DMCloud: a Cloud-enabling DataMining Framework An architectural perspective

Khaled Tannir
contact@khaledtannir.net

ABSTRACT

DataMiningCloud is a framework that extends the current *Weka Grid-enabler Toolkits* for supporting parallel data mining algorithms and distributed data mining on a Cloud environment. Service Oriented Architecture (SOA) approach is used to implement all the data mining applications independent services, each capable of carrying out a set of predefined tasks.

DMCloud architecture relies on the Cloud IaaS *OpenNebula* Infrastructure extended with a specific workflows-based SLA broker, and other open technology and standards. It provides tools and services facilitating the user development, deployment and transparent utilization of data mining applications in a cloud environment.

Keywords Distributed Data mining Framework; Cloud Service Level Agreement Broker, Cloud workflow-based Service-Oriented architecture

1. INTRODUCTION

In this paper I present the reference architecture of *DMCloud*, a specific distributed data mining framework supporting parallel data mining algorithms and distributed data mining on a Cloud infrastructure. The challenge is to develop an environment suitable for executing data analysis and knowledge discovery tasks in a wide range of different applications within different sectors in science and technology (eg, automotive, biological and medical, etc.). I enhance the existing Distributed Weka Data mining tools in order to take advantage of features inherent to cloud characteristics that include elasticity, flexibility, extensibility, scalability, efficiency, conceptual simplicity and ease of use.

The existing distributed grid-based data mining applications toolkits facilitate the sharing of data, data mining application programs, processing units and storage devices in order to improve existing, and enable novel, data mining applications. In general, they are built on top of open Grid-based infrastructures with all the generic functionalities of these middlewares, but with additional features that ease the development and execution of complex data mining tasks.

DMCloud architecture relies on the Cloud IaaS OpenNebula Infrastructure [17] extended with a specific workflows-based SLA broker, and other open technology and standards. The current clouds offer interfaces too close to the infrastructure resources management, while users demand functionalities that automate the management of their application and/or business-based services as a whole unit. To overcome this limitation, I propose a new abstraction layer closer to the user point of view of data mining applications execution that allows for their automatic deployment escalation and efficient execution depending on the status of all services and resources of a Cloud.

DMCloud architecture is a business-ready service-oriented infrastructure empowering the service economy and data mining applications development, deployment and utilization in a flexible way. Service Oriented Architecture (SOA) approach is used to implement all these data mining applications independent services, each capable of carrying out a set of predefined tasks.

The remainder of this paper is organized as follows. Section II presents the main current Grid-based data mining frameworks in pointing out their limitation. Section III describes the developed reference architecture and Section IV describes the main characteristics of the data mining applications running in this environment. Section V concludes with a brief summary and outlook on the future works.

2. COMPARAISON OF CURRENT DISTRIBUTED WEKA SOLUTIONS

The "*Waikato Environment for Knowledge Analysis*"[1] (Weka) is an open source tool providing a collection of machine learning algorithms for data mining tasks developed in Java. Weka contains tools for data preprocessing, classification, regression, clustering, association rules and visualization. It also includes a variety of tools for transforming datasets.

Nowadays, Weka is recognized as a landmark system in data mining and machine learning [1]. The Weka Toolkit has been rewritten entirely from scratch since the first release in 1993 [2]. Weka is a standalone application which it can run on a single machine only. To go beyond this limitation there are a number of projects that have extended Weka for distributed data mining. Some of those projects are: Weka-Parallel which provides a distributed cross-validation facility [3]; GridWeka provides distributed scoring and testing as well as cross-validation [4]; FAEHIM and Weka4WS [5] make Weka available as a web service and WekaG [6] which extends the data mining Weka Toolkit to perform data mining tasks on a grid environment.

Weka has three main graphical user interfaces that enable easy access to the underlying functionality. Those graphical user interfaces are "*Explorer*", "*Classify*" and "*Experimenter*". Weka supports several input file format such *CSV*, *LibSVM's* format, *C4.5's* format and ARRF (*Attribute-Relation File Format*[2]) its own input file format. In Weka, different data mining algorithms are implemented in components and users can specify one or more options of algorithms through command line or GUI.

[1] - http://www.cs.waikato.ac.nz/ml/weka/

[2] - http://www.cs.waikato.ac.nz/~ml/weka/arff.html

Weka provides implementations of learning algorithms that it can easily be applied to a dataset. It also includes a variety of tools for transforming datasets, such the algorithms for discretization process which is known to be one of the most important data preprocessing tasks in data mining. We can preprocess a dataset, feed it into a learning scheme, and analyze the resulting classifier and its performance—all without writing any program code at all.

The Weka's API is organized in a hierarchical structure, and the algorithms are delimitated by their relevancy to the classes of data mining tasks as presented in Figure 1:

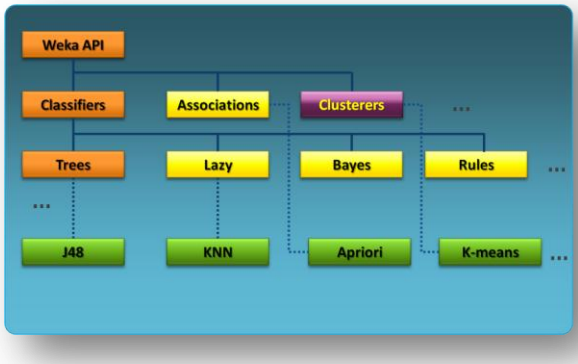

Figure 1-Taxonomy of Weka API algorithms

After this short presentation of the original stand-alone version of the Weka Toolkit, we describe in the rest of this section their different parallelization initiatives.

(1) Weka Parallel:

Weka-parallel aims to parallelize Weka classifiers. It expands upon the original program by performing n-fold cross-validations in parallel to demonstrate a significant speed increase over Weka. Weka-Parallel was designed for the sole purpose of drastically decreasing the amount of time necessary to run cross-validation on a dataset using any given classifier.

Weka-parallel consists of two components; The *Server* side and the *Client* side. Firstly each contributing machine start a daemon that would listen on a specific port, and then a client begins to do cross-validation within Weka in parallel by reading all servers information from a configuration file.

The server translates client requests into calls to the corresponding Weka functions. At the client side, a Weka-parallel session is run on a single machine and a number of distributed servers. The client establishes each connection thread for each server and a thread for using client side computation.

Weka-Parallel handles all the details of cross-validation, result aggregation, and multiprocessor communication. The client need only ready the remote machines for incoming work requests.

Weka-parallel supports *Load balancing* and *Fault monitoring*. The load balancing idea is that when program decide which server to give the next fold to, it take into account their performance based on servers information. Fault monitoring is used to identify faults as quickly as possible after they occur and to identify the cause of the fault so that remedial action may be taken.

The computer running Weka-Parallel (called 'the client') can connect to the available servers to start distributing work. Each server will receive an index of fold to do exactly computation. Weka-parallel uses a *round-robin* algorithm to allocate folds to servers.

In Weka-Parallel only one algorithm cross-validation has been parallelized. As future work some additional distributed Weka functions and features will be added to Weka-Parallel program . One is to use a server to build an evaluation model instead of using client side. Another one is to use some servers to apply an existing model on a new dataset instead of using client side. The last one is to apply an existing model to predict a new dataset instead of using client side. All of them will use fastest available servers.

Weka-parallel doesn't support OGSA-style service and therefore it cannot use Globus API toolkit.

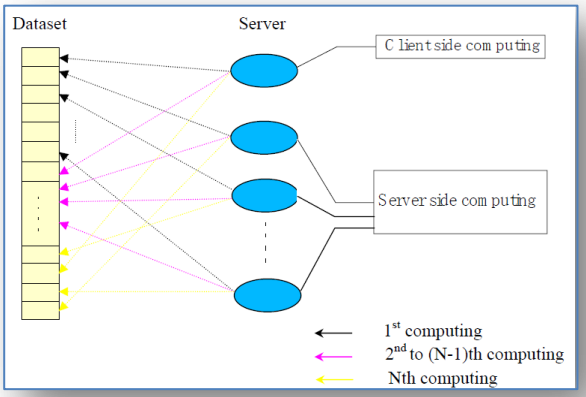

Figure 2 - Weka-Parallel distributed computing model

(2) Grid-enabled Weka:

Grid-enabled Weka is a toolkit for machine learning on the Grid. Weka focuses on classifiers algorithms, and the goal of classifier learning process is to derive a classifier from a set of labeled data. This task can be time consuming and require significant amounts of memory, especially for large data sets.

Labeling in Grid-enabled Weka involves applying a previously learned classifier to an unlabelled data set to predict instance labels. Testing takes a labeled data set, temporarily removes class labels, applies the classifier, and then analyses the quality of the classification algorithm by comparing the actual and the predicted labels. Finally, for n-fold cross-validation, a labeled data set is partitioned into n folds, and n training and testing iterations are performed. For each iteration, one fold is used as a test set, and the rest of the data is used as a training set. A classifier is learned on the training set and then validated on the test data.

This labeling function is distributed by partitioning the data set, by labeling several partitions in parallel on different available machines, and merging the results into a single labeled data set [9].

The quality of a classifier often depends on various algorithm parameters. The same classifier may need to be trained with different parameters to obtain better results. In Grid-enabled Weka, the user can run a training task on a remote machine. This allows the same user to train several classifiers in parallel by launching multiple Weka tasks from the user's computer.

Grid-enabled Weka is based on Weka-Parallel and consists of two main components. The Weka server, and the Weka client:

a. The Weka server is based on the original Weka. Each machine participating in a Weka Grid runs the server component. The server translates client requests into calls to the corresponding Weka functions. It also provides additional functions like data set recovery from local storage after a crash. The same server can be used by several different clients, which allows users to share resources of the same machine.

b. The Weka client is responsible for accepting a learning task and input data from a user and distributing the work on the Grid. The client implements the necessary functionality for load balancing and fault monitoring/recovery. It also allows the user to specify resource constraints for a given task and takes these into account when allocating the jobs to servers.

The Grid-enabled Weka system uses a custom interface for communication between clients and servers utilizing native Java object serialization for data exchange.

The figure 3 illustrates a Grid-enabled Weka usage scenario.

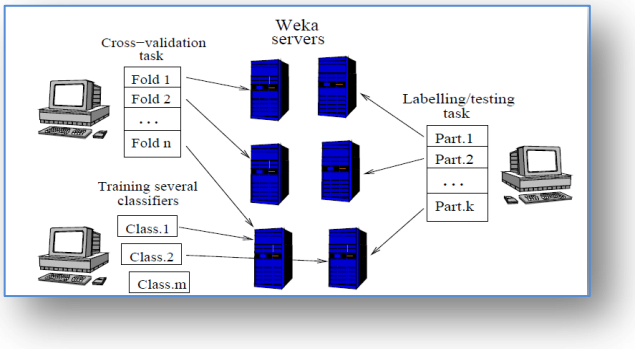

Figure 3 - Grid-enabled Weka, usage scenario

Grid-enabled Weka dosen't support OGSA style service and therefore it cannot use Globus API toolkit. An advantage of the current Grid-enabled Weka implementation is that in a trusted and centrally controlled environment it allows for utilizing idling computing resources with minimal set up, configuration, and maintenance efforts. This is especially convenient for machine learning practitioners who may not be proficient in parallel computing or Grid technologies [9].

(3) GridWeka 2:

GridWeka2 [10] is able to run cross-validations in the *Weka Explorer* and *Experimenter* in parallel and distributed over several machines while being easily configurable. GridWeka2 consists in two modules; the Server module and the Client module.

Before starting GridWeka2 it is necessary to tell it where the servers of the grid are located. Each server of GridWeka2 has to be set up manually. It is not the case that when one server is set up on the grid that the server looks for available nodes on the grid. The server only runs on one node and if more than one node wants to be used the other ones has to be set up manually as well. GridWeka2 is an application that can use multiple computing sources in parallel.

The GridWeka2 tool doesn't modify the Weka GUI which is the same as the original and it does not work in command line interface (CLI) but it works in both the Weka Explorer as well as the Weka Experimenter mode. Furthermore in these two modes it works in the same way as the original Weka for as long as the user is concerned. GridWeka2 uses the servers.csv file to connect to the available servers to perform the cross validation in parallel.

GridWeka2 includes the *Triskel* algorithm, which is an interesting ensemble learning method. Triskel learns an ensemble of classifiers that are biased to have high precision. Also, Triskel uses weighted voting like most ensemble methods, but the weights are assigned so that certain pairs of biased classifiers outweigh the rest of the ensemble, if their predictions agree [10].

(4) WekaG:

WekaG is an application that performs data mining tasks on a grid. It is able to support parallel implementations of data mining algorithms. WekaG extends the data mining toolkit Weka and is based on client/server architecture. WekaG is a particular implementation of a more general architecture named *Data Mining Grid Architecture* (DMGA) [6].

WekaG is a vertical architecture implementation of the DMGA, which is based on the three data mining phases and is composed of generic, data grid and specific data mining grid services. WekaG offers two separate modules: a *server* module and a *client* module.

The WekaG server side module is responsible for the creation of instances of data mining grid services that implement the functionality of every algorithm and stage of the data mining process and data mining phases. The WekaG client side is integrated in the Weka application interface (without modifying Weka itself) and it interacts with the server [11].

WekaG was implemented to include at least the following features: a) coupling data sources, b) authorization access to resources, c) discovery based on metadata, d) planning and scheduling tasks e) identifying the available and appropriate resources.

WekaG includes the *AprioriG* service, which implements the capabilities of the *Apriori* algorithm in a grid environment. The main function of the *AprioriG* service (named Build Association) aims to produce association rules from a data set. With the first prototype of WekaG, only the Apriori algorithm was implemented to be used on the grid.

WekaG uses the Globus Toolkit 4 (GT4) as the grid middleware. GT4 supports the grid standard *Web Services Resource Framework* (WSRF) [12]. WSRF has been performed by the Globus Alliance and IBM, with the goal of integrating previous work on the so-called *Open Grid Services Architecture* (*OGSA*) [15] with new Web Services mechanisms and standards.

WekaG makes use of GridFTP, if needed, to transfer data to the server nodes in the *pre-processing* stage. GridFTP is a common data transfer and access protocol in Grid environments. This protocol, which extends the standard FTP protocol, provides a superset of the features offered by the various Grid storage systems currently in use [13]. Finally, WekaG implementation (which is based on Weka) is restricted to applications implemented in WekaG.

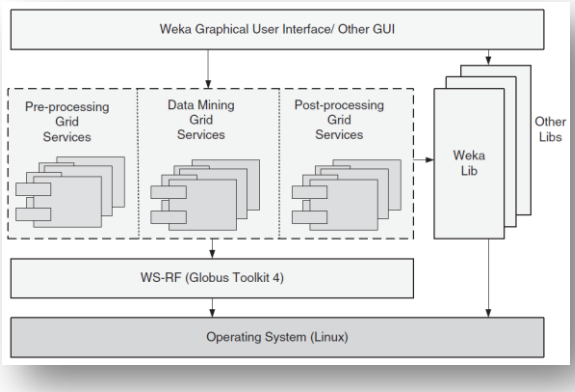

Figure 4 - DMGA Overview

(5) Weka4WS:

Weka4WS is a framework that extends the Weka toolkit for supporting distributed data mining on Grid environments. Weka4WS adopts the *Web Services Resource Framework* (WSRF) for accessing remote data mining algorithms and managing distributed computations. The Weka4WS user interface is a modified *Weka Explorer* environment that supports the execution of both local and remote data mining tasks. On every computing node, a WSRF-compliant Web Service is used to expose all the data mining algorithms provided by the Weka library.

The goal of Weka4WS is to extend Weka to support remote execution of the data mining algorithms. In such a way, distributed data mining tasks can be executed on decentralized Grid nodes by exploiting data distribution and improving application performance.

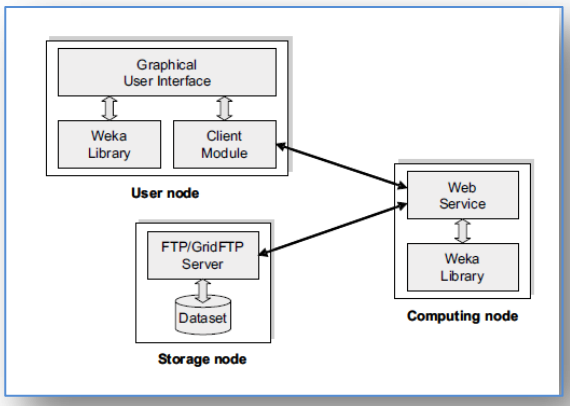

Figure 5 - The general architecture of Weka4WS

In Weka4WS, the data-preprocessing and visualization phases are executed locally, whereas data mining algorithms for classification, clustering and association rules can be also executed on remote Grid resources.

Weka4WS extends the Weka GUI to enable the invocation of the data mining algorithms that are exposed as Web Services on remote machines. It has been designed and developed by using *Web Services Resource Framework* (WSRF).

Figure 5 shows the general architecture of the Weka4WS framework that includes three kinds of nodes: *storage nodes*, which store the datasets to be mined; *computing nodes*, on which the remote data mining tasks are executed; *user nodes*, which are the local machines of the users.

The Weka4WS modify the Weka GUI implementation by adding a "Remote" pane to the original Weka Explorer. This pane provides a list of the remote Web Services that can be invoked, and two buttons to start and stop the data mining task on the selected Web Service.

At any time user can start multiple data mining tasks in parallel on different Web Services, this way taking full advantage of the distributed Grid environment for implementing parallel and distributed data mining tasks. Whenever the output of a data mining task has been received from a remote computing node, it is visualized in the standard "output" pane.

Computing nodes include two components: a *Web Service* (WS) and the *Weka Library* (WL). The WS is a WSRF-compliant Web Service that exposes the data mining algorithms provided by the underlying WL. Therefore, requests to the WS are executed by invoking the corresponding WL algorithms. Finally, storage nodes provide access to data to be mined. To this end, an FTP server or a GridFTP server [13] is executed on each storage node. The dataset to be mined can be locally available on a computing node, or downloaded to a computing node in response to an explicit request of the corresponding WS.

I consider the following criteria for comparing all the current Grid Weka-enabled data mining frameworks:

1) **Service-oriented architecture support**

The system is based on principles of service-oriented architecture. This architecture style implements many independent services, each capable of carrying out a set of predefined tasks.

2) User interfaces types

The system may provide a workflow-based client technology (i.e. Triana [17] or a single Web client. The workflow client facilitates the composition and execution of complex data mining processes by users with comprehensive data mining expertise. The Web client is designed for data mining domain oriented end users without knowledge of the underlying data mining and grid technology.

3) Extensibility

The system is built following a component-based black box plug-in approach. All system components can be extended and exchanged easily. Custom plug-ins can be implemented without consideration of programming aspects like parallelism or internal details.

4) Data mining-aware Cloud Resources transparency

The system provides a data mining-aware resource brokering that takes into account requirements and constraints relevant to data mining.

5) Applications extensibility and platform independence

The application developers should be able to implement cloud-enable existing data mining applications with little or no modification to the original data mining application program from one platform to another.

6) Families of Data mining algorithms available

7) Parallelized Data mining algorithms available

8) Repository of applications metadata

The system allows the characterization of data mining application programs or components with computer-readable metadata. The search in this repository can be performed based on different criteria.

9) Ease of deployment

The system is able to mine data without the need to install grid middleware or any data mining software at the location where their data resides.

10) Grid infrastructure Standards supported

WSRF standard has been widely adopted by the grid community. Globus Toolkit 4 is one the main middleware used. Globus Toolkit 4 implements the WSRF and follows the OGSA standard.

11) Data Mining Standards supported

Main data mining standards supported by the system (e.g.: CRISP-DM, PMML).

12) Distributed data management

The system facilitates management of distributed data sources, i.e., access to distributed data sources and shipping/staging of data across grid nodes.

Feature/system	Weka	WekaG	Grid-enabled Weka	Weka Parallel	GridWeka 2	Weka4WS
Architecture	Single	DMGA	C/S	C/S	C/S	C/S
Open Source	Yes	No	No	No	Yes	Yes
Grid enabled	N/A	Yes	Yes	Yes	Yes	Yes
Middleware	N/A	GT3	No	No	No	No
Workflow	Yes	N/I	N/I	N/I	N/I	Yes
Parallelized tasks	N/A	Association	Classifier	Cross-validation	Cross-validation	Classification Clustering Association
SOA Support	N/A	No	No	No	No	Yes
Extensibility	Yes	No	No	No	No	Yes
User Grid Ressource Broker	N/A	N/I	No	No	No	N/I
Platform Independence	Yes	Yes	Yes	Yes	Yes	Yes
DM Algorithms	Many	Apriori	N/I	None	None	Many
Parallelized DM algorithm	N/A	AprioriG	None	None	None	Many
Metadata repository	Yes	Yes	N/I	N/I	N/I	No
Ease of deployment	Yes	Yes	No	No	No	Yes
Grid infrastructure standard	N/A	WSRF	No	No	No	WSRF
DM Standard support	PMML	none	none	none	none	No
Distributed data management	N/A	Yes	Yes	Yes	Yes	Yes

C/S = Client / Server, N/A = Not Applicable, GT3 = Globus Toolkit 3,
N/I = Not implemented / Information could not be obtained

3. *DMCLOUD* DATA MINING FRAMEWORK ARCHITECTURE

DMCloud Distributed Data Mining architecture must satisfy at least the following requirements:

(1) Grid resources transparency: users should be able to carry out the data mining tasks without needing to understand detailed aspects of the underlying Grid or Cloud technologies. Data mining operations must be executed on suitable machines in the Cloud without direct user interaction.

(2) Application development support: Developers of data mining solutions should be able to cloud-enable existing data mining applications, techniques and resources easily with little or no intervention in existing application code. Parallelization of existing Weka data mining algorithms will improve their efficiency in a large scale environment.

(3) Workflow and Service-orientated architecture and interoperability: The system should adhere to existing and emerging Clouds-related standards and be based on widely used open source technology.

These high-level requirements lead to a natural breakdown of architectural layers for a cloud-based data mining system; we distinguish user, data mining applications, cloud services and cloud resources management layers. The user requirements are dictated by the need of end users to define and execute data mining tasks, and by developers and administrators who need to evolve and maintain the system. Application program requirements are driven by technical factors such as optimal Cloud resources allocation strategies, software and hardware architectures, system interfaces, standards, and so on.

In this section, I present a general architecture of the *DMCloud*, *Data Mining Cloud framework*. I present main building blocks and architecture layers (components and interactions) of the system. The data mining layer is built on the top of a multi-level *workflow-based service-oriented Cloud Broker* as depicted in figure 6.

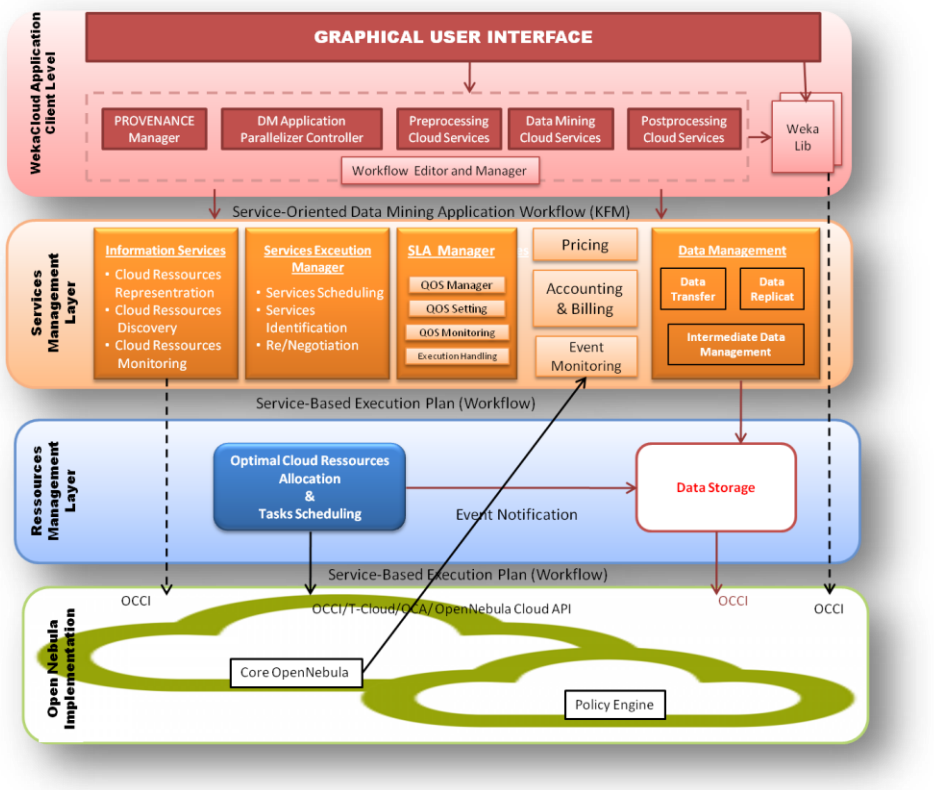

Figure 6 – DM*Cloud* Architecture: a multi-level workflow-based architecture

In this project, I try to create an environment in which data mining applications and Cloud Services can easily be deployed executed and managed. This environment provides various abstraction layers from data mining applications, services to specific resources and platforms. The general DM*Cloud* Data Mining architecture includes four basic layers from top to bottom:

- Data Mining Applications layer (user applications),
- Cloud SLA-based service layer: this workflow driven layer comprises cloud services to facilitate the development/deployment of user data mining applications. The cloud management services (e.g. brokering, pricing, accounting, and virtual machine management) and other functional components are achieved at this layer. This layer ensures service-level-scheduling tasks. The workflow schedulers play a main role in the lifecycle of a workflow instance. Based on workflow specifications (including task definitions, process structures and QoS constraints), the global scheduler assigns workflow tasks to candidate services and negotiate service level agreements. Some specific *metaheuristic scheduling algorithms* will allow generation of service-level scheduling plans in an efficient fashion.
- Cloud Resources Management. It is a unified resources management layer (abstracted/encapsulated resources by virtualization). This layer ensures a Cloud optimal resources allocation task-level scheduling operations.
- OpenNebula fabric layer (physical hardware resources). OpenNebula is an open source virtual infrastructure engine which provides features to control, deploy and monitor virtual machines. A modular and open architecture is used which support the integration of all kind of hypervisors. Three main components are present within *OpenNebula*. The Core is a centralized component, which manages the lifecycles of the virtual machines. Next to the core a Capacity Manager adjusts the placement of the virtual workloads. To support multiple hypervisors a third component is used for abstraction. The *Virtualizer Access Layer* exposes the features of the hypervisors to the Core.

Finally, the top layer depicts the client side components of DM*Cloud* architecture. I use a workflow editor like Triana [17] to compose data mining application workflows. Web-based clients for data mining applications may also be used. They can offer a greater degree of user-friendliness for those users who do not require detailed technological control. A user has to specify application information via the Web Application client, and this includes information on the following: (1) general information about the application; (2) general execution details; (3) input data requirements; (4) where to store output data; and (5) specific execution requirements. In this respect the *Triana solution* for example provides more modularity and greater flexibility.

Data mining services are essential in this architecture and are usually linked to data mining techniques and algorithms. We extend the *OpenNebula Cloud* services architecture by the definition of specific data mining services. The main functionalities of every data mining activities are deployed by means of cloud services including the following:

- *Specific Data Filtering Service*: this service is in charge to filter the huge amount of data involved in the process of data mining.
- *Specific Data Replication Service*: This service deals with data replication.

- *Specific Data Access Service*: this service provides data access mechanisms to data mining applications on the Cloud.
- *Specific Data Discovery Service*: This service improves the discovery phase in the Cloud for mining applications.

4. CLOUD-ENABLING DATA MINING APPLICATIONS FEATURES

I define a data mining application as an executable workflow-based composition of user-oriented services that performs one or more data mining tasks. Data mining applications include executable data mining algorithms, data mining tools, components, libraries etc… Enabling this type of resource to achieve user-based services in cloud computing environments facilitates the development of flexible, scalable and distributed data mining applications.

A data mining application follows the basic *preprocessing-modeling-postprocessing* scheme to reveal the information and knowledge behind the data according the PMML and CRISP-DM data mining standards. Client user interface comprises two different Data Mining application clients: A workflow editor and manager, and a Web client. Both clients are able to handle multi-component data mining applications. The workflow editor and manager client are designed to facilitate detailed user control of the individual data mining application components, including workflow composition, parameter settings, input and output setting, etc.

The resulting data mining application workflow describes application information that is specific to the data mining aspect of the application. The content structure of this workflow is mainly based on CRISP-DM. CRISP-DM stands for CRoss-Industry Standard Process for Data Mining. It consists on a cycle that comprises six stages [18]

The data is analyzed within the preprocessing step of the Data Mining process. Then, the following modeling step consists of applying data analysis and discovery algorithms to generate models that describe the data. It commonly involves four classes of task: *classification*, *clustering*, *regression* or *association rule learning*. The Data Mining process finishes with the interpretation and evaluation of the models. The evaluation of a Data Mining problem typically involves applying the new model to data that was not used to generate the model.

Depending on the amount of data, or the type of function to be modeled, different evaluation approaches may be used. Therefore, in the DM process, there are four main elements to define: the dataset (obtained by selecting, pre-processing, and transforming the initial dataset), the model or language representation, the learning algorithm, and how to evaluate the model. The number of combinations of those four elements is huge, since there are many different methods and techniques appropriate for each phase, all of them with different parameter settings, which can be applied in different orders. The complexity of this combinatorial process suggests using automated approaches

DMCloud offers two architectures implementations possibilities for data mining *applications* composition: *horizontal* and *vertical.*

The goal of the vertical composition is to enhance the performance of the data mining process by combining the use of several services that provide the same functionality. The data may be partitioned into homogeneous or heterogeneous segments. The vertical composition allows accessing a different data portion and each service instance accesses different views of the same data set depending on data division. Then, results from every resource are combined to compute the answer, which is then transferred to the client.

In horizontal composition we aim to involve the collaboration of different functional services. Typically, the output of one service is the input of the subsequent service. Horizontal composition can be use to combine data mining services and data transfer services, to move data to a specific location after that to run a specific algorithm on it. In some case, it is possible that an adapter may be required to adjust the output of one service to be used as the input of other.

The common usage scenario of the system is a combination of three main steps:

- Step 1: Workflow data mining application composition by the user.
- Step 2: Execution of the workflow in the Cloud.
- Step 3: Retrieval and annotation of the results.

Classically, the WEKA Toolkit incorporates the *WEKA Knowledge Flow* that allows building of a knowledge flow for processing and analyzing data. We use this component that bring most of WEKA functionalities as: loading data, preparing data for cross-validation evaluation, applying filters, applying learning algorithms, showing the results graphically or in a text window etc. Knowledge flows are stored in *KFML* files that can be given also as input to WEKA Toolkit.

A *KFML* file is an XML based file including two sections: In the section 1 is defined all the components involved in the knowledge flow, as data file loaders, filters, learning algorithms, or evaluators. Section 2 enumerates the links among the components. This section defines how the data flows in the data mining process, or how to connect the output of a component with the input of other components.

Weka Knowledge Flow allows loading, graphically editing, executing and saving KFML files. KFML files can be executed both by using the graphical interface or the WEKA API. A knowledge flow can be seen as the sequence of steps that must be performed to execute a data mining process.

We use *Predictive Model Markup Language* (PMML) [14] to support the exchange of data mining models between different applications and visualization tools. PMML is an XML-based language for describing data mining models. PMML is a set of *Document Type Descriptions* (DTDs) specified in XML.

5. CONCLUSION AND FUTURE WORK

In this paper, I have described a DMCloud reference architecture, a framework that extends the current *Weka Grid-enabler and parallel Toolkits* for supporting parallel data mining algorithms and distributed data mining on a Cloud environment. I discuss the main architectural components and interfaces and the main features of data mining applications running in this environment.

The technology developed facilitates cloud-enabling and parallelization of data mining applications. Major features of the DM*Cloud* technology include high performance, elasticity, scalability, flexibility, ease of use, conceptual simplicity, compliance with emerging Cloud and data mining standards (e.g., CRISP-DM) in a large scale, data-massive and high performance environment,. This architecture that seems an attractive alternative of existing distributed data mining Weka solutions in Grids will be evaluated in my future works on the basis of a wide range of representative modern data mining applications.

6. REFERENCES

[1] Piatetsky-Shapiro, Gregory. Winner of SIGKDD Data Mining and Knowledge Discovery Service Award . KDnuggets. [Online] June 28, 2005. [Cited: 12 29, 2010.] http://www.kdnuggets.com/news/2005/n13/2i.html.

[2] Frank, Eibe, et al. The WEKA Data Mining Software: An Update. SIGKDD Explorations. July 2009, Vol. 11, 1.

[3] Dietterich, T. G., Lathrop, R. H. and Lozano-P´erez, and T. Solving the multiple instance problem with axis-parallel rectangles. Artif. Intell. 1997, Vol. 89, (1-2), pp. 31-71.

[4] Altorf, Maarten. Data mining on grids. Universiteit Leiden. August 2007.

[5] Domenico Talia, Paolo Trunfio, Oreste Verta. Weka4WS: a WSRF-enabled Weka Toolkit for Distributed Data Mining on Grids. DEIS, University of Calabria. 2006.

[6] Perez, Maria S., et al. Adapting the Weka Data Mining Toolkit to a Grid Based Environment. Lecture Notes in Computer Science. Jan 2005, Vol. Volume 3528, pp. 492-497.

[7] Zuo, Xin. High Level Support for Distributed Computation in WEKA. University College Dublin. 2004.

[8] R.Musicant, Sebastian Celis and David. Weka-Parallel: Machine Learning in Parallel. s.l. : Carleton College Computer Science Technical Report 2002b, August 2002.

[9] Rinat Khoussainov, Xin Zuo, and Nicholas Kushmerick. Grid-enabled Weka: a toolkit for machine learning on the Grid. s.l. : ERCIM News, 2004.

[10] Hess, Andreas. GridWeka2. GridWeka2. [Online] Mars 2007. http://www.andreas-hess.info/projects/gridweka2/index.html.

[11] Perez, Marıa S., et al. Design and Implementation of a Data Mining Grid-aware Architecture. Design and Implementation of a Data Mining Grid-aware Architecture. 2008.

[12] Czajkowski, Karl, et al. The WS-Resource Framework. IBM Library. [Online] 1.0, 05 03, 2004. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

[13] W., Allcock. GridFTP: Protocol Extensions to FTP for the Grid. s.l. : Argonne National Laboratory, Avril 2003.

[14] PMML- Predictive Model Markup Language. Knol - A unit of knowledge. [Online] http://knol.google.com/k/pmml#

[15] Foster, I., Kesselman, C., Nick, J., Tuecke, S. Grid Computing: Making the Global Infrastructure a Reality. [ed.] Wiley. 2003. pp. 217-249.

[16] V. Stankovski, et al., Grid-enabling data mining applications with DataMiningGrid: An architectural perspective, Future Generation Computer Systems (2007),

[17] Ian Taylor, M. Shields, I. WANG and A. Harrison “The Triana Workflow Environment : Architecture and Applications” Workflows for e-science – Springer 2007 p. 320-339

[18] Chapman, P. et al, 2000. CRISP-DM 1.0 - Step-by-step data mining guide. Accessed from http://www.crisp-dm.org/CRISPWP-0800.pdf