Weka4WS: A WSRF-Enabled Weka Toolkit for Distributed Data Mining on Grids

Domenico Talia, Paolo Trunfio, and Oreste Verta

DEIS, University of Calabria,
Via P. Bucci 41c, 87036 Rende, Italy
{talia, trunfio}@deis.unical.it

Abstract. This paper presents Weka4WS, a framework that extends the Weka toolkit for supporting distributed data mining on Grid environments. Weka4WS adopts the emerging 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 paper describes the design and the implementation of Weka4WS using a first release of the WSRF library. To evaluate the efficiency of the proposed system, a performance analysis of Weka4WS for executing distributed data mining tasks in different network scenarios is presented.

1 Introduction

Complex business and scientific applications require access to distributed resources (e.g., computers, databases, networks, etc.). Grids have been designed to support applications that can benefit from high performance, distribution, collaboration, data sharing and complex interaction of autonomous and geographically dispersed resources. Since computational Grids emerged as effective infrastructures for distributed high-performance computing and data processing, a few Grid-based KDD systems has been proposed [1,2,3,4]. By exploting a service-oriented approach, data-intensive and knowledge discovery applications can be developed by exploiting the Grid technology to deliver high performance and manage data and knowledge distribution. As critical for scalable knowledge discovery, our focus here is on distributed data mining services beginning to allow distributed teams or virtual organizations accessing and mining data in a high-level, standard and reliable way.

This paper presents *Weka4WS*, a framework that extends the widely used Weka toolkit [5] for supporting distributed data mining on Grid environments. Weka provides a large collection of machine learning algorithms written in Java for data pre-processing, classification, clustering, association rules, and visualization, which can be invoked through a common *Graphical User Interface* (*GUI*). In Weka, the overall data mining process takes place on a single machine, since the algorithms can be executed only locally. 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.

In Weka4WS, the data-preprocessing and visualization phases are still executed locally, whereas data mining algorithms for classification, clustering and association rules can be also executed on remote Grid resources. To enable remote invocation, each data mining algorithm provided by the Weka library is exposed as a Web Service, which can be easily deployed on the available Grid nodes. Thus, Weka4WS also extends the Weka GUI to enable the invocation of the data mining algorithms that are exposed as Web Services on remote machines. To achieve integration and interoperability with standard Grid environments, Weka4WS has been designed and developed by using the emerging *Web Services Resource Framework* (*WSRF*) [6] as enabling technology.

WSRF is a family of technical specification concerned with the creation, addressing, inspection, and lifetime management of *stateful resources*. The framework codifies the relationship between Web Services and stateful resources in terms of the *implied resource pattern*. A stateful resource that participates in the implied resource pattern is termed *WS-Resource*. WSRF describes the WS-Resource definition and association with the description of a Web Service interface, and describes how to make the properties of a WS-Resource accessible through a Web Service interface.

Initial work on 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*) [7] with new Web Services mechanisms and standards. The Globus Alliance recently released the Globus Toolkit 4 (GT4) [8], which provides an open source implementation of the WSRF library and incorporates services implemented according to the WSRF specifications. The Weka4WS prototype described in this paper has been developed by using the Java WSRF library provided by a development release of Globus Toolkit 4 (Globus Toolkit 3.9.2 Core version).

The paper describes the design, implementation and performance evalution of Weka4WS. To evaluate the efficiency of the proposed system, a performance analysis of Weka4WS executing distributed data mining tasks in different network scenarios is presented. The remainder of the paper is organized as follows. Section 2 describes the architecture and the implementation of the Weka4WS framework. Section 3 presents a performance analysis of the Weka4WS prototype. Section 4 discusses related work. Finally, Section 5 concludes the paper.

2 The Weka4WS Framework

Figure 1 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.

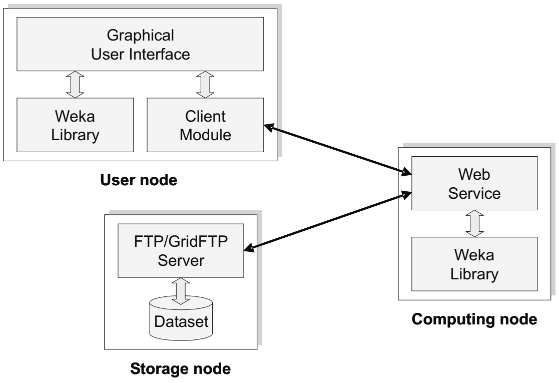
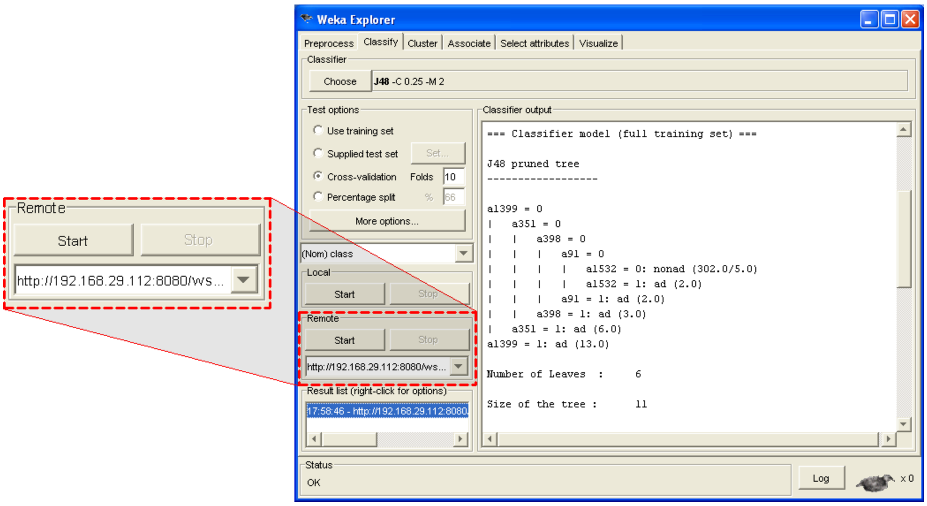


Fig. 1. The general architecture of the Weka4WS framework

User nodes include three components: *Graphical User Interface* (*GUI*), *Client Module* (*CM*), and *Weka Library* (*WL*). The GUI is an extended Weka Explorer environment that supports the execution of both local and remote data mining tasks. Local tasks are executed by directly invoking the local WL, whereas remote tasks are executed through the CM, which operates as an intermediary between the GUI and Web Services on remote computing nodes.

Figure 2 shows a snapshot of the current GUI implementation. As highlighted in the figure, a "Remote" pane has been added to the original Weka Explorer environment. This pane provides a list of the remote Web Services that can



J48 -C 0.25 -M 2

Cross-validation Folds 10

Percentage split % 66

(Nom) class

http://192.168.29.112:8080/ws...

17:58:46 - http://192.168.29.112:8080/

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------

a1399 = 0
|   a351 = 0
|   |   a398 = 0
|   |   |   a91 = 0
|   |   |   |   a1532 = 0: nonad (302.0/5.0)
|   |   |   |   a1532 = 1: ad (2.0)
|   |   |   a91 = 1: ad (2.0)
|   |   a398 = 1: ad (3.0)
|   a351 = 1: ad (6.0)
a1399 = 1: ad (13.0)

Number of Leaves  :     6

Size of the tree :      11
```

Status OK

Fig. 2. The Graphical User Interface: a "Remote" pane has been added to the original Weka Explorer to start remote data mining tasks

be invoked, and two buttons to start and stop the data mining task on the selected Web Service. Through the GUI a user can both: *i*) start the execution locally by using the "Local" pane; *ii*) start the execution remotely by using the "Remote" pane. Each task in the GUI is managed by an independent thread. Therefore, a 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 (on the right of Figure 2).

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* [9] 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.

2.1 Web Service Operations

Table 1 shows the operations provided by each Web Service in the Weka4WS framework.

Table 1. Operations provided by each Web Service in the Weka4WS framework

Operation	**Description**
`createResource`	Creates a new WS-Resource.
`subscribe`	Subscribes to notifications about resource properties changes.
`destroy`	Explicitly requests destruction of a WS-Resource.
`classification`	Submits the execution of a classification task.
`clustering`	Submits the execution of a clustering task.
`associationRules`	Submits the execution of an association rules task.

The first three operations are related to WSRF-specific invocation mechanisms (described below), whereas the last three operations - `classification`, `clustering` and `associationRules` - are used to require the execution of a specific data mining task. In particular, the `classification` operation provides access to the complete set of classifiers in the Weka Library (currently, 71 algorithms). The `clustering` and `association rules` operations expose all the clustering and association rules algorithms provided by the Weka Library (5 and 2 algorithms, respectively).

To improve concurrency the data mining operations are invoked in an asynchronous way, i.e., the client submits the execution in a non-blocking mode, and results will be notified to the client whenever they have been computed.

Table 2. Input parameters of the Web Service data mining operations

Operation	Parameter	Description
`classification`	`algorithm`	Name of the classification algorithm to be used.
	`arguments`	Arguments to be passed to the algorithm.
	`testOptions`	Options to be used during the testing phase.
	`classIndex`	Index of the attribute to use as the class.
	`dataSet`	URL of the dataset to be mined.
`clustering`	`algorithm`	Name of the clustering algorithm.
	`arguments`	Algorithm arguments.
	`testOptions`	Testing phase options.
	`selectedAttrs`	Indexes of the selected attributes.
	`classIndex`	Index of the class w.r.t. evaluate clusters.
	`dataSet`	URL of the dataset to be mined.
`associationRules`	`algorithm`	Name of the association rules algorithm.
	`arguments`	Algorithm arguments.
	`dataSet`	URL of the dataset to be mined.

Table 2 lists the input parameters of the Web Service data mining operations. Three parameters, in particular, are required in the invocation of all the data mining operations: `algorithm`, `arguments`, and `dataSet`. The `algorithm` argument specifies the name of the Java class in the Weka Library to be invoked (e.g., "*weka.classifiers.trees.J48*"). The `arguments` parameter specifies a sequence of arguments to be passed to the algorithm (e.g., "*-C 0.25 -M 2*"). Finally, the `dataSet` parameter specifies the URL of the dataset to be mined (e.g., "*gsiftp://hostname/path/ad.arff*").

2.2 Task Execution

This section describes the steps that are performed to execute a data mining task on a remote Web Service in the Weka4WS framework.

Figure 3 shows a *Client Module* (*CM*) that interacts with a remote *Web Service* (*WS*) to execute a data mining task. In particular, this example assumes that the CM is requesting the execution of a clustering analysis on a dataset local to the user node, which then acts also as a storage node. Notice that this is a worst case since in several scenarios the dataset is available on the remote computing node. In order to perform this task, the following steps are executed (see Figure 3):

1. **Resource creation**. The CM invokes the `createResource` operation, which creates a new WS-Resource used to maintain the state of the subsequent clustering computation. The state is stored as *properties* of the resource. In particular, a "clustering model" property is used to store the result of the clustering computation. The WS returns the *endpoint reference* (*EPR*) of the created resource. The EPR is unique within the WS, and distinguishes

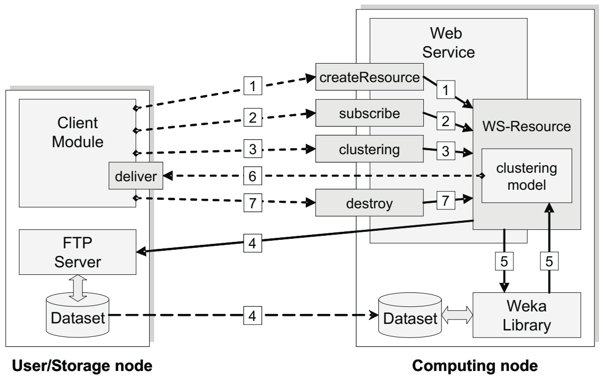


Fig. 3. Execution of a data mining task on a remote Web Service

this resource from all other resources in that service. Subsequent requests from the CM will be directed to the resource identified by that EPR.

2. **Notification subscription**. The CM invokes the `subscribe` operation, which subscribes for notifications about changes that will occur to the "clustering model" resource property. Whenever this property will change its value (i.e., whenever the model has been computed), the CM will receive a notification containing that value, which represents the result of the computation.
3. **Task submission**. The CM invokes the `clustering` operation to require the execution of the clustering analysis. This operation receives six parameters as shown in Table 2, among which the name of the clustering algorithm and the URL of the dataset to be mined. The operation is invoked in an asynchronous way, i.e., the client may proceed its execution without waiting for the completion of the operation.
4. **Dataset download**. Since in this example the dataset is assumed not available on the computing node, the WS downloads the dataset to be mined from the URL specified in the `clustering` invocation. The download request is directed to an FTP server running on the user node. Note that different protocols could be used, such as HTTP or GridFTP, as mentioned before.
5. **Data mining**. After the dataset has been downloaded to the computing node, the clustering analysis is started by invoking the appropriate Java class in the Weka Library. The execution is handled within the WS-Resource created on Step 1, and the result of the computation (i.e., the inferred model) is stored in the "clustering model" property.
6. **Results notification**. Whenever the "clustering model" property has been changed, its new value is notified to the CM, by invoking its implicit `deliver` operation. This mechanism allows for the asynchronous delivery of the execution results whenever they are generated.
7. **Resource destruction**. The CM invokes the `destroy` operation, which destroys the WS-Resource created on Step 1.

The next section presents a performance analysis of the execution mechanisms described above.

3 Performance Analysis

To evaluate the efficiency of the proposed system, we carried out a performance analysis of Weka4WS for executing a typical data mining task in different network scenarios. In particular, we evaluated the execution times of the different steps needed to perform the overall data mining task, as described at the end of the previous section. The main goal of our analysis is to evaluate the overhead introduced by the WSRF mechanisms with respect to the overall execution time.

For our analysis we used the *census* dataset from the UCI repository [10]. Through random sampling we extracted from it ten datasets, containing a number of instances ranging from 1700 to 17000, with a size ranging from 0.5 to 5 MB. We used Weka4WS to perform a clustering analysis on each of these datasets. In particular, we used the *Expectation Maximization* (*EM*) clustering algorithm, using 10 as the number of clusters to be identified on each dataset.

The clustering analysis on each dataset was executed in two network scenarios:

- **LAN**: the computing node N_c and the user/storage node N_u are connected by a LAN network, with an average bandwidth of 94.4 Mbps and an average round-trip time (RTT) of 1.4 ms. Both N_c and N_u machines are Pentium4 2.4 GHz with 1 GB RAM.
- **WAN**: the computing node N_c and the user/storage node N_u are connected by a WAN network, with an average bandwidth of 213 kbps and an average RTT of 19 ms. N_c is an Pentium4 2.4 GHz with 1 GB RAM, whereas N_u is an Athlon 2.14 GHz with 512 MB RAM.

For each dataset size and network scanario we run 20 independent executions. The values reported in the following graphs are computed as an average of the values measured in the 20 executions.

Figure 4 represents the execution times of the different steps of the clustering task in the LAN scenario for a dataset size ranging from 0.5 to 5 MB. As shown in the figure, the execution times of the WSRF-specific steps are independent from the dataset size, namely: *resource creation* (1698 ms, on the average), *notification subscription* (275 ms), *task submission* (342 ms), *results notification* (1354 ms), and *resource destruction* (214 ms).

On the contrary, the execution times of the *dataset download* and *data mining* steps are proportional to the dataset size. In particular, the execution time of the *dataset download* ranges from 218 ms for 0.5 MB to 665 ms for 5 MB, while the *data mining* execution time ranges from 107474 ms for the dataset of 0.5 MB, to 1026584 ms for the dataset of 5 MB. The total execution time ranges from 111798 ms for the dataset of 0.5 MB, to 1031209 ms for the dataset of 5 MB. Note that in Figure 4 the lines representing the total execution time and

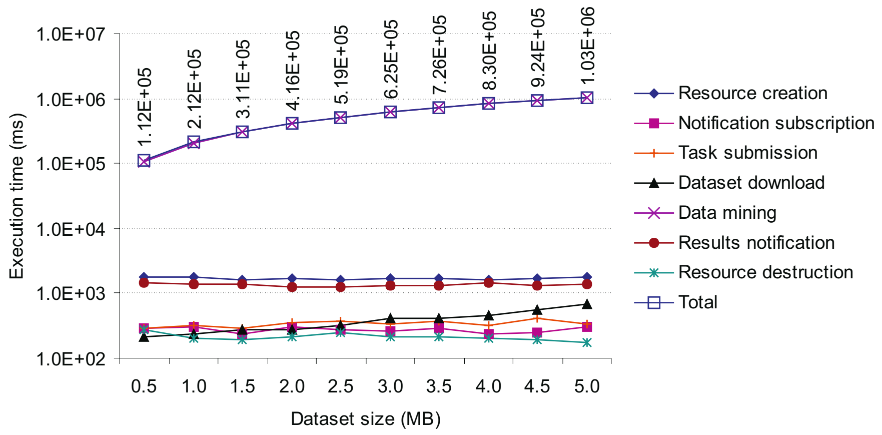


Fig. 4. Execution times of the different steps of the clustering task in the LAN scenario

the *data mining* execution time appear coincident, because the *data mining* step takes from 96% to 99% of the total execution time, as discussed below.

Figure 5 represents the execution times of the different steps in the WAN scenario. The execution times of the WSRF-specific steps are similar to those measured in the LAN scenario. The only significant difference is the execution time of the *results notification* step, which passes from an average of 1354 ms in the LAN scenario to an average of 2790 ms in the WAN scenario, due to additional time needed to transfer the clustering model through a low-speed network. For the same reason, the transfer of the dataset to be mined requires an execution time significantly greater than the one measured in the LAN scenario. In particular, the execution time of the *dataset download* step in the WAN scenario ranges from 14638 ms for 0.5 MB to 132463 ms for 5 MB.

The *data mining* execution time is similar to that measured in the LAN scenario, since the clustering analysis is executed on an identical computing node, as mentioned before. Mainly due to the additional time required by the *dataset download* step, the total execution time is greater than the one measured in the LAN scenario, ranging from 130488 ms for the dataset of 0.5 MB to 1182723 ms for the dataset of 5 MB. Like Figure 4, the line of the total execution time is very close to the line of the *data mining* execution time.

To better highlight the overhead introduced by the WSRF mechanisms and the distributed scenario, Figure 6 and Figure 7 show the percentage of the execution times of the *data mining*, *dataset download*, and the other steps (i.e., *resource creation*, *notification subscription*, *task submission*, *results notification*, *resource destruction*), with respect to the total execution time in the LAN and WAN scenarios.

In the LAN scenario (see Figure 6) the *data mining* step represents from 96.13% to 99.55% of the total execution time, the *dataset download* ranges from 0.19% to 0.06%, and the other steps range from 3.67% to 0.38%. Notice that the *data mining* time corresponds to total Weka execution time on a single node.

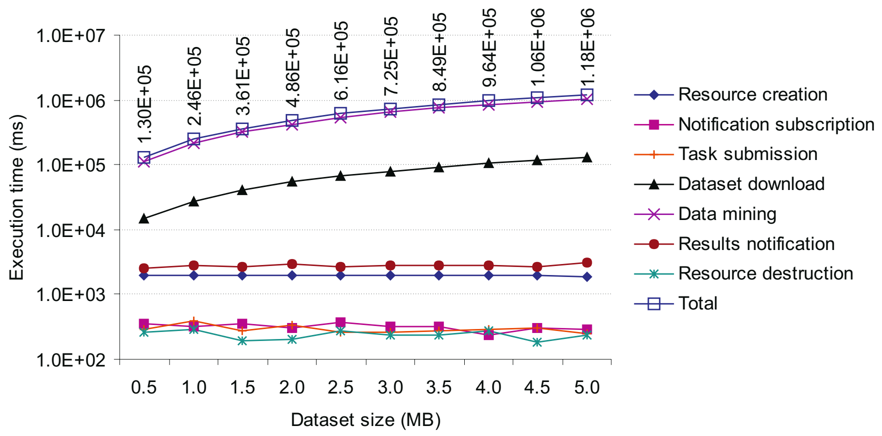


Fig. 5. Execution times of the different steps of the clustering task in the WAN scenario

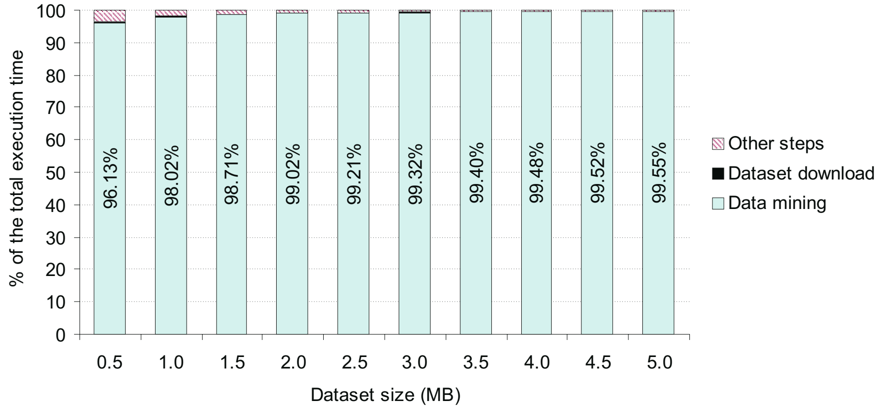


Fig. 6. Percentage of the execution times of the different steps in the LAN scenario

In the WAN scenario (see Figure 7) the *data mining* step represents from 84.62% to 88.32% of the total execution time, the *dataset download* ranges from 11.22% to 11.20%, while the other steps range from 4.16% to 0.48%.

We can observe that in the LAN scenario neither the *dataset download* nor the other steps represent a significant overhead with respect to the total execution time. In the WAN scenario, on the contrary, the *dataset download* is a critical step that can significantly affect the overall execution time. For this reason, the use of high-performance file transfer protocols such as GridFTP can be of great importance.

The performance analysis discussed above demonstrates the efficiency of the WSRF mechanisms as a means to execute data mining tasks on remote machines. By exploiting such mechanisms, Weka4WS can provide an effective way to perform compute-intensive distributed data analysis on a large-scale Grid environment.

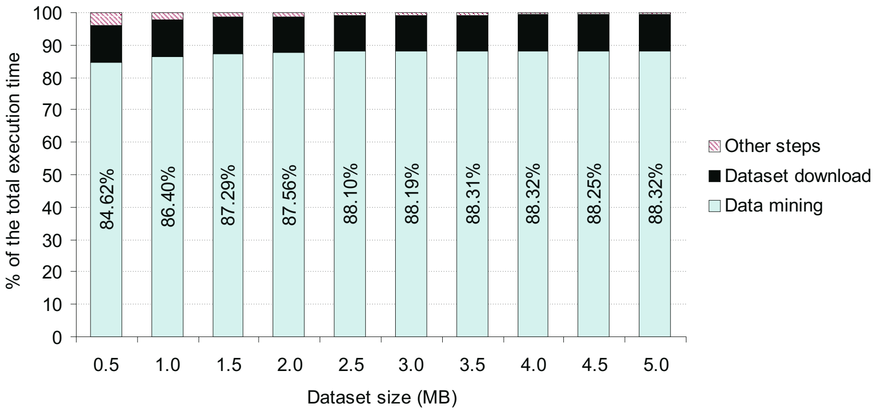


Fig. 7. Percentage of the execution times of the different steps in the WAN scenario

4 Related Work

The idea of adapting the Weka toolkit to a Grid environment has been recently explored, although none of the proposed systems makes use of WSRF as enabling technology.

Grid Weka [11] modifies the Weka toolkit to enable the use of multiple computational resources when performing data analysis. In this system, a set of data mining tasks can be distributed across several machines in an ad-hoc environment. Tasks that can be executed using Grid Weka include: building a classifier on a remote machine, labelling a dataset using a previously built classifier, testing a classifier on a dataset, and cross-validation. Even if Grid Weka provides a way to use multiple resources to execute distributed data mining tasks, it has been designed to work within an ad-hoc environment, which does not constitute a Grid per se. In particular, the invocation of remote resources in the Weka Grid framework is not service-oriented, and makes use of ad-hoc solutions that do not take into considerations fundamental Grid aspects (e.g., interoperability, security, etc.). On the contrary, Weka4WS exposes all its functionalities as WSRF-compliant Web Services, which enable important benefits, such as dynamic service discovery and composition, standard support for authorization and cryptography, and so on.

FAEHIM (Federated Analysis Environment for Heterogeneous Intelligent Mining) [12] is a Web Services-based toolkit for supporting distributed data mining. This toolkit consists of a set of data mining services, a set of tools to interact with these services, and a workflow system used to assemble these services and tools. The Triana problem solving environment [13] is used as the workflow system. Data mining services are exposed as Web Services to enable an easy integration with other third party services, allowing data mining algorithms to be embedded within existing applications. Most of the Web Services in FAEHIM are derived from the Weka library. All the data mining algorithms available in Weka were converted into a set of Web Services. In particular, a general “Classifier Web Service” has been implemented to act as a wrapper for a complete set of classifiers in Weka, a “Clustering Web Service” has been used to wrap a variety of clustering algorithms, and so on. This service-oriented approach is similar to that adopted in Weka4WS. However, in Weka4WS standard WSRF mechanisms are used for managing remote tasks execution and asynchronous results notification (which is missing in that system), and all the algorithms are exposed on every node as a single WSRF-compliant Web Service to facilitate the deployment in a large Grid environment.

WekaG [14] is another adaptation of the Weka toolkit to a Grid environment. WekaG is based on a client/server architecture. The server side defines a set of *Grid Services* that implement the functionalities of the different algorithms and phases of the data mining process. A WekaG client is responsible for communicating with Grid Services and offering the interface to users. A prototype that implements the capabilities of the *Apriori* algorithm has been developed using Globus Toolkit 3. In this prototype an *Apriori Grid Service* has been developed to produce association rules from a dataset, while GridFTP is used for the deployment of the files to the Grid Service node. WekaG shares this service-orientation with Weka4WS. However, the WekaG prototype provides access to only one data mining algorithm (Apriori), whereas Weka4WS currently provides access to 78 different Weka algorithms through a single Web Service interface. Moreover, WekaG uses the old Grid Service technology [15], which - differently from WSRF - is largely incompatible with current Web Service and Grid computing standards.

5 Conclusions

Weka4WS adopts the emerging Web Services Resource Framework (WSRF) for accessing remote data mining algorithms and composing distributed KDD applications.

The paper described the design and the implementation of Weka4WS using a first release of the WSRF library. To evaluate the efficiency of the proposed system, a performance analysis of Weka4WS for executing a distributed data mining task in different network scenarios has been also discussed.

The experimental results demonstrate the efficiency of the WSRF mechanisms as a means to execute data mining tasks on remote resources. By exploiting such mechanisms, Weka4WS can provide an effective way to perform compute-intensive distributed data analysis on large-scale Grids. The Weka4WS software prototype will be made available to the research community.

Acknowledgements

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). This work has been also supported by the Italian MIUR FIRB Grid.it project RBNE01KNFP on High Performance Grid Platforms and Tools.

References

1. Curcin, V., Ghanem, M., Guo, Y., Kohler, M., Rowe, A., Syed, J., Wendel, P.: Discovery Net: Towards a Grid of Knowledge Discovery. 8th Int. Conf. on Knowledge Discovery and Data Mining (2002).
2. Brezany, P., Hofer, J., Tjoa, A. M., Woehrer, A.: Towards an open service architecture for data mining on the grid. Conf. on Database and Expert Systems Applications (2003).
3. Skillicorn, D., Talia, D.: Mining Large Data Sets on Grids: Issues and Prospects. Computing and Informatics, vol. 21 n. 4 (2002) 347-362.
4. Cannataro, M., Talia, D.: The Knowledge Grid. Communications of the ACM, vol. 46 n. 1 (2003) 89-93.
5. Witten, H., Frank, E.: Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann (2000).
6. Czajkowski, K. et al: The WS-Resource Framework Version 1.0 (2004). http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.
7. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid. In: Berman, F., Fox, G., A. Hey, A. (Eds.), Grid Computing: Making the Global Infrastructure a Reality, Wiley (2003) 217-249.
8. Foster, I.: A Globus Primer (2005). http://www.globus.org/primer.
9. Allcock, B., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., Foster, I.: The Globus Striped GridFTP Framework and Server. Conf. on Supercomputing (SC'05) (2005).
10. The UCI Machine Learning Repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
11. Khoussainov, R., Zuo, X., Kushmerick, N.: Grid-enabled Weka: A Toolkit for Machine Learning on the Grid. ERCIM News, n. 59 (2004).
12. Shaikh Ali, A., Rana, O. F., Taylor, I. J.: Web Services Composition for Distributed Data Mining. Workshop on Web and Grid Services for Scientific Data Analysis (2005).
13. The Triana Problem Solving Environment. http://www.trianacode.org.
14. Prez, M. S., Sanchez, A, Herrero, P, Robles, V., Pea. J. M.: Adapting the Weka Data Mining Toolkit to a Grid based environment. 3rd Atlantic Web Intelligence Conf. (2005).
15. Tuecke, S. et al.: Open Grid Services Infrastructure (OGSI) Version 1.0 (2003). http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf.