WSRF Services for Composing Distributed Data Mining Applications on Grids: Functionality and Performance

Domenico Talia, Paolo Trunfio, and Oreste Verta

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

Abstract. The Web Services Resource Framework (WSRF) has recently emerged as the standard for the implementation of Grid 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 Weka toolkit for supporting distributed data mining on WSRF-enabled Grids. Weka4WS adopts the WSRF technology for running remote data mining algorithms and managing distributed computations. The paper describes 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 two network scenarios is presented.

1 Introduction

In many scientific and business scenarios computational Grids are exploited to access distributed resources and run decentralized applications. Through a service-oriented model, application and system functions can be distributed among several computers or domains. 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. Since Grids emerged as effective infrastructures for distributed high-performance computing and data processing, a few Grid-based data mining systems have been proposed [1–4]. Our focus here is on distributed knowledge discovery services that allow decentralized teams or virtual organizations accessing and mining data in a high-level, standard and reliable way.

This paper describes *Weka4WS*, a framework that extends the widely-used Weka toolkit [5] for supporting distributed data mining in 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.

In Weka4WS the data mining algorithms for classification, clustering and association rules can be executed on remote Grid resources. To enable remote invocation, all data mining algorithms provided by the Weka library are 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 interoperability with standard Grid environments, Weka4WS has been developed by using the *Web Services Resource Framework* (*WSRF*) [6] as enabling technology. As WSRF implementations are recent, to the best of our knowledge, *Weka4WS* is the first data mining framework exploiting WSRF Grid services.

The paper describes the design, implementation and performance evaluation of Weka4WS for distributed data mining. To evaluate the overhead introduced by the service invocation mechanisms and their effects on the efficiency of the proposed system, a performance analysis of Weka4WS executing distributed data mining tasks in two 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. Finally, Section 4 concludes the paper.

2 Architecture and Implementation

The Web Services Resource Framework defines a family of technical specifications for accessing and managing *stateful resources* using Web Services. The composition of a Web Service and a stateful resource is termed *WS-Resource*. The Globus Alliance recently released the Globus Toolkit 4 (GT4) [7], 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 GT4.

In the Weka4WS framework all nodes use the GT4 services for standard Grid functionalities, such as security, data management, and so on. We distinguish those nodes 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 the 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. 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.

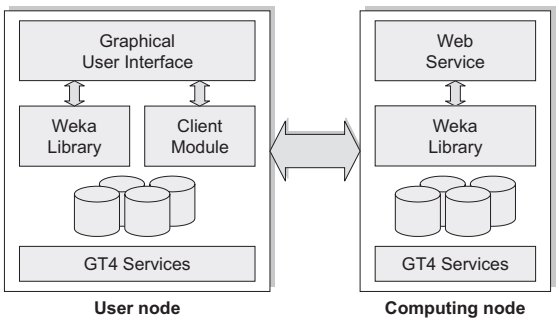

Fig. 1. Software components of user nodes and computing nodes.

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 available remote Grid nodes and two buttons to start and stop the data mining task on the selected Grid node. Through the GUI a user can either: *i*) start the execution locally by using the *Local* pane; *ii*) start the execution remotely by using the *Remote* pane. 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 all the data mining algorithms provided by the underlying WL. Therefore, requests to the WS are executed by invoking the corresponding WL algorithms.

2.1 Operations

Table 1 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).

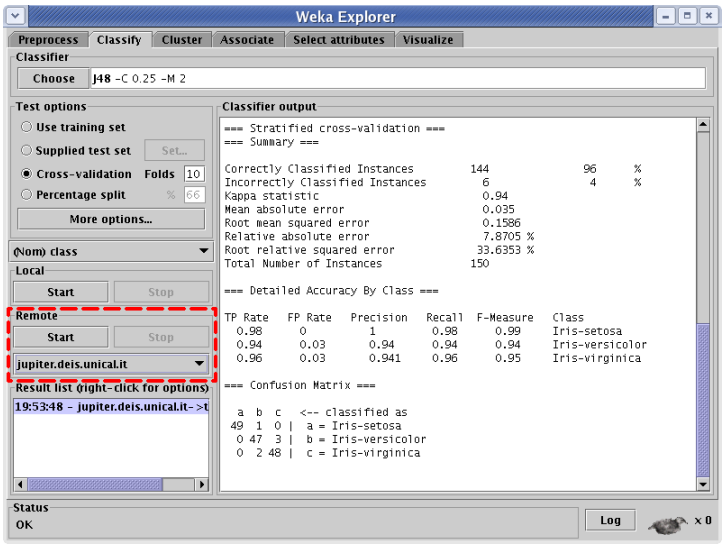

Fig. 2. The GUI: a Remote pane has been added to the original Weka Explorer.

Table 1. 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.

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

Table 2 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 be passed to the algorithm (e.g., "*-C 0.25 -M 2*"). The `dataSet` parameter specifies the URL of the dataset to be mined (e.g., "*gsiftp://hostname/path/ad.arff*"). Finally, the `crcValue` parameter specifies the checksum of the dataset to be mined.

2.2 Execution Mechanisms

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

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.
	`crcValue`	Checksum value 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.
	`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.

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. Subsequent requests from the CM will be directed to the resource identified by that EPR.
2. **Notification subscription**. The CM invokes the `subscribe` operation that subscribes to notifications about changes that will occur to the *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 classification task.
3. **Task submission**. The CM invokes the `classification` operation requiring the execution of the classification task. This operation receives six parameters as shown in Table 2, 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 [8] running on the user and computing nodes.
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**. Whenever 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 whenever they are generated.
7. **Resource destruction**. The CM invokes the `destroy` operation, which eliminates the resource created on Step 1.

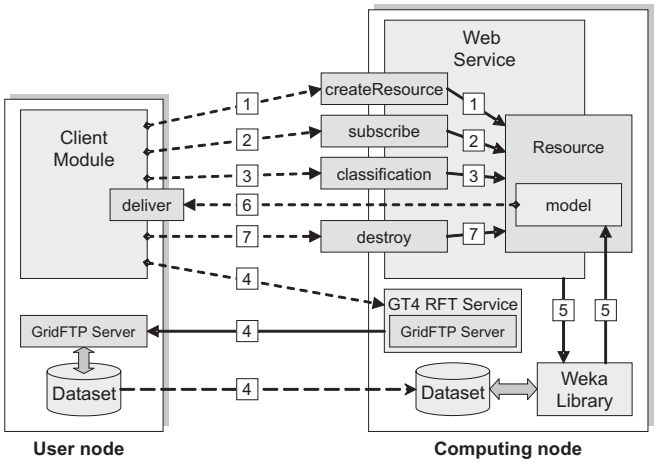

Fig. 3. Execution of a data mining task on a remote Web Service.

3 Performance Evaluation

To evaluate the performance of the implemented system, we carried out some experiments where we used Weka4WS for executing data mining tasks in different network scenarios. In particular, here we evaluate the execution times of the different steps needed to perform data classification tasks, 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 *ad* dataset from the UCI repository [9]. Through random sampling we extracted from it ten datasets, containing a number of instances ranging from 328 to 3280, with a size ranging from 1 to 10 MB. We used Weka4WS to perform a classification analysis on each of these datasets. In particular, we used the *J48* classification algorithm, using cross-validation with 10 folds. Other preliminary results obtained by running clustering tasks are discussed in [10] and are in accordance with the results discussed here.

The classification analysis on each dataset was executed 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 94.4 Mbps and an average round-trip time (RTT) of 1.4 ms.
- *Wide Area Grid* (*WAG*): the computing node and the user node are connected by a wide area network, with an average bandwidth of 213 Kbps and an average RTT of 19 ms.

Figure 4 shows the execution times of the different steps of the classification task in the LAG scenario for a dataset size ranging from 1 to 10 MB. As shown in the figure, the execution times of the WSRF-specific steps are independent from the dataset size, namely: *resource creation* (1732 ms, on the average), *notification subscription* (269 ms), *task submission* (302 ms), *results notification* (1348 ms), and *resource destruction* (194 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 240 ms for 1 MB to 1263 ms for 10 MB, while the *data mining* execution time ranges from 20311 ms for the dataset of 1 MB, to 1262603 ms for the dataset of 10 MB. The total execution time ranges from 24320 ms for the dataset of 1 MB, to 1267620 ms for the dataset of 10 MB.

Figure 5 shows the execution times of the different steps in the WAG scenario. The execution times of the WSRF-specific steps are similar to those measured in the LAG scenario. The only significant difference is the average execution time of the *results notification* step, which increases from 1348 ms in the LAG scenario to 2227 ms in the WAG scenario, due to additional time needed to transfer the classification 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 LAG scenario. In particular, the execution time of the *file transfer* step in the WAG scenario ranges from 27501 ms for 1 MB to 256021 ms for 10 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. Mainly due to the additional time required by the *file transfer* step, the total execution time is greater than the one measured in the LAG scenario, ranging from 52914 ms for the dataset of 1 MB to 1532678 ms for the dataset

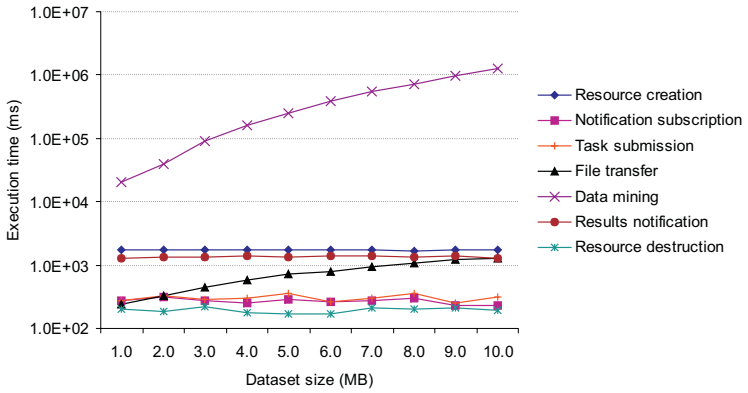

Fig. 4. Execution times of the different steps in the LAG scenario.

of 10 MB. Notice that the *data mining* line crosses the *file transfer* line in correspondence of the dataset of 3 MB. Under this size the *data mining* step is very fast, so the *file transfer* time is the dominant one, whereas for larger datasets the data mining time is larger than the data transfer overhead.

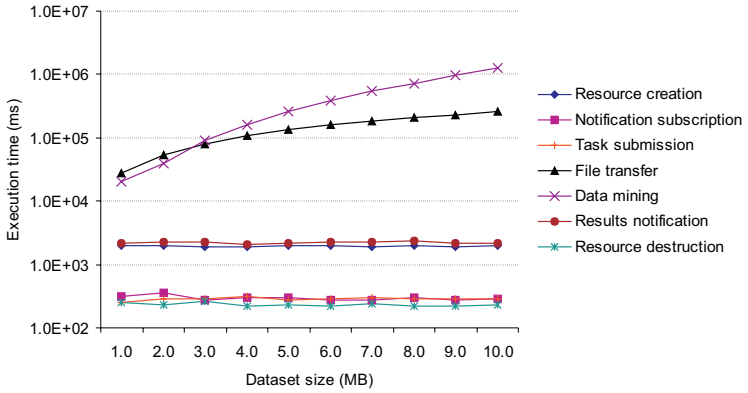

Fig. 5. Execution times of the different steps in the WAG scenario.

To better highlight the overhead introduced by the WSRF mechanisms and the distributed scenario, Figure 6 shows the percentage of execution times of the *data mining*, *file transfer*, and the other steps with respect to the total execution time in the LAG and WAG scenarios. For space reasons, the values reported in this figure refer only to the datasets of 2, 6, 4, 8 and 10 MB.

In the LAG scenario the *data mining* step takes the 90.40% of the total execution time for the dataset of 2 MB, whereas it takes the 99.60% of the total execution time for the dataset of 10 MB. At the same time, the *file transfer* ranges from 0.75% to 0.10%, and the other steps range from 8.85% to 0.30%. In the WAG scenario the *data mining* step takes from 40.64% to 82.97% of the total execution time, the *file transfer* ranges from 54.21% to 16.70%, while the other steps range from 5.15% to 0.32%.

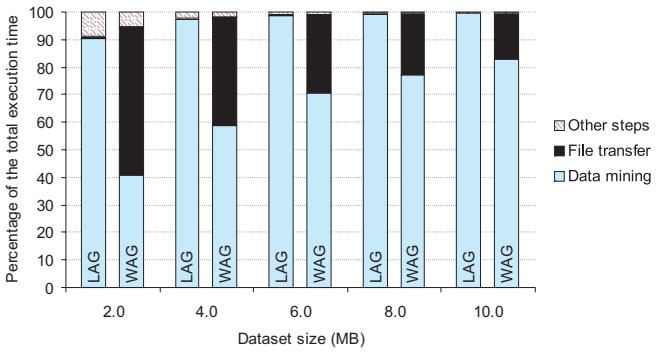

Fig. 6. Percentage of the execution times of the different steps in the LAG and WAG scenarios.

We can observe that in the LAG scenario neither the *file transfer* nor the other steps represent a significant overhead with respect to the total execution time. In the WAG scenario the *file transfer* is a critical step only on small datasets, since in this case 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 resources. By exploiting such mechanisms, Weka4WS can provide an effective way to perform compute-intensive data analysis in Grid environments.

4 Conclusions

Today many data repositories are distributed for necessity and privacy reasons, therefore distributed infrastructures 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 toolkit described in this paper offers a large set of data mining services for executing knowledge discovery applications in distributed environments. Weka4WS adopts the emerging Web Services Resource Framework (WSRF) for composing knowledge discovery applications that integrate data, algorithms, and resources available from dispersed sites.

The experimental results demonstrate the low overhead of the WSRF invocation mechanisms with respect to the execution time of data mining algorithms, and the efficiency of WSRF as a means for executing data mining tasks on remote resources. By exploiting such mechanisms, Weka4WS can provide an effective way to perform compute-intensive data analysis on Grids. The Weka4WS code is available for research and application purposes. It can be downloaded from http://grid.deis.unical.it/weka4ws.

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. 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).
2. Cannataro, M., Talia, D.: The Knowledge Grid. CACM, vol. 46 n. 1 (2003) 89-93.
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. Curcin, V., Ghanem, M., Guo, Y., Kohler, M., Rowe, A., Syed, J., Wendel, P.: Discovery Net: Towards a Grid of Knowledge Discovery. Conf. on Knowledge Discovery and Data Mining (2002).
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.: Globus Toolkit Version 4: Software for Service-Oriented Systems. Conf. on Network and Parallel Computing, LNCS 3779 (2005) 2-13.
8. Allcock, B., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., Foster, I.: The Globus Striped GridFTP Framework and Server. Conf. on Supercomputing (2005).
9. The UCI Machine Learning Repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
10. Talia, D., Trunfio, P., Verta, O.: Weka4WS: a WSRF-enabled Weka Toolkit for Distributed Data Mining on Grids. Conf. on Principles and Practice of Knowledge Discovery in Databases, LNCS 3721 (2005) 309-320.