processes, parallelizable subtasks are identified that can run in parallel on Hadoop. These processes include several algorithms for different classes of data mining problems as well as evaluation tasks. On the cluster side, the Data Mining Server version of Weka is used such that a re-use of Weka code for non-parallelizable parts is possible. Technical details like resource discovery, resource selection, scheduling, load balancing etc. for the parallel parts are all handled by the Hadoop system.

Our architecture allows to execute Data Mining Server processes asynchronously on the cluster and to let these processes access the MapReduce framework on the cluster. Regarding **asynchronous job submission**, the GUI of the Data Mining Client, which is the Explorer of Weka in our implementation, allows running multiple jobs simultaneously on the same cluster. Any parallelized data mining algorithm and any corresponding model evaluation can be submitted as background job.

In addition, the **model evaluation** itself can be easily parallelized using MapReduce, as each instance can be tested independently. Hence, it is sufficient to let each parallel map function evaluate the model on the given instance while summing up the evaluation statistics in the reduce phase. This parallel model evaluation is provided in our system in form of testing based on the training set, a test set, percentage-split or parallel cross-validation, which can be applied to any type of parallelized data mining algorithm. Since in our implementation the specification for each model is encapsulated inside the map function by calling *classifyInstance* / *clusterInstance*, the only requirement on the model is that it can be serialized in order to be given as a parameter to the MapReduce job.

In the following, we go into the details of the **parallelization and integration of data mining algorithms** based on Weka and Hadoop. First, parallelizable and non-parallelizable tasks inside the algorithms have to be identified. In order to parallelize an algorithm using the MapReduce model, the parts of the algorithm that accesses the input data must be written as MapReduce jobs. While to do so, each input instance must be accessed independently of the others. The general characteristics and limitation of the algorithms parallelizable in our approach is that only the parallelizable subtasks are able to access all data instances, whereas the sequential part works on aggregated values only. Many data mining algorithms can be parallelized using the MapReduce model [10], [11], [12]. In the following we present a case study on how to integrate the Naïve Bayes algorithm into our system. In particular, we split the algorithm into sequential and parallel parts (being executed by Hadoop) and show how our approach supports the re-use of Weka code for the sequential part. Further algorithms, such as K-Means, Linear Regression and Subgroup Discovery, have been parallelized and integrated with our approach [13].

B. Case study - Naïve Bayes

We refer to the implementation of the Naïve Bayes algorithm as found in the Weka implementation [3] and to its parallelization with MapReduce as for example presented in [11]. We assume the reader to be familiar with the Naïve Bayes algorithm, which uses the Bayes formula to calculate the probability of the class value given a set of attribute values under the assumption that the attributes are independent from each other given the class label.

The Weka implementation *NaiveBayesSimple* of the Naïve Bayes algorithm operates by counting the occurrences of the class values and storing them in an integer array `Priors` of length k. It also counts the occurrences of attribute values and stores them in a three-dimensional $(k \times p \times v_j)$-matrix `Counts` (the third dimension is implemented with the dynamic array facility of Java). Both counts are generated within a single linear scan over the data set, in which each iteration step inspects a single data item and increments the appropriate counts in `Priors` and `Counts` according to the actual values of the data item. After this counting loop, a normalization step and the computation of the probabilities of each class attribute are executed based on the values stored in `Priors` and `Counts`.

In the algorithm, the counting loop incrementing the values for the variables `Counts` and `Priors` can be easily parallelized, since it operates independently on individual data items, while the normalization step that follows the counting must be executed sequentially after that because it requires the computed values of these variables. Since in our approach the Data Mining Server contains the complete Weka-library and has access to Hadoop, it is sufficient to replace the counting loop by a call to the appropriate Hadoop map and reduce tasks. Instead of counting the occurrences in a sequential loop on *Instances* objects in memory as in the original code, in the parallel version a MapReduce job is started that works on data stored in the HDFS which is referenced by *HadoopInstances*. This job counts the occurrences and stores the results in a file. After the Hadoop job is finished, the result is read and its values are stored in the variables `Counts` and `Priors`, such that the original Weka code can go on in the Data Mining Server process. Fig. 2 depicts the original and the modified code.

It remains to be explained how the values for the variables `Counts` and `Priors` can be computed in parallel with Hadoop. All map functions get an individual instance as input. Based on the class and attribute values of the instance, they identify the indexes of the elements of `Counts` and `Priors` that have to be incremented. These indexes are then used as keys for the input to the reduce step, together with the increment value of 1. For each input instance the Mapper outputs for every attribute <(attribute,value,class_value), 1> and for the class attribute <class_val, 1>. The Reducer gets these pairs and the list of values and outputs the sum for each different key.

V. EVALUATION

In this section, we evaluate the performance of our approach based on the implemented example. Apart from proof of concept, we want to answer by the experiments where exactly we will end at a position between the stand alone Weka version on a single machine and a solution based on Hadoop in terms of performance. In [11] the performance of machine learning algorithms based on Hadoop in a shared

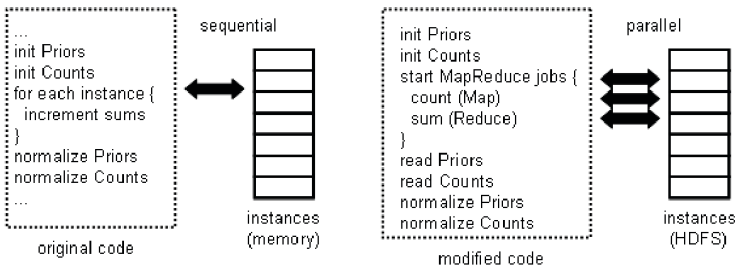


Figure 2. Original and modified Weka code of Naïve Bayes.

main memory environment is analyzed. In this solution, the learning algorithms can be parallelized to the maximum. Of course, we can not expect to compete with a shared memory parallelization because our solution includes network overhead and disk access. In addition, we also consider code sections for independent processing which cannot be parallelized. But, our solution should perform better than just a single computer. Additionally, the shared memory solution of course has no ”toolkit aspects” (e.g., like user friendliness, extensibility) and it also does not scale with clusters. Thus, for very large amounts of data this solution might not perform well.

In the following, we consider the nearly fully parallelizable implementation of Naïve Bayes with varying sizes of input data. Besides considering speedup, we also we want to identify the basic overhead introduced by our architecture and the effects of the memory limitation encountered in the original Weka toolkit. The distributed experiments have been performed on a cluster of 10 Linux machines connected via 1 GBit LAN. Each machine had the following specifications: AMD-Opteron 2.2 GHz Dual CPU, 8 GB RAM, 2 x 110 GB local disks (one reserved for Linux and the other used by Hadoop). Related sequential experiments using the original Weka have been performed on one of these machines. The datasets used during the experiments were randomly generated using the generation feature in the Weka Explorer (developer version 3.5.8). The generated datasets were of different number of instances and had 24 Boolean attributes and a nominal class attribute with values 0, 1,..., 9.

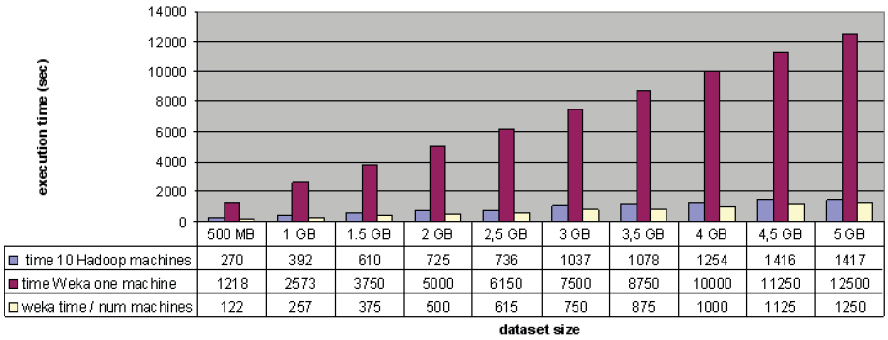
The experiments of the distributed Naïve Bayes implementation have been performed on datasets of different sizes from a few kilobytes to the size of 100 GB. Each run of the learning task included model building and evaluation on the same dataset. We compare the performance against the referencing sequential implementation of Weka. We detected that this implementation could not run the Naïve Bayes algorithm on the dataset larger than a size of 1 GB on our machines with 8 GB main memory. Due to this memory limitation, we extrapolated the sequential runtimes for larger data sets simply by assuming linear scaleup of the runtime. In addition, we compare against a hypothetical fully parallel solution assuming we had optimal speedup of 10 for the original Weka experiment.

Our experiments showed that, on small datasets of a few kilobytes, the sequential Weka implementation significantly outperforms the Hadoop implementation. As the dataset gets larger, the sequential implementation starts being slow and the distributed execution on Hadoop starts performing better. The break even for the parallelization can be recognized at a data size of about 100 MB.

On large datasets the parallel implementation scaled very well and confirmed that runtimes were increasing linearly with the data size. This was expected as model building and testing each require just one pass on the data and because the performance of Naïve Bayes algorithm is clearly I/O bound. Fig. 3 compares the measured performance against a hypothetical fully parallelized Weka version that simply assumes that it performs ten times faster than the extrapolated sequential Weka version (hence assuming unlimited main memory and 100% parallelizability). We note that our results approach this hypothetical speedup with increasing data size. Further testing on very large data showed a runtime of 25861 seconds on 100 GB input data which is pretty close to the hypothetical best case of 25000 seconds.

The runtime of the Naïve Bayes experiment on the smallest datasets is about 32 seconds for training and testing, which includes two rounds of Hadoop job submissions on the cluster side. For a further breakdown of the overhead, we analyzed the execution times for these experiments on the cluster side only. We found that the cluster side Hadoop overhead for a single Hadoop invocation is around 13 seconds at minimum. Of course, this overhead depends on the number of Mappers (which depends on the data size). However, since our experiments from Fig. 3 show that the total execution times increase linearly as excepted, the cluster side overhead does not significantly increase for large data. The measurements also allowed to derive the overhead incurred by the client server communication in our architecture, which consists of job submission and retrieval of results. We measured a constant overhead of 3 seconds. Hence, the basic overhead incurred by our architecture is about 16 seconds for model building and about 32 seconds for a job that builds and evaluates a model.

For results of further experiments with other algorithms, e.g. K-Means, Linear Regression and Subgroup Discovery, we refer to [13].



	500 MB	1 GB	1.5 GB	2 GB	2,5 GB	3 GB	3,5 GB	4 GB	4,5 GB	5 GB
time 10 Hadoop machines	270	392	610	725	736	1037	1078	1254	1416	1417
time Weka one machine	1218	2573	3750	5000	6150	7500	8750	10000	11250	12500
weka time / num machines	122	257	375	500	615	750	875	1000	1125	1250

Figure 3. Naïve Bayes experiments on large data.

VI. Conclusion and Outlook

In this paper we have presented a novel system architecture for interactive data mining of large data on MapReduce clusters based on new principles for modeling and design of the user interface, the storage model and the computational model of the integrated system. As empirical demonstration, we have shown how the Weka data mining toolkit can be integrated with a Hadoop cluster for distributed data storage and parallel processing allowing the use on large datasets. Our experiments on a 10-node cluster have shown good speedup over sequential implementations for datasets of sizes starting from a few hundred megabytes. The overhead of our system was identified to be around 16 seconds. Hence, our system is not useful for computations that Weka can perform alone within this time. However, we have seen that Weka struggles with serious memory limitations when considering large input data sets. For example, the Naïve Bayes implementation of Weka was limited to handle data sets of 1 GB on a machine with 8 GB of main memory. Even with this input size that could be handled by Weka, our parallel version was significantly faster, and it scaled up to input sizes of 100 GB.

The architecture allows for distributed and high performance data mining while keeping the way of interaction of a known GUI. In our approach, dynamically created Data Mining Server processes serve as entry point into the cluster and execute the sequential part of the data mining algorithm. However, we learned that if this sequential part is computing intensive, the machine of the cluster running the Data Mining Server processes can have quite big load. A solution would be to instantiate the Data Mining Server processes on different machines in the cluster managed by a load-balancing system, as e.g. done in the GridR toolkit for the seamless integration of R with cluster and grid computing [14].

Acknowledgments

The authors acknowledge the financial support of the European Commission for the Project ACGT, FP6/2004/IST-026996, and thank Prof. Edgar Nett, University of Magdeburg, for his kind support of this work.

References

[1] Ghemawat, S., Gobioff, H., Leung, S., *The Google File System*, In: 19th ACM Symposium on Operating Systems Principles, Lake George, NY (2003)

[2] Dean, J., Ghemawat, S., *MapReduce: Simplified data processing on large clusters*, Sixth Symposium on Operating Systems Design and Implementation, pp. 137–149, San Francisco, CA, USA (2004)

[3] Witten, I.H., Frank, E., *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco (2005)

[4] Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T., *YALE: Rapid Prototyping for Complex Data Mining Tasks*, In: 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 935-940, ACM, New York (2006)

[5] Celis, S., Musicant, D.R., *Weka-parallel: machine learning in parallel*, Technical report, Carleton College, CS TR (2002)

[6] Khoussainov, R., Zuo, X., Kushmerick, N., *Grid-enabled Weka: A Toolkit for Machine Learning on the Grid*, ERCIM News, No. 59 (2004)

[7] Talia, D., Trunfio, P., Verta, O., *Weka4WS: a WSRF-enabled Weka Toolkit for Distributed Data Mining on Grids*, In: 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal (2005)

[8] Perez, M.S., Sanchez, A., Herrero, P., Robles, V., Pea, Jos, M., *Adapting the Weka Data Mining Toolkit to a Grid Based Environment*, Advances in Web Intelligence, pp. 492–497 (2005)

[9] The Apache Hadoop Project, http://hadoop.apache.org

[10] Gillick, D., Faria, A., DeNero, J., *MapReduce: Distributed Computing for Machine Learning*, Berkeley (2006)

[11] Chu, C., Kim, S.K., Lin, Y., Yu, Y., Bradski, G., Ng, A., Olukotun, K., *Map-Reduce for Machine Learning on Multicore*, Neural Information Processing Systems, pp. 281–288, MIT Press (2006)

[12] The Apache Mahout Project, http://lucene.apache.org/mahout/

[13] Adranale, D., *Parallelization of the Weka Data Mining Toolkit on Hadoop*, Master Thesis, Department of Distributed Systems, Faculty for Informatics, Otto-von-Guericke University of Magdeburg (2008)

[14] Wegener, D., Sengstag, T., Sfakianakis, S., Rüping, S., Assi, A., *GridR: An R-based grid-enabled tool for data analysis in ACGT clinico-genomic trials*, In: 3rd IEEE International Conference on e-Science and Grid Computing, pp. 228–235, Bangalore, India (2007)