Extensibility of Grid-Enabled Data Mining Platforms: A Case Study

Dennis Wegener
Fraunhofer IAIS
Schloss Birlinghoven
53754 St. Augustin, Germany
+49 2241 14 2261
dennis.wegener@iais.fraunhofer.de

Michael May
Fraunhofer IAIS
Schloss Birlinghoven
53754 St. Augustin, Germany
+49 2241 14 2039
michael.may@iais.fraunhofer.de

ABSTRACT
In this paper, we discuss requirements for a distributed data mining platform, putting the requirement of extensibility in the focus. We describe the extensibility of the DataMiningGrid system and give a case study where we integrate several new algorithms of the Weka data mining suite into the grid environment. Using these algorithms on a regression problem, we evaluate the system's performance. Additionally we compare the extensibility with that offered by several other platforms. We conclude that DataMiningGrid offers a very flexible environment for integration of third party Data Mining algorithms, and that this flexibility does not come at the price of a large performance overhead.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Distributed applications – *grid-enabled applications*; D.2.11 [Software Architectures]: Domain-specific architectures – *service oriented architectures.*

General Terms
Algorithms, Management, Measurement, Performance, Design

Keywords
Grid, Service Oriented Architecture (SOA), Data Mining, DataMiningGrid, Weka

1. INTRODUCTION
Data mining algorithms are known to be computationally intensive, making scalability a key issue. Distributed data mining has emerged as an area of research that addresses this issue. A second issue is that data is sometimes inherently distributed and cannot be merged in an easy way, e.g. due to organizational barriers or the amount of data involved.

If one or both of these problems occurs, Grid computing [6] appears as a promising candidate technology to apply. It is able to connect single workstations or entire clusters and provides facilities for data transport and sharing across organizational boundaries. In recent years, a number of environments for grid-enabling data mining tools have been described [1,2-5,7,14,16]. Many of those environments have been developed independently and at the same time. A comprehensive review and comparison of these alternative architectures is provided in [14]. One interesting result is that by now we witness some trend to converge on a set of standard technologies, including Globus Toolkit, WSRF on the Grid side and Weka on the Data Mining side.

A data-mining expert will typically have the following set of criteria to make his choice among the available platforms.

1. The application should become more scalable and/or it should be able to handle distributed data, possibly over organizational boundaries including firewalls. (This, of course, is the main reason to use Grid technology from the application side.)
2. It should be an easy task for the data miner to extend the environment by his own tools. No modifications of the original data mining code should be necessary, since this code may be unavailable.
3. It should not be necessary to programmatically modify the Grid environment, neither on the server nor on the client side.
4. Since the data mining expert is usually not a Grid technology expert, the Grid should remain transparent to the user.
5. The environment should be based on standard technology.

In this paper, we argue that requirements 2 and 3 – jointly called the *extensibility requirement* – are a crucial requirement for a successful uptake of grid-enabled data mining platforms, and that it has been only partially addressed by most existing grid-enabled data mining platforms.

Most platforms seem to assume availability of a dedicated platform developer responsible for extending the platform by new algorithms. From this perspective, it is important that the developer has an easy task in integrating previously existing algorithms for which no source-code might be available. But it is assumed that the developer is responsible to modify *the platform itself*, e.g. by adding new GUI components and modifying server and client side source code.

While this assumption might be reasonable for more traditional data mining environments, it is often unrealistic in a grid setting. A main motivation for a data miner in using grid-technology is to scale up *pre-existing* specialized algorithms so that they can take advantage of grid infrastructure for distributed computing and data sharing. Those specialized algorithms either have been implemented by the data mining expert himself; or they provide domain-specific functionality (e.g. in bioinformatics); or their use is enforced by company policies.

Often, no dedicated grid platform developer willing to integrate those specialized algorithms will be available. But the data miner is normally not in a position to modify the source code of the grid environment itself; it might be unavailable and the learning curve for making the modifications would be too high anyway.

A grid-enabled data mining platform meeting all the criteria set out above at the same time is not easy to design. This comes from the fact that Grid technology is extremely complex and data mining tools can appear in all sorts of languages and formats. Matching the variety of formats with the complexity of the Grid, multiplied by the flexibility of various data distribution strategies, may in fact suggest that a system matching all this criteria is almost impossible.

In this paper, we argue that the DataMiningGrid [14] platform stands out among the various grid-enabled data mining platforms in meeting the requirements above in a clean design and by providing a user friendly web interface to integrate new algorithms into the grid. The system is fully implemented and distributed as open source.

The rest of this paper is organized as follows. In Sec. 2 we sketch the architecture of the DataMiningGrid. In Sec. 3 we describe an extension of the DataMiningGrid platform by three different algorithms – kNN, Model Trees and Locally Weighted Learning, taken from the Weak Data Mining suite [18]. We test scalability and performance using a Grid environment comprising two clusters in two different organizations in Sec. 4 and show that the flexibility of the system does not result in a significant overhead. Finally, in Sec. 5 we compare the DataMiningGrid platform with a set of alternative Grid-enabled Data Mining platforms.

This paper complements and extends previous work by putting the extensibility requirement of grid-enabled data mining platforms at center stage and showing in a detailed case study that the DataMiningGrid platform meets the above requirements.

This study is partially motivated by the ACGT project, which aims at developing an open-source grid-enabled biomedical IT infrastructure to provide tools needed to integrate complex clinical information in the context of biomedical informatics research [11]. The ACGT infrastructure has strong requirements with respect to extensibility.

2. RELATED WORK

The importance of extensibility for data mining platforms has already been argued in [19]. Today there are a lot of systems which are capable of distributed data mining. While in [14] a general comparison of a variety of systems has been done we focus on the extensibility requirement in this paper. Looking at the literature, extensibility is addressed in different ways.

GridMiner [3] is designed to support data mining and online-analytical processing (OLAP) in distributed computing environments. The system is based on a service oriented architecture (SOA) supporting OGSA grid services and OGSA-DAI database access. GridMiner implements a number of common data mining algorithms, including parallel versions, and also text mining tasks. A major difference between GridMiner and DataMiningGrid related to extensibility is that in the GridMiner

Figure 1: DataMiningGrid Application Enabler

system, as well as e.g. in SODDM [5], each data mining application seems to be integrated as single service. This means that for each new algorithm there is a need for programming a new service.

The Federated Analysis Environment for Heterogeneous Intelligent Mining [2] (FAEHIM) implements a toolkit for grid based data mining. It consists of data mining grid services and a workflow engine for service composition. Based on algorithms taken from Weka, the grid-services split into the types classification, clustering and association rules. Weka4WS [16] is a framework for supporting distributed data mining on Grid environments, designed by using the Web Service Resource Framework (WSRF) to achieve integration and interoperability with standard grid environments. Among others, both systems are based on a special data mining suite, namely Weka. This implies that the extensibility of these systems is restricted to these algorithms.

Anteater [7] is a service-oriented architecture (SOA) for data mining that is based on Web services to achieve extensibility and interoperability but does not support grid standards such as WSRF or OGSA. The system provides the capability to distributed fine-grained parallel data mining applications, based on a runtime system called Anthill. However, Anteater requires data mining applications to be converted into a filter-stream structure. While this feature greatly increases scalability, this approach might limit the number of applications that will actually be ported to this platform because of the need for a special customization of every algorithm to be included.

The Platform Independent Text Mining Engine Tool (Pimiento) [1] is an object-oriented application framework (OOAF) for text mining. The Pimiento OOAF aims to hide the complexity implicit in a text mining engine, the scalable management of text documents, or access control when developing text mining applications, facing amongst others the following key requirements: An open architecture and open interfaces should ensure the interoperability of the platform and text-mining applications, a modular approach shall avoid the need for changes on the applications in case of changes on the platforms functionality and new applications should not affect the existing environment and developers without requiring in-depth knowledge on text mining should be able to add text mining functionalities to their applications.

The Pimiento OOAF, which can be seen as an application "template" implemented in Java Standard Edition, provides advantages such as modularity, reusability, extensibility, and inversion of control. The different text mining tools implemented include functions for text categorization, language identification, clustering and similarity analysis. However, Pimiento is focused basically on text mining including distributed tasks but does not seem to address grid aspects as e.g. resource brokering.

Knowledge Grid (K-Grid) [4] is a high-level service-oriented framework that has been designed to provide grid-based data mining tools and services. The system facilitates data mining and related tasks such as data management and knowledge representation. The system architecture is organized in different layers: The Core K-Grid Services handle the publication and discovery of data sources, data mining and visualization tools, and mining results as well as the management of abstract execution plans that describe complex data mining processes. The High-level K-Grid Services are responsible for resource describing metadata, the mapping of resource requests from the execution plans to the available resources in the grid and the task execution. While the development of the DataMiningGrid was driven by the requirements of a different set of scenarios, the design of K-Grid was focused on a more conceptual view of the knowledge-discovery process. Also K-Grid seems to be not available as open source in the time of writing.

Discovery Net [13] provides a service-oriented computing model for knowledge discovery, focused on scientific discovery from high-throughput data generated in life science, geo-hazard and environmental domains that allows to access and use third party data analysis software and data sources. Based on Globus Toolkit the system provides components to declare the properties of analysis tools and data stores, to integrate various data sources (e.g. SQL-Databases, OGSA_DAI sources etc.), to discover and compose Knowledge Discovery Services, to integrate data from different data sources using XML, and to deploy knowledge discovery processes as new services. The Discovery Net system seems not to support WSRF and it is not clear how e.g. the resource brokering in the grid is addressed.

For the lastly mentioned systems it is not always clearly stated how the inclusion of new algorithms is done in the systems, e.g. Pimiento [1] and DiscoveryNet [13]. The KnowledgeGrid [4] if combined with the Vega front-end seems to be the most similar system to DataMiningGrid related to the extensibility but may differ in other aspects. One such aspect is that DataMiningGrid is available as OpenSource. Also, we did not find any description of a functionality similar to the DataMiningGrid's web-based interface for grid-enabling applications (see Figure 1).

3. DATAMININGGRID

The DataMiningGrid project (2004-2006) is a shared cost Strategic Targeted Research Project (STREP) granted by the European Commission (grant no. IST-2004-004475) as part of the Sixth Framework Programme of the Information Society Technologies Programme (IST). The main project outcome is a platform consisting of tools and services for deploying data mining applications on the grid. The software is freely available under the Apache License 2.0.

The DataMiningGrid system is designed to meet the requirements of modern and distributed data mining scenarios. The system is a service-oriented architecture (SOA) which is based on open source technology like Globus Toolkit and is compliant to the common standards Open Grid Service Architecture (OGSA) and the Web Services Resource Framework (WSRF).

The architecture of the DataMiningGrid system is described in [14]. The system is based on a layered architecture which consists of 4 layers: Hardware & Software Resources, Grid middleware, DataMiningGrid High-Level Services and DataMiningGrid application Client layer.

The Hardware and Software resource layer refers to the machines and/or clusters in the grid as well as the grid-enabled applications and the data they use. The components of the grid middleware Globus Toolkit, which are used by the DataMiningGrid system and the DataMiningGrid extensions, are depicted in the grid middleware layer. This layer also contains an enhanced Condor Adaptor which allows connecting local Condor [10] pools to the grid environment. The High-Level Services layer consists of the components Resource Broker, Information Services and Data Services. These provide the functionality of resource brokering [9], providing information about available applications and the

data management inside the system. The client layer refers to the client side components of the DataMiningGrid system as the Triana Workflow Editor and Manager [17] and web based clients. The main user interface is Triana, which was extended by components for interoperability with the DataMiningGrid environment. Details on the application's runtime can be found in [9].

A typical DataMiningGrid grid environment consists of a *head* site that runs unique middleware and high-level services (e.g. the Resource Broker service, central MDS4 service, Information Integrator service), several *central* sites that run the GRAM service and manage the aggregated resources and a lot of *worker* machines orchestrated e.g. by a local scheduler like Condor.

In the DataMiningGrid environment an application is defined as a stand alone application which is executable (e.g., C, Python, Java) by command line. A central feature of the system is the DataMiningGrid Application Description Schema (ADS), which is used to grid-enable applications in an easy way and helps to discover applications in the grid, create user interfaces dynamically and generate the GT4 job descriptions. The schema is an XML schema for describing the executable application in order to define how it is used with the system. Each grid-enabled application refers to a particular instance of the ADS that are passed, at different levels of specification, between the system components to manage interaction. In detail, general information about the application (metadata like name, textual description etc), execution information (executable file, programming language, required libraries etc.) and application information (the number and type of the applications options and data in- and outputs, the minimum resource requirements etc.) has to be specified when grid enabling the applications. When executing the application within a workflow the users specify additional information like the values for parameters, the specific in- and output files or directories, the execution machine if the algorithm is to be shipped to a special machine, etc).

In order to grid-enable an application the executable has to be uploaded to the grid and an instance of the ADS describing the executable has to be created. This can either be done by manually creating the application description file locally and uploading it together with the executable to the grid or by using the DataMiningGrid Application Enabler (Figure 1), a web page that guides the user through the process of grid enabling. The process of grid-enabling is shown in the following section.

Triana, as the main user interface for the DataMiningGrid environment, is used for composition and execution of complex data mining workflows. A component inside a Triana workflow, which can be sees as wrapper component that refers to special operations, is called unit. In order to allow to access and interact with the DataMiningGrid grid environment, Triana was extended by a set of those components. Aim of the graphical tool is to allow to set up even complex distributed data mining workflows in a user-friendly way, hiding much of the complexity of grid technology.

The DataMiningGrid Triana units are grouped, together with the standard Triana units, in a tree-like structure. They are organized in several subgroups referring to their functionality, e.g. the Applications package which contains units for application discovery and selection, the Data Resources package that is responsible for accessing various data sources, the Execution package containing units for the execution and monitoring of the applications, the Provenance package with units for getting provenance info about the application's execution and the Security containing units for creating proxy certificates.

A workflow for running a Weka application that will be used for the experiments (Figure 2) consists e.g. of the following six units:

- **LoadDescription**: This unit is from the *Applications* package. Unlike the unit **ApplicationExplorer** from the same package, which is used to browse the grid wide application registry, it loads an application description file from the client machine. For our experiments this is faster than browsing the grid registry for the application and does not require user interaction through the process of application selection. Additionally, in a local application description file, we are able to set the default parameter setting to our needs so that less customization of the applications parameters is necessary at the **ParameterControl** unit.
- **GridURI**: This unit specifies the URI of the file in the grid which is used as input file for the selected application. This is faster than browsing the grid, e.g. by the unit **GridExplorer,** for the input file and does not require user interaction.
- **ParameterControl**: This unit is used for the specification of the applications parameters, which are in detail the options, in- and outputs and requirements. For the parameters sweeps can be specified.
- **Execution**: The execution unit is used for specifying the applications output dir and the execution of the application itself.
- **Provenance**: The provenance unit shows provenance information about the applications execution. This is information about the runtime, the machines used etc. The provenance information can be stored in an XML-file which is later used for analysis and the generation of the runtime figures.
- **GridExplorer**: The grid explorer is used for browsing the applications result directory, which contains the application's output file as well as the standard out and the standard err.

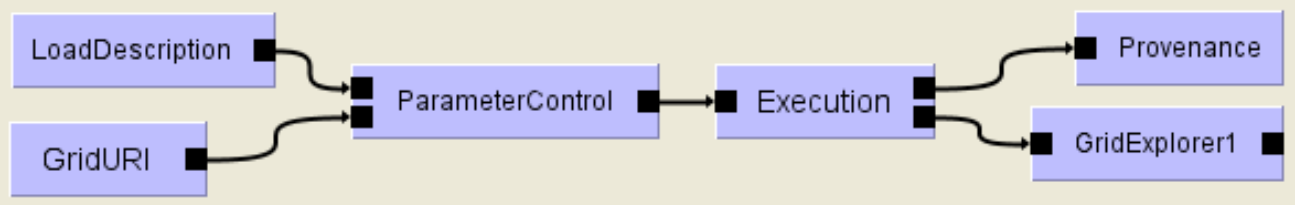

Figure 2: Weka Workflow

It would also be possible to specify the input file, parameters etc. directly in the local application description file. This would result in a fully specified application description which can directly be used as input for the execution unit which would make the units **GridURI** and **ParameterControl** redundant. This way of specifying the applications parameters is less user friendly but faster in workflow execution once the applications are set up because it does not need any user interaction before the job submission at all.

4. GRID ENABLING WEKA-ALGORITHMS

This section describes the grid-enabling of some Weka algorithms from the data mining expert's or user's perspective. The following subsections will introduce the Weka suite and the algorithms which are going to be grid enabled as well as the process of grid enabling.

4.1 Weka

Weka [18] is a comprehensive Data Mining toolbox written in Java. It is available as Open Source and is in widespread use especially in the academic community. It has modules for pre-processing, classification, regression, clustering, feature selection and visualization. Weka is equipped with a set of user interfaces, but the individual components can also be run in command-line mode. We use this second option.

We use Weka 3.5.5 in our experiments to allow easy reproducibility of our experiments, as well as a performance comparison with other Grid-enabled Data Mining platforms. We stress the fact that the DataMiningGrid platform can integrate algorithms in all sorts of formats and is not limited to Weka.

In our case study, we focus on a regression problem. We use the following Weka algorithms for our experiments:

4.1.1 K-nearest neighbors classifier (IBK)

K-Nearest-Neighbors [18] is a well-known, simple yet powerful method both for classification and regression. For predicting an unknown instance the *k* nearest instances, according to some distance function are selected and, for regression, the target value is calculated using some possibly distance-weighted mean of the nearest neighbors. Crucial parameters are the correct choice of the distance function, the weighting function and the number of neighbors.

4.1.2 Locally weighted learning (LWL)

Locally weighted learning [18] is an instance-based algorithm that assigns weights to instances according to the distance to the test instance. This is similar to kNN, but not limited to a fixed number of neighbors. LWL performs a linear or non-linear regression on the weighted data, using the weighted instances.

4.1.3 M5P

M5P [18] is a tree-based algorithm. In contrast to a regression tree, which uses the mean value in the leafs of a tree for prediction, a linear model is fitted for each leaf.

All three methods have proven to be powerful and robust when applied to real-world data. Each of them is able to capture the local structure of a data set.

4.2 Process of grid enabling

The Data Miner's application to be made available in the grid has to be compliant to the definition of an application in the DataMiningGrid environment (see section 2). If the application that is to be grid enabled supports a flag/value format it can directly be described with an application description file. Otherwise a wrapper has to be provided that translates a flag/value command line call into the format of the specific algorithm.

As Weka does not use flag/value pairs when specifying parameters by command line and Weka command line calls do not execute the algorithms directly but call them through an evaluation class we need a wrapper component which is responsible for the transformation of the command line calls. The Weka source code does not have to be customized. The wrapper component for the Weka algorithms is the class which is called by the grid job and has to be capable of the following parameters:

- The class name of the algorithm to execute
- The name/path of the input file
- The class attribute to predict
- The name/path of the output file to create
- The number of cross validation runs which shall be performed
- The options of the algorithm

As a convenience, we have provided a standard wrapper for Weka, which does the translation automatically for many Weka algorithms. This is flexible, because there is no need for a specific translation for each different algorithm. Even the Weka version can be changed. The standard wrapper has the disadvantage that it is not possible to specify parameters for sub-classes for some algorithms.

The wrapper, together with the Weka distribution, is packaged into a jar file which is the executable file compliant with the DataMiningGrid environment.

In order to make the three algorithms available in the grid we now have to follow the procedure of grid enabling it and create the application description files. These files are instances of the ADS and contain the following description for each application: The information in the element *application type* is Data Mining specific metadata about the application like the application's name (which is the algorithms name) the group (Weka), the application domain (Data Mining), the CrispDMPhase (Modeling) etc. The element *general information* contains further metadata such as *version*, *id*, a *description*, the *upload date* and so on. In the *execution* element we have to specify execution type (java), the main class (e.g. weka.Wrapper), interpreter arguments (e.g. the maximum java heap size -Xmx1000m) and the application run file (path to the jar file). The element *application information* contains information about the options of the application (which are the options which can be specified in the Weka GUI) as well as the number of cross validation folds, the class attribute to predict and a hidden option specifying the algorithm's class name. Each of these options is specified by data type, default value, a tooltip, the flag, a label shown in the GUI etc. Additionally it specifies the data input, which is a single file in the arff format, the data output, which is a text file containing the textual output

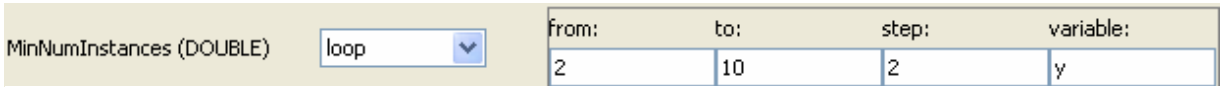

Figure 3: Parameter sweep - loop

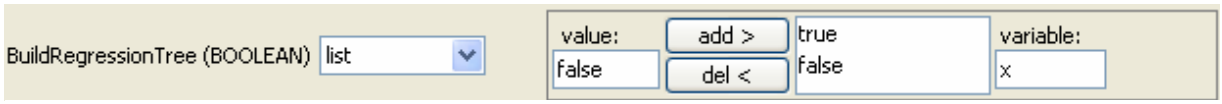

Figure 4: Parameter sweep - list

Weka creates and the system requirements (which are not set for the Weka algorithms).

The last step is to upload the executable and the application description files to the designated place in the grid. Once the application is grid-enabled it appears in the grid wide application registry. (It is left to the service provider to define proper access restrictions for this functionality; it can also be disabled).

5. EXPERIMENTS

In the following we will perform different experiments with the grid enabled Weka algorithms in the grid. We show two simple experiments with the execution of a single algorithm with different parameter settings and one more complex scenario where we test different classifiers on the same input data.

5.1 The Workflows

At first we will run two experiments in which just a single Weka algorithm will run in the grid. The workflow which is set up for running one of the Weka applications was shown in Figure 2. The workflow, as described in section 2, consists of the six units LoadDescription, GridURI, ParameterControl, Execution, Provenance and GridExplorer.

For the more complex experiment there is an obviously more complex workflow (Figure 5). The units used are more the less the same but occur multiple times and are renamed in this full workflow for a clearer view. In principle it consists of 3 times the workflow from the simple experiments in parallel but with the same input data and an additional Transfer unit which is responsible for copying the result files to a single directory.

5.2 The experimental setup

In the following we describe the settings during the execution of the workflow for the new included Weka algorithms.

The application to execute and the input file are, dependent on the experiment, preset by the values specified in the **LoadDescription** and the **GridURI** units.

When starting the workflow, it directly passes on to the **ParameterControl** unit, where the application's parameters can be specified. Most of the setting, e.g. the default values for the application's options and the system requirements are preset. These default values are taken from the application description file where the default values from the Weka algorithms were included (e.g. it is specified that each job shall perform 10 fold cross validation). What is left to be done is the following: We want to perform jobs which run the same algorithm with different parameter setting, so we have to select the options on which we will perform the sweep. At the Options panel of the **ParameterControl** unit we can then set the details for the sweep by choosing either a list or a loop for the parameter. For two parameters of the M5P algorithm this is shown in Figure 3 (loop) and Figure 4 (list). Out of these settings the system generates a multi-job description. The detailed settings which are used for the experiments in this paper are described in the following section about the evaluation.

Additionally we have to specify the applications data input. This is done by just selecting the URI which was passed from the **GridURI** unit in the input data drop down box at the Data mappings tab. At the same tab we have to set the naming of the output files which will be generated by the Weka algorithm. We can set some string and additional reference to the variable used for the parameter sweep so that each output file has a unique name and is not overwritten.

For the regression experiments all algorithms will be executed on the same dataset. We used the dataset House(16L) from a database which was designed on the basis of data provided by US Census Bureau and is concerned with predicting the median price of the house in the region based on demographic composition and a state of housing market in the region. The dataset, which was taken from the UCI Machine Learning Repository and the UCI KDD archive [8,12] and made available in the grid environment, contains 22784 cases and 17 continuous attributes. This size of data is justified because we are mainly interested in measurements of the overhead caused by grid computing in the DataMiningGrid environment which becomes clearer when using smaller datasets.

For the evaluation and the resulting comparison of runtimes on a different number of machines and with a different number of jobs we can set execution mode (Fork/Condor) and the maximum number of machines at the requirements tab. It is not necessary to restrict these requirements when running jobs.

The next step of the workflow is the execution unit, which submits the (multi) jobs to the resource broker. The jobs will be executed, and after all jobs are finished the **Provenance** unit and the **GridExplorer** show the provenance information about the execution and the result directory.

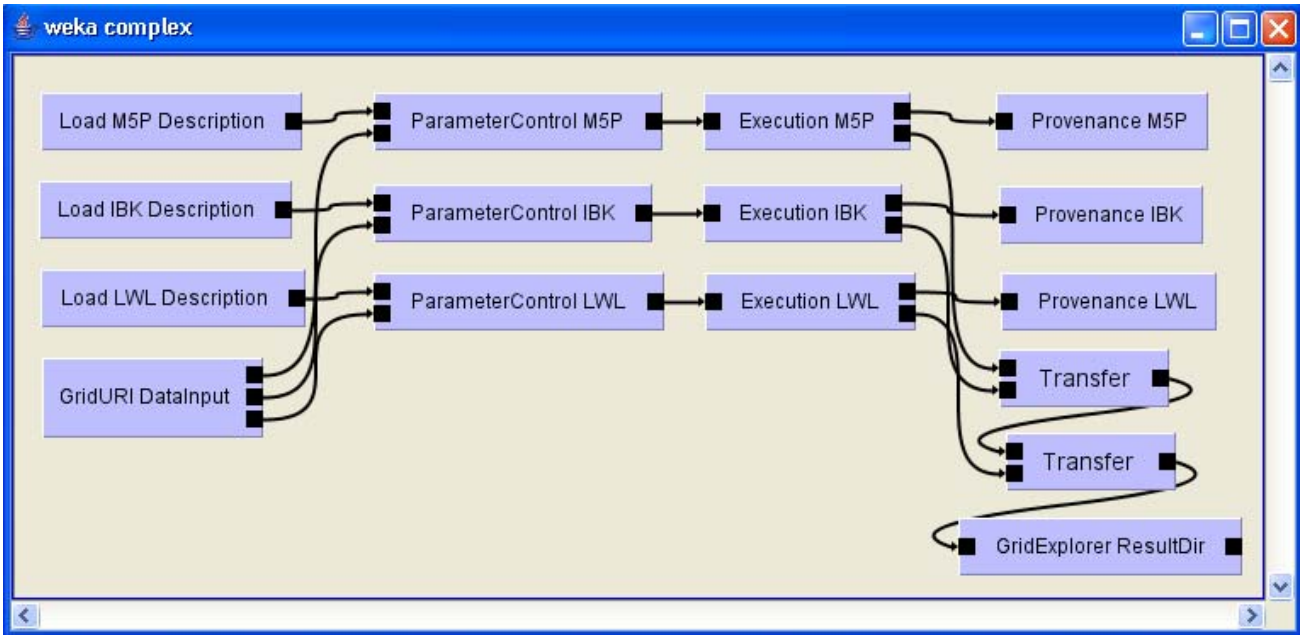

Figure 5: Complex Weka Workflow

5.3 Evaluation

In the following we will perform different kind of experiments. The grid-test bed on which we will run the jobs contains 2 GRAMs (Intel Pentium 4 2.40GHz, 2GB memory) and 6 Condor processors (AMD Opteron 244 1.80GHz, 4GB memory). An evaluation on a large test bed is not that easy, because in order to have comparable results the pool has to be free for the experiments and the jobs do not get blocked by other users and wait in the queue for a long time. Therefore the number of machines on which we run the experiments is not that large, but we expect the results to look the same on larger pools. For the evaluation we will vary the number of machines and/or the number of jobs and we will look at and compare the runtime.

Experiment M5P

During the M5P experiment we submit jobs to the grid which execute the Weka M5P algorithm with different parameter settings. The execution mode is Condor, which means that all jobs are submitted to the Condor pools which are connected to the GRAMs. In this experiment we will have a fixed number of jobs and we will vary the number of machines in the grid. We generate 10 jobs in total by using a list for the option BuildRegressionTree (true/false) and a loop for the option MinNumInstances (from 2 to 10 step 2). These jobs will be submitted to 1 to 6 machines.

Figure 6 visualizes the results of the M5P experiments. The graph shows the relation of the number of machines in the grid and the runtime of all 10 jobs. As expected, in general the runtime decreases the more machines the grid contains till the number of machines in the grid reaches the number of jobs. The jobs are distributed equally to the Condor machines.

Table 1 shows the maximum number of jobs which one of the machines has to compute (e.g. 10 jobs on 3 machines, so 2 machines take 3 jobs and one takes 4). This explains why there is no decrease in total runtime from 5 to 6 machines.

Table 1: Job distribution M5P

NumMachines	1	2	3	4	5	6
MaxJobNum per Machine	10	5	4	3	2	2

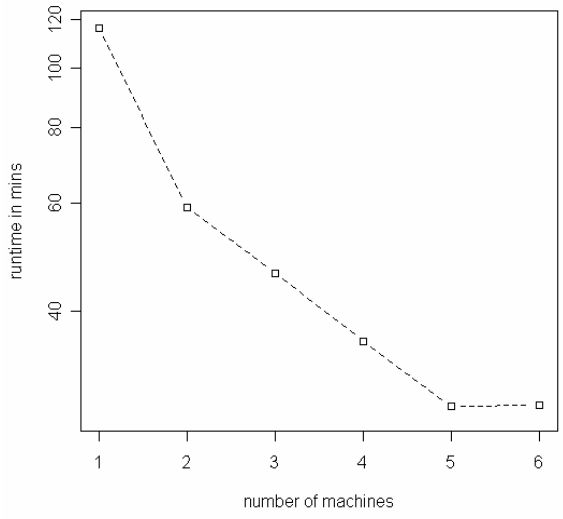

Figure 6: M5P results. Results in logscale.

Experiment IBK

In the IBK experiment we will submit jobs to the grid which execute the Weka IBK algorithm. We make different experiment series, each on a different kind of machine/pool, which are compared afterwards. The jobs are generated by varying the parameter KNN (from 1 to maximum 16) so that we have up to 16 jobs in total. These jobs will run a) in fork mode on the Globus machine, b) on a single machine inside the Condor pool and c) on the whole grid (which consists for our tests of 6 Condor machines). In each experiment series we have a fixed number of machines and we will vary the number of jobs.

The result (Figure 7) is as expected. At a) and b) the jobs are all executed on a single machine, the fork execution on the Globus machine has worse performance than the Condor machine. The runtime increases linear, but the Condor execution seems to be faster. This looks confusing, because the submission from the Globus machine to Condor should take some time so that the Condor execution should definitely take longer. The reason for this result is that the Globus machine has older hardware. When executing the jobs c) on 6 Condor machines, the runtime also increases linear, but in comparison to the Condor execution on a single machine the runtime decreases by a factor about 6.

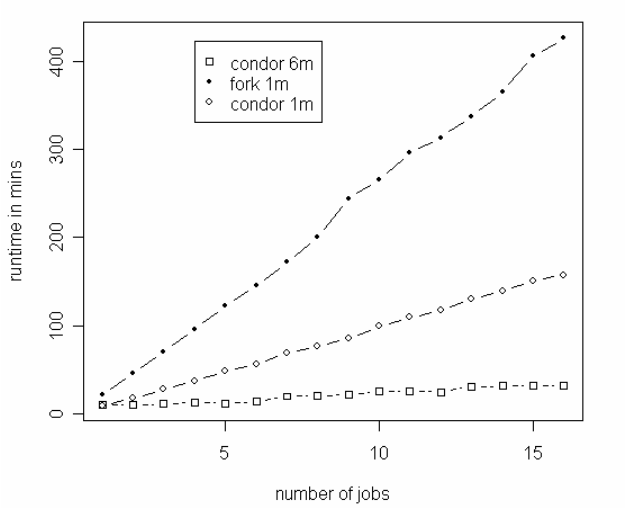

Figure 7: IBK results

Complex Experiment

This experiment is a more complex experiment with a more complex workflow than the previous ones. As can be seen in Figure 5 there are 3 branches inside the workflow which means that we have 3 application executions in parallel. Each application execution can, as in the previous experiments, result in a number of different jobs.

Each branch of the workflow belongs to one of the grid enabled Weka algorithms. For the M5P and the IBK algorithms we will vary the same parameters as in the previous experiments. For LWL we will vary the value for kNN and the weighting kernel, where we can use a Linear, Epnechnikov, Tricube and Constant kernel. Inverse and Gaussian kernels increase the runtime in comparison to the other jobs in a way (by a factor > 25) that the results are not usable.

For each of the algorithms the same number of jobs is submitted. Starting from 1 job per algorithm up to 6 we will have from 3 to 18 jobs.

As it is not possible to specify the jobs to run on the same Condor machine when having 3 different executions in the workflow (each of the executions would run on a single, but different machine) and we do not want to remove all but one machine from the pool we estimate the runtime on a single machine by summing up and multiplying the runtimes generated by the running of a single job per algorithm.

The results of the experiment are shown in Figure 8. The results are as expected and comparable with the ones from the previous experiment, although the steps in the runtime graph of 6 Condor machines are not as different. From the experiment we can see that the runtime does not depend on the complexity of the workflow but just on the number of jobs which are submitted and their runtime.

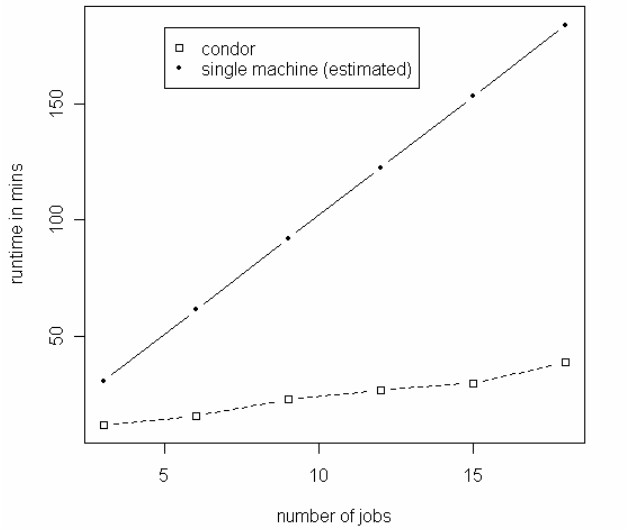

Figure 8: complex experiment results

The evaluation of the different – simple and complex - analysis scenarios emphasizes the easy set up and submission of data mining tasks. As the runtime analysis shows, the flexibility of the system does not result in a big performance overhead. The runtime of the experiments depend on the speed and the number of available machines. Grid enabled applications in the DataMiningGrid system can reach a very good scalability, especially when making use of organization overlapping computing resources.

6. CONCLUSION

In the last years, a number of Grid-enabled Data Mining platforms have been described. We see some technological convergence on standards based technology, e.g. WSRF and Globus. From a superficial architectural level one might be tempted to think that all those platforms are quite similar.

What is needed in the next stage of discussion is a systematic comparison of the capabilities, the strengths and the weaknesses of these platforms from a Data Miner's point of view. In this paper we made a first attempt to evaluate a system from the user's point of view, putting the extensibility requirement at center stage

and investigating in some detail how it is met by the DataMiningGrid platform.

To this end, we introduced the DataMiningGrid system as environment for distributed Data Mining in the Grid. In a detailed case study, we integrated new Weka algorithms into the system. We have shown that the DataMiningGrid platform stands out among other platforms by meeting the extensibility requirement without sacrificing scalability and increasing the grid computing related overhead.

We gave an evaluation of simple and more complex experiments which showed that the requirements were met. The system is capable of handling even more complex scenarios, e.g. where algorithms or the data should not be moved etc. The system is user friendly in a way that a data miner is able to use the system - from the inclusion of new applications to their execution in the context of complex experiments - without any specific knowledge of the system details.

For the future, we believe more such case studies and specific comparisons of functionality from a data miner's point of view are needed.

7. ACKNOWLEDGMENTS

This work was supported by the European Commission FP6 grant No. 004475. The DataMiningGrid Consortium (www.datamininggrid.org) consists of five organizations: University of Ulster, Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS, DaimlerChrysler AG, Technion - Israel Institute of Technology, and University of Ljubljana. We hereby acknowledge the cooperation of all DataMiningGrid partners and collaborators in the DataMiningGrid project.

The authors also gratefully acknowledge the financial support of the European Commission for the Project ACGT, FP6/2004/IST-026996.

8. REFERENCES

[1] Adeva, J. and Calvo, R., Text mining with Pimiento, IEEE Internet Computing 10 (4), 2006, pp. 27-35.

[2] Ali, A.S., Rana, O. and Taylor, I., Web services composition for distributed data mining, in: Proc of the 2005 IEEE International Conference on Parallel Processing Workshops (ICPPW'05), 2005.

[3] Brezany, P., Janciak, I. and AM, A.T., GridMiner: A fundamental infrastructure for building intelligent grid systems, in: The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05), 2005.

[4] Cannataro, M., Talia, D. and Trufino, P., Distributed data mining on the grid, Future Generation Computer Systems 18 (8), 2002, pp. 1101-1112.

[5] Cheung, W., Zhang, X.-F., . Luo, Z.-W. and Tong, F., Service-oriented distributed data mining, IEEE Internet Computing 10 (4) 2006, pp. 44-54.

[6] Foster, I., Kesselman, C., and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, 2001, pp. 200-222.

[7] Guedes, D., Meira, W.J. and Ferreira, R., Anteater: A service-oriented architecture for high-performance data mining, IEEE Internet Computing 10 (4), 2006, pp. 36-43.

[8] Hettich, S. and Bay, S. D., The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[9] Kravtsov, V., Niessen, T., Stankovski, V., and Schuster, A., Service-based Resource Brokering for Grid-based Data Mining, In Proceedings of The 2006 International Conference on Grid Computing and Applications, Las-Vegas, USA, 2006.

[10] Litzkow, M., Livny, M., Experience with the condor distributed batch system, in: Proc IEEE Workshop on Experimental Distributed Systems, 1990.

[11] May, M., Potamias, G. and Rüping, S., Grid-based Knowledge Discovery in Clinico-Genomic Data, In: Proceedings of the 7th International Symposium on Biological and Medical Data Analysis (ISBMDA 2006), pp. 219-230, Springer, 2006.

[12] Newman, D.J. & Hettich, S. & Blake, C.L. and Merz, C.J. UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[13] Saira, S. A. , Emmanouil, F. S., Ghanem, M., Giannadakis, N., Guo, Y., Kalaitzopolous, D., Osmond, M., Rowe, A., Syed, iJ. and Wendel, P., The design of discovery net: Towards open grid services for knowledge discovery, International Journal of High Performance Computing Applications 17.

[14] Stankovski, V., Swain, M., Kravtsov, V., Niessen, T., Wegener, D., Kindermann, J. and Dubitzky, W., Grid-enabling data mining applications with DataMiningGrid: An architectural perspective, Future Generation Computer Systems, 24 (4), 2008, p.p 259-279.

[15] Sotomayor, B. and Childers, L., Globus Toolkit 4: Programming Java Services, Morgan Kaufmann, 2006.

[16] Talia, D., Trunfio, P. and Verta, O., Weka4WS: A WSRF-Enabled Weka Toolkit for Distributed Data Mining on Grids. In Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005), Vol. 3721:309-320 of LNAI, Springer, Porto, Portugal, 2005.

[17] Taylor, I., Shields, M., Wang, I. and Harrison, A., The Triana Workflow Environment: Architecture and Applications, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (eds.), Workflows for e-Science, Springer, New York, NJ, USA, 2007, pp. 320-339.

[18] Witten, I. and Frank, E., Practical machine learning tools and techniques. 2nd Edition, Morgan Kaufmann, 2005.

[19] Wrobel, S., Wettschereck, D., Emde, W.. Extensibility in Data Mining Systems. In Simoudis, E, Han, J., and Fayyad, U., 2nd International Conference on Knowledge Discovery and Data Mining, KDD-96, 1996, pp. 214-219.