The Weka4WS framework for distributed data mining in service-oriented Grids

Domenico Talia, Paolo Trunfio[†,*], Oreste Verta

DEIS, University of Calabria
Via P. Bucci 41C
87036 Rende (CS), Italy

SUMMARY

The service oriented architecture (SOA) paradigm can be exploited for the implementation of data and knowledge-based applications in distributed environments. The Web Services Resource Framework (WSRF) has recently emerged as the standard for the implementation of Grid services and applications. WSRF can be exploited for developing high-level services for distributed data mining applications. This paper describes Weka4WS, a framework that extends the widely used open source Weka toolkit to support distributed data mining on WSRF-enabled Grids. Weka4WS adopts the WSRF technology for running remote data mining algorithms and managing distributed computations. The Weka4WS user interface 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 the WSRF libraries and services provided by Globus Toolkit 4. A performance analysis of Weka4WS for executing distributed data mining tasks in different network scenarios is presented.

KEY WORDS: Grid computing; Distributed data mining; Web Services Resource Framework

INTRODUCTION

In enterprise and scientific scenarios handling and analyzing distributed data sources through Grid systems is as important as delivering high-performance computing. Complex business and scientific applications are exploiting Grids to access distributed resources (e.g., computers, databases, networks, etc.) and run decentralized applications operating on dispersed data and resources. In fact, Grids have been designed to support applications that can benefit

[†]E-mail: trunfio@deis.unical.it
[*]Correspondence to: Paolo Trunfio, DEIS, University of Calabria, Via P. Bucci 41C, 87036 Rende (CS), Italy

from high performance, distribution, collaboration, data sharing and complex interaction of autonomous and geographically dispersed resources. Recently, a few Grid-based data mining systems have been proposed including general frameworks, abstract models, and domain-specific implementations [1, 2, 3, 4].

Through a service-oriented model, application and system functions can be distributed among several computers or domains for improving efficiency or for necessity. By using this approach, data-intensive and knowledge discovery applications can be developed through the exploitation of Grid technology for delivering high performance and managing data and knowledge distribution. Our focus here is on distributed data mining services that allow decentralized teams to analyze data in a high-level, standard and reliable way. Members of a decentralized team can be part of a physical organization (e.g., a company having more sites) or a *virtual organization* [5] composed of a set of people belonging to different physical organizations that share resources and goals.

This paper describes *Weka4WS*, a framework that extends the widely used Weka toolkit [6] 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. 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 concurrently executed on decentralized Grid nodes by exploiting data distribution and improving application performance.

Data mining algorithms for classification, clustering and association rules in Weka4WS can be executed on remote Grid resources. To enable remote invocation, all the data mining algorithms provided by the Weka library are exposed through 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 with standard Grid environments, the Weka4WS Web services have been implemented by using the *Web Services Resource Framework* (*WSRF*) [7] as enabling technology. As WSRF implementations are recent, to the best of our knowledge, *Weka4WS* is the first data mining framework exploiting the WSRF technology.

In the following we describe the design and implementation of Weka4WS. To evaluate the overhead introduced by the service invocation mechanisms and its effects on the efficiency of the proposed system, we also present a performance analysis of Weka4WS executing distributed data mining tasks in different network scenarios. This work is one of the first performance evaluations of WSRF. The work presented in this paper is an extended version of the work presented in [8]. This new version describes an improved implementation of the framework, includes a wider set of experimental results, and outlines system extensions on which we are currently working.

WSRF AND THE WEKA4WS FRAMEWORK

WSRF is a family of technical specifications concerned with the creation, addressing, inspection and lifetime management of *stateful resources* using Web services [7].

A Web service is a software component that can be accessed by remote entities (clients or other services) using standard Internet protocols such as HTTP. The capabilities offered by a service are defined using the *Web Services Description Language* (*WSDL*), an XML-based formalism that allows to define the operations exposed by a Web service, as well as specifying the input and output messages that must be exchanged to invoke such operations. The set of operations and associated messages constitute the *interface* of a service.

An important feature of Web services is the independence of the service interface from the implementation of the operations. To invoke a Web service, a remote entity needs to know only its WSDL interface, without worrying about the actual programming language used to implement its operations. This allows to couple in an easy way distributed software components implemented using different languages and running on heterogeneous platforms.

Web services in Grid computing are used as uniform interfaces for accessing remote resources and composing distributed applications, independently from their location and specific implementation. The so-called *Open Grid Services Architecture* (*OGSA*) [9] defines an architectural model for Grid systems in which distributed resources and applications are modelled as Web services that interact each other using Internet-based standards.

WSRF implements the OGSA philosophy by defining a set of Web service standards for the implementation of Grid systems. As mentioned above, WSRF focuses on managing stateful resources using Web services. The combination of a stateful resource with a Web service is termed *WS-Resource*. The possibility to define a "state" associated to a Web service is the most important difference between WSRF-compliant Web services and pre-WSRF ones. This is a key feature in implementing Grid systems, because Grid applications can be composed by multiple long-running processes, whose state needs to be accessed and monitored to control the overall execution. In this context, WS-Resources provide a standard way to represent, advertise, and access properties associated to processes as required by complex Grid applications.

Currently, WSRF includes the following standards:

- *WS-Resource Lifetime*: Defines mechanisms for managing the lifecycle of WS-Resources, including their creation and destruction.
- *WS-Resource Properties*: Defines types and values of the components of a WS-Resource that can be accessed or modified by service requestors through the Web service interface. The term *resource property* is used to refer to an individual component of a WS-Resource.
- *WS-Renewable References*: Defines mechanisms to retrieve an updated version of the *endpoint reference* (*EPR*) of a WS-Resource. Each EPR is globally unique and includes the URL of the Web service and a unique identifier of the resource it is associated to.
- *WS-Service Group*: Defines a means by which Web services and WS-Resources can be aggregated or grouped together for a specific purpose (e.g., for service/resource discovery).
- *WS-Base Faults*: Defines a standard exception type, along with rules for how this exception type is used by Web services.

A separate standard, called *WS-Notification* [10], defines a publish-subscribe notification model for Web services that is exploited to notify interested clients and/or services about changes that occur to the state of a WS-Resource. The combination of the WSRF and WS-Notification mechanisms are exploited in Grids to build long-lived distributed applications, in which the state of the computation is managed across multiple nodes, and services cooperate by exchanging messages in a highly-distributed way.

While WSRF and WS-Notification are currently adopted as the standard implementation of the OGSA model, some researchers [11] have evaluated the feasibility of an alternative implementation of OGSA based on two different standards: *WS-Transfer* and *WS-Eventing*.

WS-Transfer [12] defines mechanisms for acquiring XML-based representations of resources using the Web service infrastructure. Specifically, it defines operations for sending and receiving the representation of a given resource, and operations for creating and deleting a resource and its corresponding representation. WS-Eventing [13] defines a notification protocol that allows clients to request asynchronous delivery of event messages generated by Web services. Standardized messages are defined to allow clients to subscribe to unsubscribe from these event sources.

The comparative study proposed in [11] shows a substantial equivalence of WSRF/WS-Notification and WS-Transfer/WS-Eventing in terms of functionality and implied performance, arguing the feasibility of an alternative implementation of OGSA based on the WS-Transfer/WS-Eventing specifications. According to this study, it should be relatively easy to transform a WSRF/WS-Notification-based application, such as Weka4WS, into a WS-Transfer/WS-Eventing-based application, without affecting general design and performance.

One of the most adopted implementations of the WSRF specifications has been developed by the Globus Alliance in the context of the Globus project. The Globus Toolkit 4 (GT4) [14] provides both an implementation of WSRF as an open source library and a set of WSRF-compliant services that can be used to create Grid applications.

The Weka4WS system described here has been developed by using the Java WSRF library provided by GT4. Moreover, all nodes involved in Weka4WS applications use the GT4 services for standard Grid functionality, such as security, data management, and so on. These nodes are distinguished in two categories on the basis of the available Weka4WS components: *user nodes* that are the local machines providing the Weka4WS client software; and *computing nodes* that provide the Weka4WS Web services allowing for the execution of remote data mining tasks. Data can be located on computing nodes, user nodes, or third-party nodes (e.g., shared data repositories). If a dataset to be mined is not available on a computing node, it can be uploaded by means of the GT4 data management services. Figure 1 shows the software components of user nodes and computing nodes in the Weka4WS framework.

Computing nodes include two components: a *Web Service* (*WS*) and the *Weka Library* (*WL*). The WS exposes all the data mining algorithms provided by the underlying WL. Therefore, requests to the WS are executed by invoking the corresponding WL algorithms.

User nodes include three components: *Graphical User Interface* (*GUI*), *Client Module* (*CM*), and *Weka Library* (*WL*). The GUI is an extended version of the Weka Explorer environment to support 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.

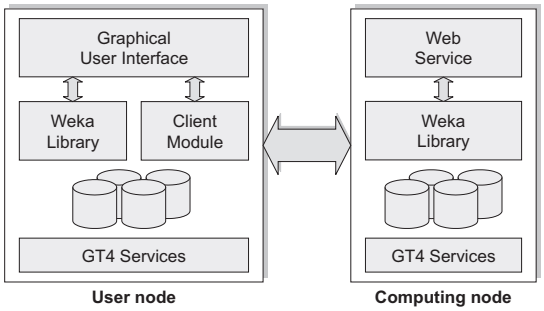

Figure 1. Software components of user nodes and computing nodes.

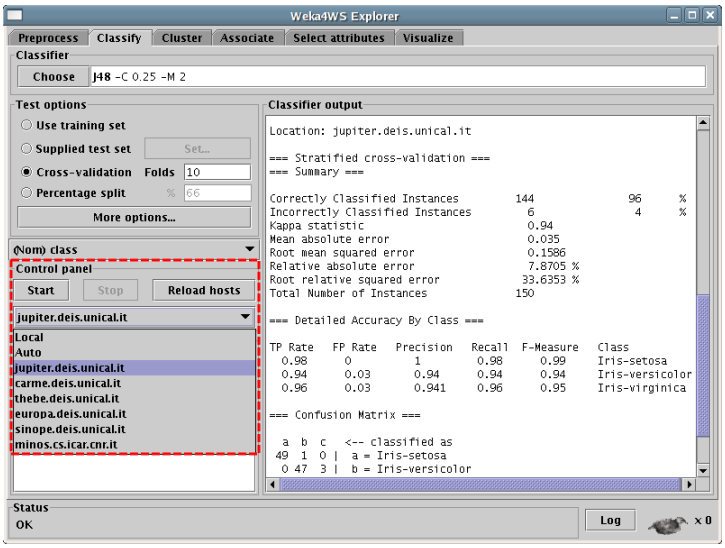

Figure 2. The Weka Explorer extended with a Control Panel that allows to choose the Grid node to which submit the data mining task.

Figure 2 shows a snapshot of the current GUI implementation. As highlighted in the broken box, a *Control Panel* has been added to the original Weka Explorer environment. This panel includes the following components: *i*) a drop-down list to choose the Grid node to which submit the data mining task; *ii*) a *Reload hosts* button to update the list of available computing nodes; *iii*) the *Start* and *Stop* buttons to start and stop the data mining task.

Through the drop-down list the user can select one of the following items:

Table I. Operations provided by each Web service in the Weka4WS framework.

Operation	Description
`createResource`	Creates a new stateful resource.
`subscribe`	Subscribes to notifications about resource properties changes.
`destroy`	Explicitly requests destruction of a 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.

- *Local*: the task will be executed on the local machine, as in the standard Weka environment.
- *Auto*: the task will be submitted to one of the listed hosts; the host is selected automatically using a round-robin strategy (i.e., at each invocation, the next host in the list is chosen).
- *One of the listed hosts*: the task will be submitted to the Grid node selected by the user.

Each data mining task in the GUI is managed by an independent thread. Therefore, a user can start multiple tasks in parallel on different nodes, this way taking full advantage of the distributed Grid environment. Whenever the output of a data mining task has been received from a remote computing node, it is visualized in the standard *Output* panel (on the right of Figure 2).

Web service operations

Table I lists the operations provided by each Web service in the Weka4WS framework. 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 `associationRules` operations expose all the clustering and association rules algorithms provided by the Weka library (5 and 2 algorithms, respectively).

To improve concurrency, data mining task are invoked in an asynchronous way, that is, the client submits the task in a non-blocking mode and results are notified to it as soon as they are computed.

Table II lists the input parameters of the `classification`, `clustering`, and `associationRules` data mining operations. Four parameters are required in the invocation of all the data mining operations: `algorithm`, `arguments`, `dataSet`, and `crcValue`. 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

Table II. 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.
	`crcValue`	Checksum value of the dataset to be mined.
`clustering`	`algorithm`	Name of the clustering algorithm.
	`arguments`	Algorithm arguments.
	`testOptions`	Testing phase options.
	`selectedAttrs`	Indices of the selected attributes.
	`classIndex`	Index of the class w.r.t. evaluate clusters.
	`dataSet`	URL of the dataset to be mined.
	`crcValue`	Checksum value 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.
	`crcValue`	Checksum value of the dataset to be mined.

be passed to the algorithm (e.g., "*-C 0.25 -M 2*"). The `dataSet` parameter specifies the URL of the dataset to be mined, and the `crcValue` parameter specifies its checksum.

Task execution

This section describes the steps 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 classification task on a dataset located on the user node. Notice that this is a worst case, since in many scenarios the datasets to be mined may be already available (e.g., replicated) on most computing nodes.

The following steps are executed in order to perform this task, (see Figure 3):

1. **Resource creation**. The CM invokes the `createResource` operation to creates a new resource that will maintain the state of the subsequent classification analysis. The state is stored as *properties* of the resource. In this example, a *model* property is used to store the result of the classification task. The WS returns the *endpoint reference* (*EPR*) of the created resource. The EPR is globally unique and distinguishes this resource from all other resources over the Grid. Subsequent requests from the CM will be directed to the resource identified by that EPR.

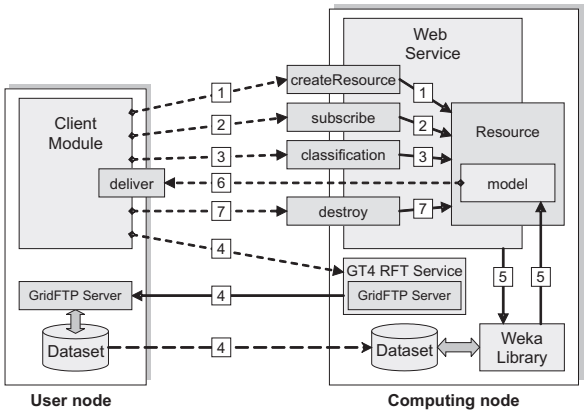

Figure 3. Execution of a data mining task on a remote Web service.

2. **Notification subscription**. The CM invokes the `subscribe` operation that subscribes to notifications about changes that will occur to the *model* resource property. As soon as this property will change its value (i.e., as soon as the model has been computed), the CM will receive a notification containing that value, which represents the result of the classification task.
3. **Task submission**. The CM invokes the `classification` operation requiring the execution of the classification task. This operation receives a set of parameters as shown in Table II, among which the name of the classification algorithm to be used, the URL of the dataset to be mined and its checksum. If a copy of the dataset is not already available on the computing node, this operation returns the URL where the dataset has to be uploaded.
4. **File transfer**. Since in this example we assume that the dataset is not already available on the computing node, the CM requests to transfer it to the URL specified as a return value by the `classification` operation. The transfer request is managed by the GT4 *Reliable File Transfer* (*RFT*) service running on the computing node, which in turn invokes the *GridFTP* servers [15] running on the user and computing nodes. If the size of the dataset is over a given threshold, it is compressed before the transfer and decompressed at destination.
5. **Data mining**. The classification analysis is started by the WS through the invocation of the appropriate Java class in the Weka library. The result of the computation (i.e., the inferred model) is stored in the *model* property of the resource created on Step 1.
6. **Results notification**. As soon as the *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 as soon as they are generated.

7. **Resource destruction**. The CM invokes the `destroy` operation, which eliminates the resource created on Step 1.

PERFORMANCE EVALUATION

To evaluate the performance of the system, we carried out some experiments where we used Weka4WS for executing different data mining tasks in two network scenarios:

- **Local Area Grid (LAG)**: the computing node and the user node are connected by a local area network, with an average bandwidth of 41.2MB/s.
- **Wide Area Grid (WAG)**: the computing node and the user node are connected by a wide area network, with an average bandwidth of 52.7kB/s.

In the following we discuss performance results obtained by executing *clustering* and *classification* data mining tasks on publicly available datasets. The main goal of our analysis is to evaluate the overhead introduced by the WSRF mechanisms and the distributed scenario with respect to the overall execution time.

Clustering experiments

For our clustering experiments we used the *USCensus1990* dataset [†] from the UCI KDD Archive [16]. The dataset contains data extracted from the U.S. Census Bureau Web site as part of the 1990 U.S. census. We extracted from it ten datasets containing a number of instances ranging from 143000 to 1430000, with a size ranging from 20 to 200 MB. We used Weka4WS to execute the *Expectation Maximization* (*EM*) clustering algorithm on each of these datasets and asked the system to group data in 5 clusters on the basis of 10 selected attributes.

As mentioned before, the clustering analysis was executed both in the LAG and WAG scenarios. For each dataset size and network scenario we run 20 independent executions. The measures reported in the following graphs resulted from the average values of all executions.

Figure 4 shows the execution times of the different steps in the LAG scenario for a dataset size ranging from 20 to 200 MB. As shown in the figure, the execution times of the WSRF-specific steps are independent from the dataset size, namely: *resource creation* (1681 ms, on the average), *notification subscription* (261 ms), *task submission* (289 ms), *results notification* (1347 ms), and *resource destruction* (207 ms).

On the contrary, the execution times of the *file transfer* and *data mining* steps are proportional to the dataset size. In particular, the execution time of the *file transfer* ranges from 2.8 s for 20 MB to 27.7 s for 200 MB, while the *data mining* execution time ranges from 254 s for the dataset of 20 MB, to 3709 s for the dataset of 200 MB. The total execution time (not shown in the figure) ranges from 261 s for 20 MB, to 3740 s for 200 MB.

[†] http://kdd.ics.uci.edu/databases/census1990/USCensus1990.html [19 March 2007]

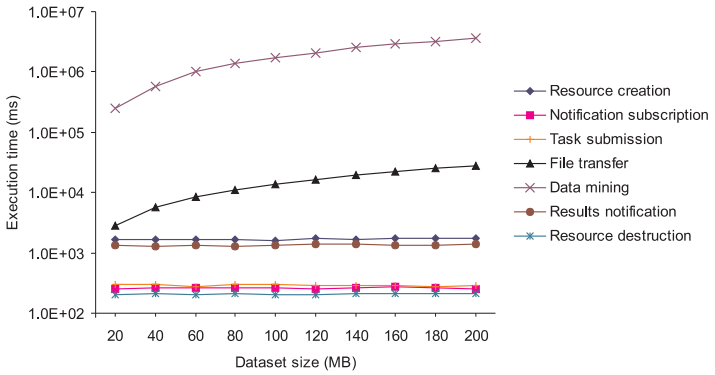

Figure 4. Execution times of the clustering task in the LAG scenario.

Figure 5 shows the execution times of all steps in the WAG scenario. The execution times of the WSRF-specific steps are pretty equal to those measured in the LAG scenario. The main difference is the execution time of the *results notification* step, which in the LAG scenario is on average of 1347 ms whereas in the WAG scenario is equal to 2671 ms, 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 required an execution time significantly greater than the one measured in the LAN scenario. In particular, the execution time of the *file transfer* step in the WAG scenario ranges from 63.3 s for 20 MB to 627.9 s for 200 MB, taking into account that datasets are transferred in compressed form, as mentioned in the previous section.

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. Mainly due to the additional time required by the *file transfer* step, the total execution time (not shown in figure) is greater than that measured in the LAG scenario; it ranges from 320 s for the 20 MB dataset to 4244 s for the 200 MB dataset.

Classification experiments

For data classification experiments we used the *kddcup99* dataset [‡] available at the UCI archive. This dataset, used for the KDD'99 Competition, contains a wide set of data produced during seven weeks of monitoring in a military network environment subject to simulated intrusions. As before, we extracted ten datasets from it, with a number of instances ranging

[‡]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [19 March 2007]

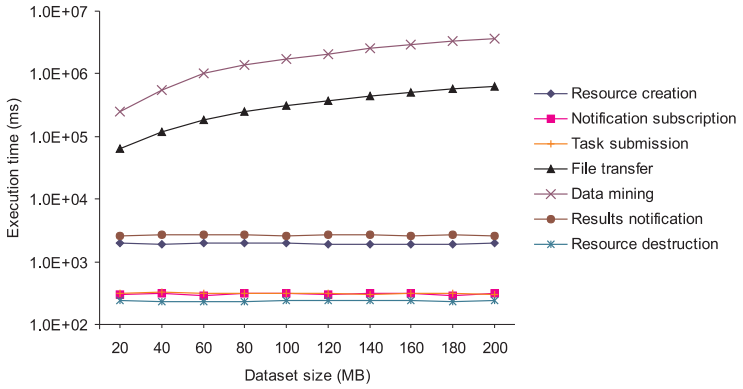

Figure 5. Execution times of the clustering task in the WAG scenario.

from 172000 to 1720000 and a size ranging from 25 to 250 MB, and then we used Weka4WS to perform a classification analysis on each of those datasets. In particular, we employed the *J48* classification algorithm, using 5-folds cross-validation based on 10 attributes.

Figure 6 shows the execution times of the different steps of the classification task in the LAG scenario for a dataset size ranging from 25 to 250 MB. As highlighted in the clustering experiments, the execution times of the WSRF-specific steps are independent from the dataset size, whereas the execution times of the *file transfer* and *data mining* steps are proportional to the dataset size.

In particular, the execution time of the *file transfer* ranges from 1.0 s for 25 MB to 13.2 s for 250 MB, while the *data mining* execution time ranges from 301 s for the dataset of 25 MB, to 2038 s for the dataset of 250 MB. The total execution time ranges from 349 s for the dataset of 25 MB, to 2055 s for the dataset of 250 MB.

Figure 7 shows the execution times in the WAG scenario. As expected and noted in the clustering experiments, the *file transfer* step requires an execution time significantly greater than in the LAG scenario. In this case times range from 27.8 s for 25 MB to 201.7 s for 250 MB.

The *data mining* execution time is similar to that measured in the LAG scenario, since the classification analysis is executed on an identical computing node, as mentioned before. The total execution time in this case ranges from 65 s for the dataset of 25 MB to 2256 s for the dataset of 250 MB.

To better highlight the overhead introduced by the WSRF mechanisms and the distributed scenario, Figure 8 shows the percentage of *data mining*, *file transfer*, and *WSRF overhead* (i.e., the sum of *resource creation*, *notification subscription*, *task submission*, *results notification*, and *resource destruction* steps), with respect to the total execution time in the LAG and WAG

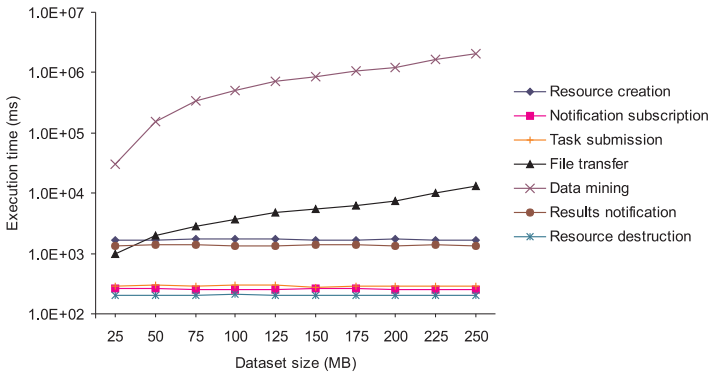

Figure 6. Execution times of the classification task in the LAG scenario.

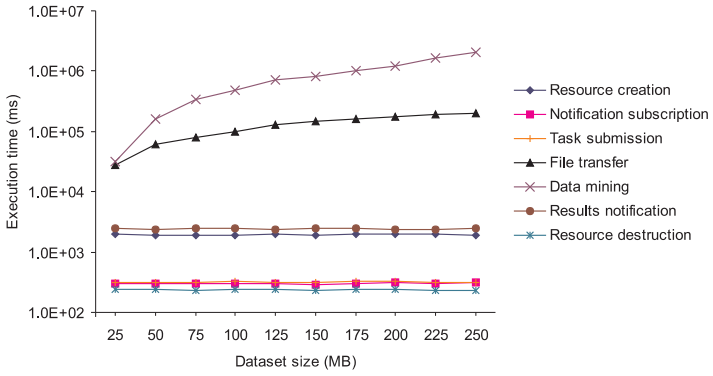

Figure 7. Execution times of the classification task in the WAG scenario.

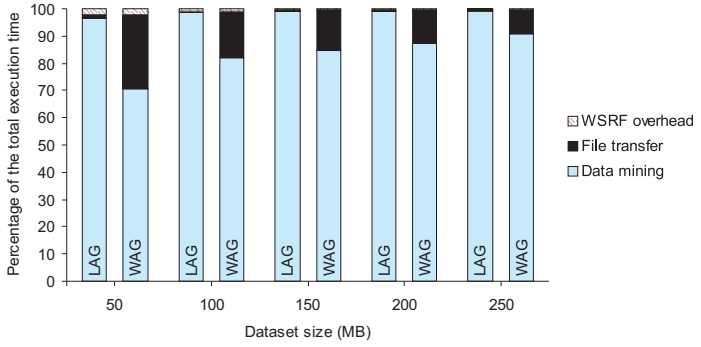

Figure 8. Percentage of the execution times of the different steps of the classification task in the LAG and WAG scenarios.

scenarios. For space reasons, the values reported in this figure refers only to the datasets of 50, 100, 150, 200 and 250 MB.

In the LAG scenario the *data mining* step takes the 96.39% of the total execution time for the dataset of 50 MB, whereas it takes the 99.18% of the total execution time for the dataset of 250 MB. At the same time, the *file transfer* ranges from 1.25% to 0.64%, and the WSRF overhead range from 2.36% to 0.18%. In the WAG scenario the *data mining* step takes from 70.75% to 90.83% of the total execution time, the *file transfer* ranges from 26.95% to 8.94%, while the WSRF overhead range from 2.30% to 0.23%.

We can observe that in the LAG scenario neither the *file transfer* nor the WSRF overhead represent a significant overhead with respect to the total execution time. In the WAG scenario the *file transfer* is a critical step only in relatively small datasets, since in these cases the *data mining* step is very fast. However, in most scenarios the data mining step is a very time-consuming task, so the *file transfer* step - if needed - is marginal when large datasets must be transferred, as shown in the figure.

As a concluding remark, 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.

APPLICATION EXAMPLE

The ability of Weka4WS to manage multiple concurrent tasks on different machines allows users to implement a wide range of data mining applications that take advantage of the distributed computing power of a Grid.

A class of applications that can efficiently exploit the Weka4WS approach is that of a single dataset analyzed in parallel on multiple Grid nodes using different data mining algorithms. For example, a given dataset could be concurrently classified using different classification algorithms with the aim of finding the "best" classifier on the basis of some evaluation criteria (e.g., error rate, confusion matrix, etc.).

A variant of this class of applications is that of a single dataset analyzed using multiple instances of the same algorithm with different parameters (parameter sweeping). In the following we describe an example of application in which a real dataset is analyzed with Weka4WS by running multiple instances of the same clustering algorithm, with the goal of obtaining multiple clustering models from the same data source.

The *covertype* dataset [§] from the UCI archive has been used as data source. The dataset has a size of about 72 MB and contains information about forest cover type for 581012 sites in the United States. Each dataset instance, corresponding to a site observation, is described by 54 attributes that give information about the main features of a site (e.g., elevation, aspect, slope, etc.). The 55th attribute contains the cover type, represented as an integer in the range 1 to 7.

Weka4WS has been used to run an application in which 6 independent instances of the *KMeans* algorithm [17] perform a different clustering task on the *covertype* dataset. In particular, each *KMeans* instance has been asked to group data into a given number of clusters, ranging from 2 to 7, based on all the attributes but the last one (the cover type). The same application has been executed using a number of computing nodes ranging from 1 to 6 in order to evaluate the speedup of the system.

Table III reports the execution times of the application when 1, 2, 4 and 6 computing nodes are used. The 6 clustering tasks that constitute the overall application are indicated as C2..C7, where the notation Cn refers to the task of grouping data into n clusters. The table shows how the clustering tasks are assigned to the computing nodes (denoted as N1..N6), as well as the partial execution times (file transfer time, data mining time, and WSRF overhead), and the total execution time.

The file transfer time includes the processes of compressing the dateset on the user node, transferring the compressed dataset, and decompressing the dataset on the computing node. As stated in the previous section, the WSRF overhead is given by the sum of the execution times needed to perform the various WSRF-related steps. Note that the case of 1 node corresponds to the sequential execution of tasks C2..C7 on a single machine. Thus, the file transfer time and the WSRF overhead in this case are not present, and the total time corresponds to the data mining time.

The total execution time decreases from 10882 s obtained using 1 computing node, to 2444 s obtained with 6 nodes. The achieved execution speedup ranged from 1.78 using 2 nodes, to 4.45 using 6 nodes. The execution times and speedup values for different number of nodes are represented in Figure 9.

[§]http://kdd.ics.uci.edu/databases/covertype/covertype.html [19 March 2007]

Table III. Task assignments and execution times for different number of nodes (times in seconds).

No of nodes	**Task assignments (Node ← Tasks)**	**File transfer**	**Data mining**	**WSRF overhead**	**Total time**
1	N1 ← C2,C3,C4,C5,C6,C7	0	10882	0	10882
2	N1 ← C2,C4,C6 N2 ← C3,C5,C7	199.6	5912	15.9	6128
4	N1 ← C2,C6 N2 ← C3,C7 N3 ← C4 N4 ← C5	193.6	3740	11.6	3945
6	N1 ← C2 N2 ← C3 N3 ← C4 N4 ← C5 N5 ← C6 N6 ← C7	197.6	2240	5.9	2444

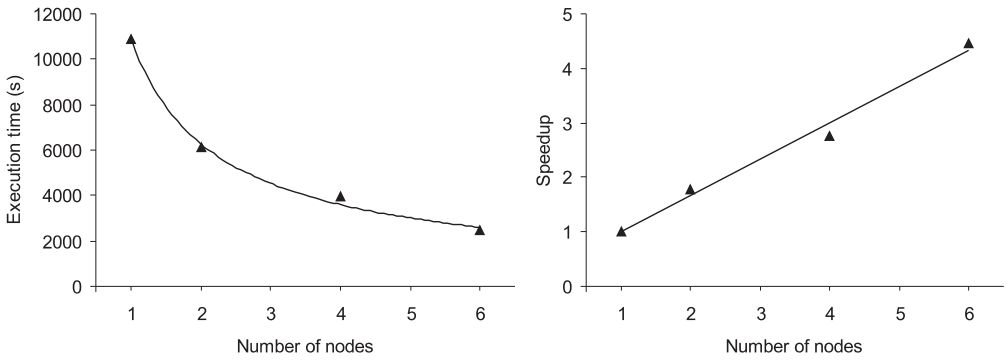

Figure 9. Execution times and speedup values for different number of nodes.

ONGOING WORK

As shown by the example described in the previous section, through the Weka4WS Explorer GUI a user can run applications composed of several data mining tasks that execute in parallel on single or multiple datasets. These data mining applications are effectively supported by the Explorer GUI, and their execution on multiple Grid nodes results in a significant increase of performance.

Besides the kind of applications discussed above, distributed data mining includes a variety of techniques that combine parallel and centralized execution of algorithms. An example of distributed data mining technique is *meta-learning*, which aims to build a global classifier from a set of inherently distributed data sources [18]. Meta-learning is basically a two-step process: first, a number of independent classifiers are generated by applying learning programs to a collection of distributed and homogeneous datasets in parallel. Then, the classifiers computed by learning programs are collected in a centralized site and combined to obtain the global classifier.

Another example of distributed data mining is *collective data mining* [19]. Instead of combining incomplete local models, collective data mining builds the global model through the identification of significant sets of local information. In other terms, the local blocks directly form the global model. This result is based on the fact that any function can be expressed in a distributed fashion using a set of appropriate basis functions. If the basis functions are orthogonal the local analysis generates results that can be correctly used as components of the global model.

The Weka4WS Explorer GUI does not support the visual design of such complex distributed data mining scenarios, because it is based on the Weka Explorer environment that is targeted to the execution of single data mining tasks. On the other hand, the Explorer is not the only way for implementing distributed data mining tasks in Weka4WS. In fact, the Weka4WS Web services can be directly invoked within ad hoc programs or scripts that exploits the Weka4WS client module. This allows developers to implement applications that coordinate the invocation of multiple data mining services in a distributed scenario more complex of that shown in the Application example section. Therefore, a distributed data mining application can be composed by several tasks that are submitted to different Weka4WS Web services in parallel and/or in sequence.

We are currently working to extend Weka4WS in order to support the visual design of distributed data mining applications that coordinate the execution of algorithms on different Grid nodes. To this end, we are extending the *KnowledgeFlow* environment provided by the original Weka. The KnowledgeFlow is an alternative to the Weka Explorer as a front end to the algorithms in the Weka library. Using the KnowledgeFlow a user can select the Weka components from a tool bar, place them onto a panel and connect them together in order to form a "knowledge flow" for processing and analyzing data. Currently, all the Weka classifiers and filters are available as components in the KnowledgeFlow. The most important feature of the KnowledgeFlow is that multiple batches of data can be processed in parallel, because each flow executes in its own thread.

Like in the Weka Explorer, all the components in the KnowledgeFlow are executed locally. We are extending the KnowledgeFlow in order to allow the execution of all the data mining

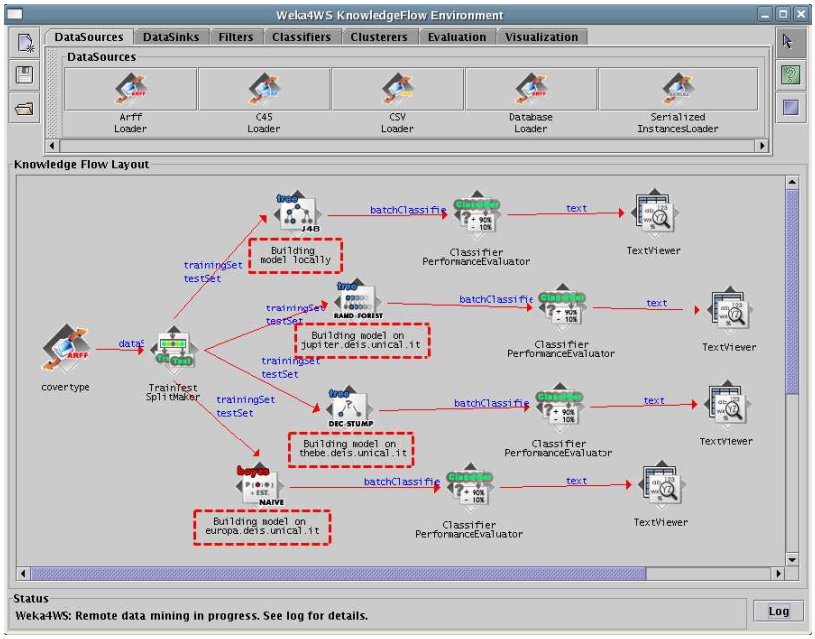

Figure 10. A distributed data mining application designed using the Weka4WS KnowledgeFlow environment: the broken boxes highlight the hosts where the single data mining tasks are running.

components in the "knowledge flow" on different Web services. In this way, the overall execution time can result significantly reduced because different parts of the computation are executed in parallel on different nodes, taking at the same time advantage of data and algorithms distribution.

Figure 10 shows a snapshot of the Weka4WS KnowledgeFlow prototype we are currently working on. As in the original KnowledgeFlow environment, the user can compose a "knowledge flow" as a workflow that links together data loaders, filters, data mining algorithms, and results visualizers. The new feature in our Grid-enabled KnowledgeFlow is that, for each algorithm in the application flow, the user can specify where such algorithm must be executed. The user can do that by right-clicking on the node that represents the algorithm, and then selecting the location of its execution through a control panel similar to that provided by the Weka4WS Explorer environment. As in the Explorer, the location can be set either to *Local*, *Auto*, or to a node specified by the user.

As an example, the "knowledge flow" in Figure 10 represents a distributed data mining applications in which a dataset (*covertype*) is analyzed by using four different classification algorithms (*J48*, *Random Forest*, *Decision Stump* and *Naive Bayes*). The flow starts (on the left) with an *ArffLoader* node, used to load the dataset, which is connected to a *TrainTest*

SplitMaker node that splits the dataset into training and test set. The *TrainTest SplitMaker* node is connected to four nodes, each one performing a given classification algorithm. These are in turn connected to a *ClassifierPerformanceEvaluator* node for the model validation, and then to a *TextViewer* node for results visualization. The user can see the results of the data mining tasks by clicking on the *TextViewer* nodes. When the application is started, the four branches of the "knowledge flow" are executed in parallel on four Grid nodes, selected as specified by the user. As highlighted in the figure, during the application execution the environment shows the names of the hosts on which the single data mining algorithms are running.

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 [20] extends the Weka toolkit to enable the use of multiple computational resources when performing data analysis. In that system, a set of data mining tasks can be distributed across several machines in an ad-hoc environment. Tasks that can be executed by 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.

Although 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 Grid Weka framework is not service-oriented, and it 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) [21] 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 [22] is used as the workflow system. FAEHIM exposes data mining services as Web services to enable 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, 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.

CONCLUSIONS

The Grid computing paradigm focuses on managing distributed applications and coordinated service composition involving networked computers and data sources rather than single resources. Today many data repositories are distributed for necessity and privacy reasons, therefore distributed infrastructures and applications can help in designing data management systems that access and analize data sources where they are or where it is necessary for functionality and/or performance purposes. To pursue this approach, distributed middleware is a key element and can help users in achieving their goals.

The middleware toolkit we discussed here offers a large set of data mining services for composing flexible knowledge discovery applications in distributed environments. Weka4WS adopts the emerging Web Services Resource Framework (WSRF) for remotely running data mining algorithms and composing distributed knowledge discovery applications that integrate data, tools, and resources available from dispersed sites through the SOA paradigm.

We described the design and the implementation of Weka4WS by exploiting the WSRF library provided by Globus Toolkit 4. To evaluate the efficiency of the implemented system, we also presented a performance analysis of Weka4WS that discussed the execution of two distributed data mining tasks in different network scenarios. This work is one of the first performance evaluations of WSRF. In particular, to the best of our knowledge, it proposes the first implementation and performance analysis of WSRF for data mining on the Grid.

The experimental results demonstrate the low overhead of the WSRF Web service invocation mechanisms with respect to the execution time of data mining algorithms and the efficiency of the WSRF framework as a means for executing data mining tasks on remote resources. By exploiting such mechanisms, Weka4WS provides an effective way to perform compute-intensive distributed data analysis on large-scale Grid environments.

The Weka4WS Web services can be directly invoked within ad hoc programs to implement applications that coordinate the invocation of multiple data mining services in a distributed scenario. Thus, a distributed data mining application can be composed by several tasks that execute on multiple Grid nodes in parallel and/or in sequence. We are currently working to extend the Weka KnowledgeFlow environment to support the visual design of distributed data mining applications. This will allow users to design and execute complex data mining applications on the Grid in a simple and effective way.

The Weka4WS code is available for research and application purposes. It can be downloaded from http://grid.deis.unical.it/weka4ws.

ACKNOWLEDGEMENTS

We would like to thank Marco Lackovic for his contribution to the system implementation. 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 also been 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 AM, Woehrer A. Towards an open service architecture for data mining on the grid. *Fourteenth International Workshop on Database and Expert Systems Applications* 2003.
3. Skillicorn D, Talia D. Mining large data sets on Grids: Issues and prospects. *Computing and Informatics* 2002; **21**(4):347–362.
4. Cannataro M, Talia D. The Knowledge Grid. *Communications of the ACM* 2003; **46**(1):89–93.
5. Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. Journal of Supercomputing Applications* 2001, **15**(1):200–222.
6. Witten H, Frank E. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, 2000.
7. Czajkowski K, et al. The WS-Resource Framework Version 1.0. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf [15 May 2006].
8. Talia D, Trunfio P, Verta O. Weka4WS: a WSRF-enabled Weka Toolkit for distributed data mining on Grids. *9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)* 2005; LNCS 3721, 309–320.
9. Foster I, Kesselman C, Nick J, Tuecke S. The Physiology of the Grid. In *Grid Computing: Making the Global Infrastructure a Reality*, Berman F, Fox G, Hey A (eds.). Wiley: New York, 2003; 217–249.
10. Graham S, et al. Publish-Subscribe Notification for Web services. http://www.oasis-open.org/committees/download.php/6661/WSNpubsub-1-0.pdf [26 April 2007].
11. Humphrey M, Wasson G, Kiryakov Y, Park S, Vecchio D, Beekwilder N. Alternative Software Stacks for OGSA-based Grids. *Supercomputing Conf.* 2005.
12. Alexander J, et al. Web Services Transfer (WS-Transfer). http://www.w3.org/Submission/2006/SUBM-WS-Transfer-20060927 [26 April 2007].
13. Box D, et al. Web Services Eventing (WS-Eventing). http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315 [26 April 2007].
14. Foster I. Globus Toolkit Version 4: Software for service-oriented systems. *Conf. on Network and Parallel Computing* 2005; LNCS 3779, 2–13.
15. Allcock W, Bresnahan J, Kettimuthu R, Link M, Dumitrescu C, Raicu I, Foster I. The Globus striped GridFTP framework and server. *Supercomputing Conf.* 2005.
16. Hettich S, Bay S D. The UCI KDD Archive, University of California, Department of Information and Computer Science. http://kdd.ics.uci.edu [19 March 2007].
17. MacQueen J B. Some Methods for classification and Analysis of Multivariate Observations, *5-th Berkeley Symposium on Mathematical Statistics and Probability* 1967; University of California Press, 281–297.
18. Prodromidis AL, Chan PK, Stolfo SJ. Meta-learning in distributed data mining systems: issues and approaches. In *Advances in Distributed and Parallel Knowledge Discovery*, Kargupta H, Chan P (eds.). AAAI/MIT Press: Menlo Park, 2000; 81-87.
19. Kargupta H, Park B, Hershberger D, Johnson E. Collective data mining: a new perspective toward distributed data mining. In *Advances in Distributed and Parallel Knowledge Discovery*, Kargupta H, Chan P (eds.). AAAI/MIT Press: Menlo Park, 2000; 133-184.
20. Khoussainov R, Zuo X, Kushmerick N. Grid-enabled Weka: A toolkit for machine learning on the Grid. *ERCIM News* 2004; **59**.
21. Shaikh Ali A, Rana OF, Taylor IJ. Web Services Composition for Distributed Data Mining. *Workshop on Web and Grid Services for Scientific Data Analysis* 2005.
22. The Triana Problem Solving Environment. http://www.trianacode.org [15 May 2006].