CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2012; **24**:1751–1774
Published online 17 October 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1877

Distributed data mining patterns and services: an architecture and experiments

Eugenio Cesario [1,*,†] and Domenico Talia [1,2]

[1]*ICAR-CNR, Via P. Bucci 41C, 87036 Rende (CS), Italy*
[2]*DEIS-University of Calabria, Via P. Bucci 41C, 87036 Rende (CS), Italy*

SUMMARY

Distributed data mining implements techniques for analyzing data on distributed computing systems by exploiting data distribution and parallel algorithms. The grid is a computing infrastructure for implementing distributed high-performance applications and solving complex problems, offering effective support to the implementation and use of data mining and knowledge discovery systems. The Web Services Resource Framework has become the standard for the implementation of grid services and applications, and it can be exploited for developing high-level services for distributed data mining applications. This paper describes how distributed data mining patterns, such as collective learning, ensemble learning, and meta-learning models, can be implemented as Web Services Resource Framework mining services by exploiting the grid infrastructure. The goal of this work was to design a distributed architectural model that can be exploited for different distributed mining patterns deployed as grid services for the analysis of dispersed data sources. In order to validate such an approach, we presented also the implementation of two clustering algorithms on the developed architecture. In particular, the distributed k-means and distributed expectation maximization were exploited as pilot examples to show the suitability of the implemented service-oriented framework. An extensive evaluation of its performance was provided. Copyright © 2011 John Wiley & Sons, Ltd.

Received 14 October 2009; Revised 7 March 2011; Accepted 28 August 2011

KEY WORDS: grid computing; distributed data mining; OGSA; WSRF

1. INTRODUCTION

1.1. Reference context

Because of advances in information technology availability and use, digital data volumes are growing exponentially in many fields of human activities. This trend concerns scientific disciplines, as well as industry and commerce. Moreover, with the availability of inexpensive storage and the progress in data capture technology, many organizations have created ultra-large databases of business and scientific data, and this trend is largely and quickly growing. Examples of such data sources include gene and DNA databases, astronomy and meteorology data repositories, network access, and data about web usage, content, and structure.

An important part of *data mining* research [1] has been historically focused on centralized data mining, where the data set to be analyzed is stored at a single site. Indeed, even though this approach is absolutely valid and is making a lot of progress and giving many important results, *distributed data mining (DDM)* is a fast growing technology that solves the problem of finding data patterns in scenarios where data and computation are distributed. DDM was originated by two main reasons: the first one is related to the computational cost of processing large data stores, the second is concerned with the geographical location of data repositories [2]. Concerning the first issue, under

*Correspondence to: Eugenio Cesario, ICAR-CNR, Via P. Bucci 41C, 87036 Rende (CS), Italy.
†E-mail: cesario@icar.cnr.it

the perspective of analyzing huge data repositories (as we have aforementioned), mining large data sets requires powerful computational resources. This is becoming a more and more important need, especially considering the very large size of current and next-generation databases. A generic algorithm on a single machine could take a very long processing time on current realistic volumes of data to get results. The second issue is related to the fact that in some applications, data are inherently distributed among many sites, but it is necessary to gain global knowledge from the distributed data partitions. For example, several sites of a multinational company manage their own operational data locally, but all data must be analyzed for global patterns to allow company-wide activities (such as planning, marketing, and sales). Moreover, sharing the data to a central site could be too expensive (because of excessively long time of transfer) or infeasible (for privacy or security issues).

Distributed data mining [2–7] analyzes data in a distributed way and pays particular attention to the trade-off between centralized collection and distributed analysis of data. Knowledge discovery is speeded up by executing concurrently a number of data mining tasks on different data subsets and then combining the results at a centralized site. However, DDM does not come without overhead. Some DDM algorithms suffer from excessive communication and data dependencies and will not scale well. For such a reason, DDM is not the most suitable approach in all the cases, even though it is becoming more and more feasible with the advances in networking and communication technology (especially the growth of Internet and intranets).

The *grid* is a computing infrastructure for implementing distributed high-performance applications and solving complex problems [8]. *Grid computing* systems received a lot of attention both from the research community and from industry and governments. The driving grid applications are traditional high-performance applications, such as high-energy particle physics, and astronomy and environmental modeling, in which experimental devices create large quantities of data that require scientific analysis. Grid computing differs from conventional distributed computing because it focuses on large-scale resource sharing, it offers innovative applications, and, in some cases, it is geared toward high-performance systems. For these reasons, grids offer effective support to the implementation and use of DDM algorithms and knowledge discovery systems. To achieve such a goal, DDM frameworks and applications exploiting the grid infrastructure have been designed and implemented.

1.2. Motivations and contribution

Typically, DDM data sets are stored in local databases or file systems hosted by local computers/repositories that are connected in a computer network. Data mining takes place both locally at each remote site and at a global level where the local knowledge is fused in order to discover global knowledge. One of the most common DDM patterns is reported in Figure 1. The first phase normally involves the analysis of the local datasets at each site; this step infers statistics or local models. In the second step, the locally discovered knowledge is usually transmitted to a merger (or central) site, where the integration/refinement of the distributed local models is performed. To achieve such goal, in some cases, the dataset (or part of it) is replicated also in the central node. The merger site evaluates if the algorithm is terminated or not. If it is the case, the merger site produces the global mining model that could be returned to the user. Otherwise, the merger site requests further elaboration to the local sites (and in some cases, the temporary global model is broadcasted to all the local sites) and, so on, until the merger site obtains the convergence of the algorithm. In some approaches, at the end of the computation, the local models are finally broadcasted to all other sites, so that each site can compute the global model in parallel.

It is worth noting that such a pattern is very common in many classes of DDM algorithms. Examples of solutions adhering to this design pattern fall into the following categories: clustering [3, 9–12], classification [13], association rule and frequent itemsets mining [13–16], ensemble learning [1], collective data mining [3], and meta-learning [6].

The aforementioned schema is not the only pattern used in DDM, but it is common to several DDM algorithms. An example of DDM technique is *meta-learning*, which aims at building a global classifier from a set of inherently distributed data sources [6]. Meta-learning is basically a two-step process: first, a number of independent classifiers are generated by applying learning programs to

DOI: 10.1002/cpe

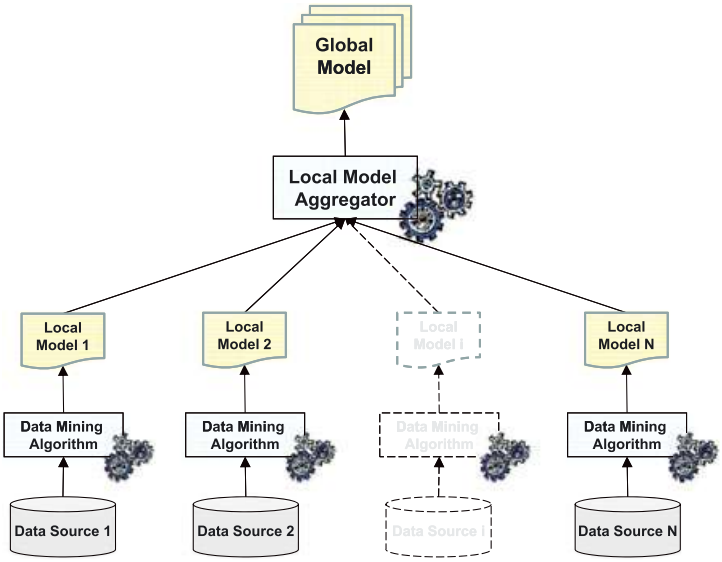


Figure 1. Typical architecture of a distributed data mining algorithm.

a collection of distributed and homogeneous data sets in parallel. Then, the classifiers computed by local learning programs are collected in a single site and combined to obtain a global classifier. Another example of DDM is the *collective data mining* [7]: instead of combining partial local models, collective data mining builds the global model through the identification of significant sets of local information. In other words, the local blocks are directly composed to form the global model. This result is based on the fact that any mining 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. Another DDM model is the *ensemble learning* [1], a well-known technique for improving classification accuracy by aggregating predictions of multiple classifiers. An ensemble method constructs a set of base classifiers from training data and performs classification by voting (in the case of classification) or by averaging (in the case of regression) on the predictions made by each classifier. The final result is the ensemble classifier that generally has higher classification quality than that of any single classifier composing it.

The main contribution of this paper is the design and implementation of a framework for developing and executing DDM algorithms on a grid infrastructure. In particular, one main motivation of the work is to simplify the reuse of data mining patterns, differently from generic tools that often require a user to program ex-novo data mining applications. To do that, we developed a service-oriented system to implement prearranged algorithmic patterns, adhering to the schema described in Section 1.2. In particular, if a user needs to exploit a master/worker schema, the framework provides a specific prearranged pattern; therefore, the user is not requested to design an ad hoc structure (i.e., workflow) modeling the application to be executed. We firstly present the design of such a distributed architecture, in which mining algorithms deployed as grid services can run on. Such services, defined as Open Grid Services Architecture (OGSA)-compliant, have been developed by exploiting the Web Services Resource Framework (WSRF) specifications. Technical details about the implementation of the framework and low-level specifications on its usage will be provided. Then, the suitability of the provided solution is discussed by two clustering task examples, chosen as pilot case studies. Moreover, it is pointed out that any process adhering to the schema shown in Figure 1 can be easily integrated in our framework. Finally, we describe an extensive experimental evaluation of the system on a real grid, aimed at evaluating both its efficiency and effectiveness. In addition, we provide a quantitative comparison with other grid-based data mining frameworks found in literature.

DOI: 10.1002/cpe

1.3. Plan of the paper

The rest of the paper is organized as follows. Section 2 introduces grid service technologies and presents how these are used in the design of the architectural model that we developed. Section 3 describes the implementation of two clustering algorithms in such service-oriented grid data mining architecture, the k-means and expectation maximization (EM) algorithms. Section 4 discusses the experimental evaluation of the implemented algorithms. Section 5 describes other data mining systems exploiting the grid infrastructure and analyzes the main differences with the developed framework. Section 6 gives some concluding remarks.

The work described in this paper is an extended version of the work presented in [17]. 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.

2. DESCRIPTION OF THE FRAMEWORK

Based on the algorithmic schema described in Section 1.2, here we present the service-oriented architecture (SOA) we designed for the execution of DDM patterns on grids. To describe in detail our proposal, we give a brief description of what a service-oriented architecture is and its impact on grid computing.

2.1. Service-Oriented Architecture, Open Grid Services Architecture, and Web Services Resource Framework

The SOA is a model for building flexible, modular, and interoperable software applications. The key aspect of SOA is the concept of *service*, a software block capable of performing a given task or business function.

The most important implementation of SOA is represented by *Web Services*, whose popularity is mainly caused by the adoption of universally accepted technologies such as XML, SOAP, and HTTP. Also, the grid provides an SOA framework whereby a great number of services can be dynamically located, balanced, and managed, so that applications are always guaranteed to be securely executed according to the principles of on-demand computing. The grid community has adopted the OGSA as an implementation of the SOA model within the grid context. In OGSA, every resource is represented as a Web service that conforms to a set of conventions and supports standard interfaces. OGSA provides a well-defined set of Web service interfaces for the development of interoperable grid systems and applications [8].

Recently, the WSRF has been defined as an evolution of early OGSA implementations [18]. WSRF defines a family of technical specifications for accessing and managing stateful resources using Web services, as required by OGSA. In other words, WSRF depicts some specifications to implement OGSA-compliant Web services. Another way of expressing this relation is that, whereas OGSA is the architecture, WSRF is the infrastructure on which that architecture is built on [19]. The composition of a Web service and a stateful resource is termed as *WS-resource*. The possibility to define a state associated to a service is the most important difference between WSRF-compliant Web services and pre-WSRF ones. This is a key feature in designing grid applications, because WS-resources provide a way to represent, advertise, and access properties related to both computational resources and applications. In order to implement services in a highly decentralized way, it commonly used a design pattern that allows a client to be notified when interesting events happen in a server. The *WS-notification* specification defines a *publish/subscribe* notification model for Web services, which is exploited to notify interested clients and/or services about changes that occur to the status of a WS-resource. In other words, a service can publish a set of topics that a generic client can subscribe to; as soon as the topic changes, the client receives a notification of the new status.

2.2. Architecture of the proposed framework

In order to provide a simple approach for the specification and execution on the grid of DDM algorithms designed according to the master/worker pattern, we have designed the grid service

DOI: 10.1002/cpe

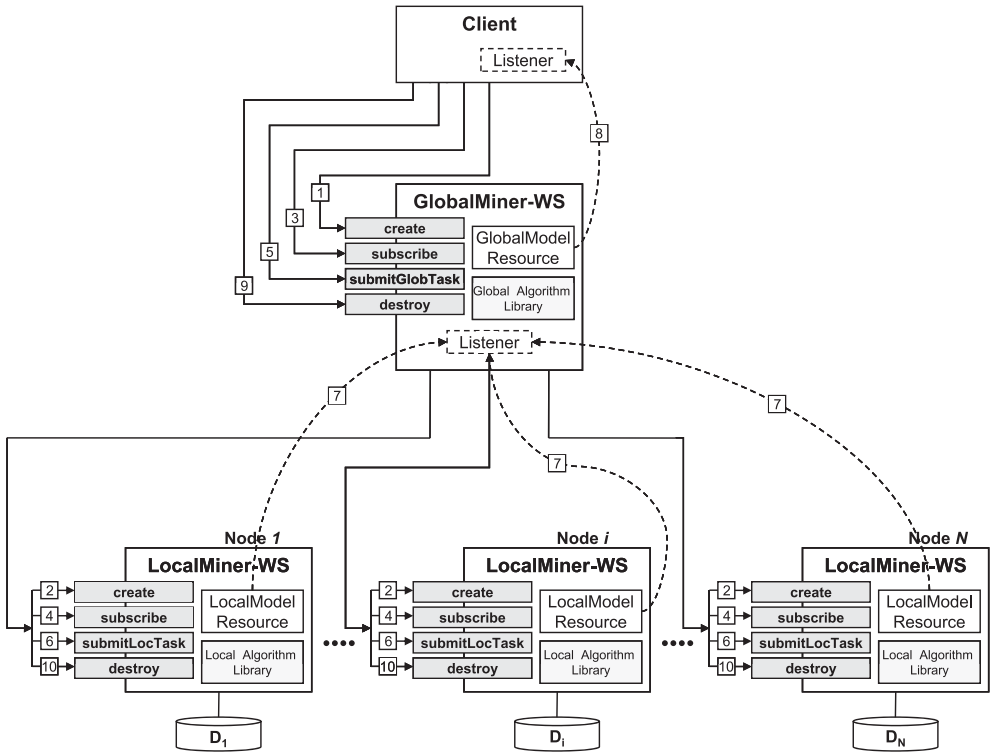


Figure 2. Architectural model of distributed data mining services.

architectural model shown in Figure 2. It is composed of two grid services: the *GlobalMiner-WS* and the *LocalMiner-WS*. The overall architecture resembles a typical DDM schema contemplating the presence of an entity acting as coordinator (the *GlobalMiner-WS*) and a certain number of entities acting as miners (*LocalMiner-WS*) on local sites. Thus, it is a master/worker architecture, in which the service on the master node arranges the operations performed by the services on the worker nodes. A resource is associated to each service: the *GlobalModel Resource* to the *GlobalMiner-WS* and the *LocalModel Resource* to the *LocalMiner-WS*. Such resources are used to store the state of the services, in this case, represented by the computed models (globally and locally, respectively). Additionally, the resources are published also as *topics*, in order to be considered as "items of interest of subscription" for *notifications*. As it can be evicted from Figure 2, a *global algorithm library (GAL)* and a *local algorithm library (LAL)* are present in the two types of nodes; they are code libraries providing the mining algorithms to be executed (at the global and local level, respectively). In the following, we describe all the steps composing the whole process, by pointing out details of the interactions between entities composing the whole architecture. Let us suppose that a client wants to execute a distributed mining algorithm on a dataset D, which is split in N partitions, $\{D_1, \ldots, D_N\}$, each one stored on one of the nodes $\{Node_1, \ldots, Node_N\}$. A request to the framework of performing a mining process can be labeled in three different main phases, each one composed of various steps, as described in the following:

Phase 1 - Resource creation and notification subscription. Whenever a client wants to submit a request, it firstly invokes the `createResource` operation of the *GlobalMiner-WS* to create the *GlobalModel Resource* (step 1). In turn, the *GlobalMiner-WS* dispatches this operation in a similar way: for each local site with a running *LocalMiner-WS* (supposing that the *GlobalMiner-WS* holds a list of them), it invokes the `createResource` operation (of the *LocalMiner-WS*) to create the *LocalModel Resource* (step 2). At this point, the client invokes the `subscribe` method to subscribe a listener (inside the client) as consumer of notifications on the *GlobalModel* topic (step 3). Finally, the *GlobalMiner-WS* invokes the `subscribe` method to subscribe a listener as consumer

DOI: 10.1002/cpe

of notifications on the *LocalModel* topic (step 4). As an effect of such subscription steps, as soon as a resource changes, its new value will be delivered to its listener.

Phase 2 - Task submission and results notification. Such a phase is the core of the application, that is, the execution of the mining process, and the result returns. The client invokes the `submitGlobalTask` operation (step 5), by passing a *GlobalTaskDescriptor*, that is, a complete description of the task to be executed (algorithm name, parameters, initialization type, etc.). At this stage, through an interaction with the *GAL*, the *GlobalMiner-WS* runs the appropriate code executing the steps to be done. During this phase, suitable data structures are initialized, and a suitable set of *LocalTaskDescriptor* are created. It invokes the `submitLocalTask` operation (step 6), by passing a LocalTaskDescriptor. At this stage, each i^{th} *LocalMiner-WS* begins the analysis of the local dataset for inferring a local model and local statistics. Such task is executed with the support of the *LAL* and is (obviously) executed concurrently on all the grid sites. While every local task is in progress, the *GlobalMiner-WS* does not need to periodically poll if they are terminated: the notification mechanism foresees an asynchronous delivery of a message when a particular event occurs (in our case, the termination of local tasks). As soon as a local computation terminates, the value of the *LocalModel Resource* is set by the value of the inferred local model. The changes in this resource are automatically notified to the listener on the *GlobalMiner-WS* (step 7); in this way, the local model computed at the i^{th} local site is delivered to the global site. As soon as all the local models are delivered to the *GlobalMiner-WS*, the integration of all these local models is performed. According to the logic of the particular algorithm (provided by the *GAL*), the *GlobalMiner-WS* evaluates if the algorithm is terminated (or not). If it is, the global model computed by the *GlobalMiner-WS* is stored on the *GlobalModel Resource*, that is immediately delivered (for the notification mechanism) to the client (step 8). Otherwise, the *GlobalMiner-WS* asks for further processing, by invoking one more time the `submitLocalTask` operation (step 6) and waiting for the delivering of the result (step 7). Such couple of steps (6 and 7) are executed, as many times as the *GlobalMiner-WS* needs, until the computation achieves a convergence condition.

Phase 3 - Resource destruction. As final steps, as soon as the computation terminates and its results have been notified to the client (step 8), the client invokes the `destroy` operation of the *GlobalMiner-WS* (step 9), which eliminates the resource previously created (*GlobalModel*). Similarly, the *GlobalMiner-WS* asks for the destruction of the *LocalModel* (step 10).

2.3. Some details on "submitGlobalTask" and "submitLocalTask" operations

Figure 3 shows the schema of the `submitGlobalTask` operation. The procedure receives a *globalTaskDescriptor* instance as parameter, that is, a description of the task to be executed (algorithm name, input parameters, initialization choice, etc.). Initially, by an interaction with the *GAL*, the appropriate *globalAlgorithm* instance is loaded (line 5). Then, the chosen algorithm performs an initialization step (line 6), consisting in the preparation of the local computation descriptions and global model initialization (if it is needed). In the main loop (lines 7–14), the local tasks are submitted to all local services (lines 8–10) and, as soon as their results have been notified, the *globalModel* is updated (line 13). The loop is repeated until the algorithm stops because the obtained global model is stable or it fits the convergence requirements specified in the input parameters (line 14). Some finalization operations (line 15) and the setting of the *GlobalModel Resource* by its correct value (line 16) complete the procedure. A similar, but simpler, structure is adopted for the `submitLocalTask` (whose schema is reported in Figure 4). Initially, the appropriate *localAlgorithm* is loaded from the *LAL* (line 3). After an initialization step (line 4), the local computation is executed (step 5). The procedure terminates with a finalization step (line 6).

We observe that if a new X-algorithm has to be added to the library, it is sufficient to implement suitable `X-globalAlgorithm` and `X-localAlgorithm` instances providing the methods used in the procedures: *initialize*, *computeGlobalModel*, *needsMoreIteration*, and *finalize* for the *GlobalAlgorithm*; *initialize*, *computeLocalModel*, and *finalize* for the *localAlgorithm*. A brief explanation of such methods is reported in Figure 5. The new algorithm instances must be added to the code libraries, so that can be retrieved for further computations. More implementation details are reported in section 4.3.

DOI: 10.1002/cpe

```
SUBMITGLOBALTASK(TaskDescriptor globalTaskDescriptor)
 1: GlobalModel globalModel;
 2: LocalModel localModels[];
 3: TaskDescriptor localTaskDescriptors[];
 4:
 5: Algorithm globalAlgorithm = GAL.getAlgorithm(globalTaskDescriptor);
 6: (localTaskDescriptors[], globalModel) = algorithm.initialize(globalTaskDescriptor);
 7: repeat
 8:   for each localService ls[i] do
 9:     ls[i].SUBMITLOCALTASK(localTaskDescriptors[i]);
10:   end for
11:   WAIT();
12:   ...The process is waked up after the asynchronous delivery of the local results in localModels[]
13:   globalModel = algorithm.computeGlobalModel(localModels[]);
14: until not algorithm.needsMoreIteration(globalModel, globalTaskDescriptor)
15: algorithm.finalize();
16: set GlobalModelResource = globalModel;
```

Figure 3. The submitGlobalTask operation.

```
SUBMITLOCALTASK(TaskDescriptor localTaskDescriptor)
1: LocalModel localModel;
2:
3: Algorithm localAlgorithm = LAL.getAlgorithm(localTaskDescriptor);
4: localModel = algorithm.initialize(localTaskDescriptor);
5: localModel = algorithm.computeLocalModel(localTaskDescriptor,localModel);
6: algorithm.finalize();
7: set LocalModelResource = localModel;
```

Figure 4. The submitLocalTask operation.

Method	Description
INITIALIZE	Initialization of the global model, variables, etc.
COMPUTEGLOBALMODEL	Compute the global model by integration of the local models
NEEDSMOREITERATION	Evaluates if the computation is terminated or not, w.r.t. convergence condition
FINALIZE	Finalization steps

(a) GlobalAlgorithm interface.

Method	Description
INITIALIZE	Initialization steps
COMPUTELOCALMODEL	Compute the local model, by processing the available data
FINALIZE	Finalization steps

(b) LocalAlgorithm interface.

Figure 5. Interface of the (a) GlobalAlgorithm and (b) LocalAlgorithm.

Let us do two final considerations on the `submitGlobalTask` operation. The first one is that the number of iterations performed in the main cycle (Figure 3, lines 7–14) cannot be generally fixed a priori. In fact, it depends on the particular mining application submitted to the framework and may also depend on the data. Some algorithms require the execution of a certain number of iterations until a convergence condition has been reached [11, 12]. Other algorithms perform the execution of exactly one [3, 6, 20] or two ([10]) iterations. The exit-loop condition is implemented in the *needsMoreIteration* method, and each algorithm exploiting the framework has to provide it. The second consideration is about the *computeGlobalModel* method (Figure 3, line 13). Its logic and complexity is strictly dependant on the particular technique used. In some cases, the *global model* is just a composition of the local models computed on the remote site (e.g., collective data mining). In other cases, the *global model* is a new mining model obtained by aggregating (voting, regression) predictions of the local models (e.g., ensemble learning). In other cases, the *global model* is

DOI: 10.1002/cpe

obtained by processing the statistics composing the local models. The `computeGlobalModel` method, provided by each algorithm, implements its particular aggregate/merge strategy.

3. TWO APPLICATIVE SCENARIOS: DISTRIBUTED K-MEANS AND DISTRIBUTED EXPECTATION MAXIMIZATION

In order to show how to apply the proposed architecture for implementing service-oriented DDM patterns, we present two examples of distributed clustering algorithms exploiting the architecture described in Section 2.

Clustering is one of the main techniques in scientific data analysis. It is an unsupervised classification technique that aims at grouping a set of unlabeled objects into meaningful clusters, with the requirement that the resulting groups are homogeneous (i.e., pairs of objects in the same cluster are highly similar), and separated (i.e., objects in distinct clusters are dissimilar). Several distributed clustering algorithms have designed [10–12, 21]. Among them, we have selected two well-known algorithms, the *distributed k-means* and *distributed expectation maximization*, as pilot examples to show the suitability of the proposed service-oriented framework. Note that our contribution is in the service-oriented infrastructure and not in new distributed mining algorithms.

3.1. Distributed k-means

K-means is a well-known partitional clustering algorithm and one of the simplest unsupervised learning techniques. It receives the number of clusters K as input and proceeds as follows: First, it randomly selects K of the objects in the dataset, each one representing a cluster mean or center (also called representative). The representative of a cluster C is an object containing, for each attribute, the mean values of the data objects in C. For each of the other objects in the dataset, an object is compared with the K representatives and is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster representative (usually, the Euclidian distance is used). Then, it computes the new representative for each cluster. This process iterates until the criterion function $Perf_{KM}$ converges, that is, cluster assignments are stable.

The *distributed k-means algorithm* has been one of the earliest distributed mining algorithms [11, 12]. Exploiting the classical DDM algorithm structure represented in Figure 1, it includes a coordinator site and a certain number (i.e., N) of local sites, where the dataset D is distributed $(D_1, \ldots, D_N)$. The key idea consists of asking the local nodes to compute sufficient statistics (or local models) in parallel that are suitably summarized to the coordinator site.

We can observe that the distributed k-means algorithm fits very well our service-oriented architecture, where the GlobalMiner-WS works as coordinator and the LocalMiner-WSs are miners on local sites. The integration of the k-means in the framework is performed by the implementation of the `k-means-GlobalAlgorithm` and `k-means-LocalAlgorithm` instances (implementing the interfaces reported in Figure 5), as described in the previous section. For completeness, the distributed k-means algorithm [11] is reported in Figure 6. To better clarify the way such an algorithm has been integrated in the proposed framework, some steps are labeled by the name of the methods (among those reported in Figure 5) implementing them.

3.2. Distributed expectation maximization

Expectation maximization is a well-known clustering algorithm belonging to the model-based category. Such methods, based on the assumption that the data to be clustered are generated by one of several distributions (i.e., Gaussian), attempt to optimize the fit between the given data and a mathematical model. A *mixture model* is a set of K probability distributions, representing K clusters that govern the attribute values for members of that cluster.

Given the input dataset and a prespecified number K of clusters, EM learns, for each instance, the membership probability of each cluster and, for each cluster, its descriptive model (i.e., the parameters that govern its generative process). In the EM framework, neither the parameters of the distribution that each instance comes from nor the parameter of the mixture models are known. Both

DOI: 10.1002/cpe

Scenario: a coordinator node and N local nodes, where local datasets $\{D_1, \ldots, D_N\}$ are stored

INITIALIZE:

1. The coordinator initializes randomly the K centroids, $\mu_1, \ldots, \mu_K$

2. The coordinator sends a copy of the centroids to each local site

3. Each i^{th}local site independently executes the following steps:

COMPUTELOCALMODEL:

3.1 for each data object $x \in D_i$, assigns it to the closest centroid w.r.t. the euclidian distance

3.2 for each cluster $C_k^{(i)}$, $k = 1, \ldots, K$, computes the local statistics (or local model) $SS_k^{(i)} = \{n_k^{(i)}, \Sigma_k^{(i)}, s_k^{(i)}\}$, where
$n_k^{(i)} = |C_k^{(i)}|, \quad \Sigma_k^{(i)} = \sum_{x \in C_k^{(i)}} x, \quad s_k^{(i)} = \sum_{x \in C_k^{(i)}} dist(x, \mu_k)^2$

3.3 collects the $SS^{(i)} = \{n_k^{(i)}, \Sigma_k^{(i)}, s_k^{(i)}\}, k = 1, \ldots, K$

3.4 it sends the local statistics to the coordinator

COMPUTEGLOBALMODEL: **4.** The coordinator adds up all the $SS^{(i)}$ received from each i^{th}local site, $i = 1, \ldots, N$, to get the global $SS = \{n_k, \Sigma_k, s_k\}, k = 1, \ldots, K$, by the following formula:
$n_k = \sum_{i=1}^{N} n_k^{(i)}, \quad \Sigma_k = \sum_{i=1}^{N} \Sigma_k^{(i)}, \quad s_k = \sum_{i=1}^{N} s_k^{(i)}$.
Based on them, it computes the new centroids $\{\mu_1, \ldots, \mu_K\}$, and updates the performance function $Perf_{KM}$:
$\mu_k = \frac{\Sigma_k}{n_k}, \quad Perf_{KM} = \sum_{k=1}^{K} s_k$

NEEDSMOREITERATION: **5.** If the performance function $Perf_{KM}$ converges or after a fixed number of iterations, the algorithm terminates; else, go to the step 2.

Figure 6. Distributed k-means algorithm.

are estimated from data. The classic EM algorithm requires initial random mixture model parameters as input. Given such parameters, a single EM iteration provides new parameter estimates, which are proven not to decrease the likelihood of the model, $Perf_{EM}$. The process is repeated until convergence, that is, the $Perf_{EM}$ at the previous iteration is sufficiently close to the likelihood of the current model.

The *distributed EM algorithm* has been presented in various references [11, 22]. The integration of the EM in the framework is performed by the implementation of the `EM-GlobalAlgorithm` and `EM-LocalAlgorithm` instances. The distributed EM algorithm [11] is reported in Figure 7. As previously performed, some labels are used to specify the methods (among those reported in Figure 5) implementing the various steps of the algorithm.

4. EXPERIMENTAL EVALUATION

To evaluate the execution time and scalability of the architectural model, we carried out a performance analysis by executing different experiments in various scenarios. We have implemented the distributed k-means and the distributed EM as composition of grid services according to the architectural model described in Section 2 and the algorithms presented in Section 3. The services development has been performed by using the Java WSRF library provided by the WS-Core, a component of the Globus Toolkit 4 [23]. The services implementing such algorithms have been deployed in a real grid composed of 18 nodes. For both algorithms, we have deployed the *GlobalMiner-WS* (working as coordinator) on a single node and the *LocalMiner-WS* on each of the other 16 nodes. The experiments were run on Intel Pentium 4 processors, CPU frequency 1.36 GHz and with 2-GB RAM.

We have performed the experiments in a scenario where a user wants to submit a clustering task to the framework on the dataset D. To do that, she/he can decide the number n of local miners and the computing nodes to be involved during the clustering process. Moreover, if the local datasets

DOI: 10.1002/cpe

Scenario: a coordinator node and N local nodes, where local datasets $\{D_1, \ldots, D_N\}$ are stored

INITIALIZE:

1. The coordinator randomly initializes, for each cluster C_k, $k = 1, \ldots, K$ the centers m_k, the covariance matrices Σ_k, the mixing probabilities $p(m_k)$

2. The coordinator sends a copy of the centers, covariance matrices and mixing probabilities to each local site

3. Each i^{th}local site independently executes the following steps:

COMPUTELOCALMODEL:

3.1 it estimates $p(m_k|x)$ (for each $x \in D$, for each $k = 1, \ldots, K$), i.e., the percentage each object belongs to the cluster C_k:
$p(m_k|x) = \frac{p(x|m_k)\cdot p(m_k)}{\sum_{k=1}^{K} p(x|m_k)\cdot p(m_k)}$
where $p(m_k)$ is the mixing probability and $p(x|m_k)$ is the prior probability with Gaussian distribution

$$p(x|m_k) = \frac{1}{\sqrt{(2\pi)^d \cdot det(\Sigma_k)}} \cdot e^{-(x-m_k)\Sigma_k^{-1}(x-m_k)^T}$$

3.2 computes the local statistics $SS^{(i)}$, composed of the value $f^{(i)}$ and $\{s1_k^{(i)}, s2_k^{(i)}, s3_k^{(i)}\}$ for each cluster C_k, where
$f^{(i)} = \sum_{x \in D_i}[-\log \sum_{k=1}^{K} p(x|m_k)p(m_k)]$,
$s1_k^{(i)} = \sum_{x \in D_i} p(m_k|x)$
$s2_k^{(i)} = \sum_{x \in D_i} p(m_k|x)x$
$s3_k^{(i)} = \sum_{x \in D_i} p(m_k|x)(x - m_k)^T(x - m_k)$

3.3 it sends the local statistics to the coordinator;

COMPUTEGLOBALMODEL:

4. The coordinator adds up all the $SS^{(i)}$ received from each i^{th}local site, $i = 1, \ldots, N$, to compute the $Perf_{EM}$, the centers m_k, the covariance matrices Σ_k, the mixing probabilities $p(m_k)$ for $k, k = 1, \ldots, K$:
$Perf_{EM} = \sum_{i=1}^{N} f^{(i)}$
$m_k = \frac{\sum_{i=1}^{N} s2_k^{(i)}}{\sum_{i=1}^{N} s1_k^{(i)}}$, $\Sigma_k = \frac{\sum_{i=1}^{N} s3_k^{(i)}}{\sum_{i=1}^{N} s1_k^{(i)}}$, $p(m_k) = \frac{\sum_{i=1}^{N} s1_k^{(i)}}{|D|}$

NEEDSMOREITERATION:

5. If the performance function $Perf_{EM}$ converges or after a fixed number of iterations, the algorithm terminates; else, go to the step 2.

Figure 7. Distributed expectation maximization algorithm.

are not yet present on the local nodes, the user can firstly ask the system to split the whole dataset D in n (horizontal) partitions $\{D_1, \ldots, D_n\}$ (whose size is $|D|/n$), then send and store them on the local nodes. At this point, the *GlobalMiner-WS* is invoked by passing suitable parameters. The *GlobalMiner-WS* in turn contacts the *n LocalMiner-WS*s. In case of $n = 1$, the whole data set D is processed on the single node.

Experiments were carried out with a twofold goal. The first one consisted of a quantitative performance analysis aimed at evaluating the efficiency of the framework (Section 4.1). The second one aims at a qualitative analysis of the results by pointing out how the execution of the algorithms on this distributed environment affects the quality of the results (Section 4.2). The performance of the framework was tested on various datasets from the UCI Machine Learning Repository[‡].

4.1. Quantitative analysis and speedup

A first set of experiments was aimed at evaluating the scalability of the framework and the overhead introduced by the WSRF mechanisms used by it (with respect to the overall execution time). In order to achieve such a goal, we have chosen two large datasets, *CoverType*[§] and *Internet Advertisements*[¶], as data sources for the algorithms.

[‡]http://kdd.ics.uci.edu
[§]http://archive.ics.uci.edu/ml/datasets/Covertype
[¶]http://archive.ics.uci.edu/ml/datasets/Internet+Advertisements

DOI: 10.1002/cpe

The `CoverType` dataset contains information about forest cover type for 581,012 sites in the USA; each instance corresponds to a site observation and is described by 54 attributes giving information about the main features of a site (e.g., elevation, aspect, slope, etc.). The whole dataset has a size of about 72 MB. The `Internet Advertisements` dataset contains a set of possible advertisements on Internet pages; each record represents a Web page, the features encoding phrases occurring in the URL, the images URL and alt text, the anchor text, and words occurring near the anchor text. Such dataset contains 3279 records and 1558 attributes. The whole dataset has a size of about 10 MB. We point out that the two datasets were chosen in order to investigate the behavior of the framework (and of the algorithms exploiting it) in two transversal scenarios. The first one is a large dataset, characterized by a huge number of tuples and a relatively low number of attributes. The second one is an high-dimensional dataset, typified by a limited cardinality (number of tuples) and a great number of attributes. In the following, we will describe the tests executed on both algorithms as well as an evaluation of their performances.

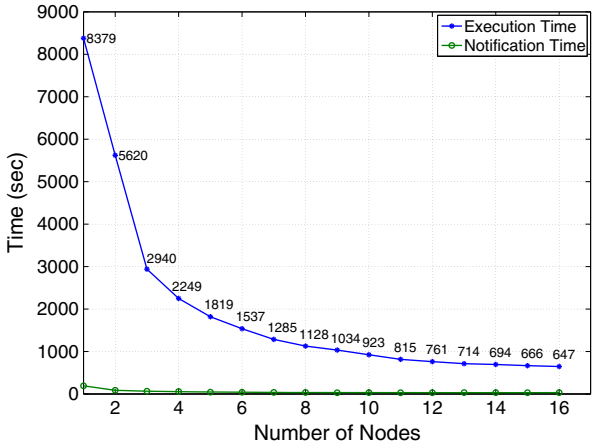
4.1.1. Distributed k-means. As it is known, each run of the k-means algorithm depends on the choice of the starting cluster centroids. In fact, the initial choice defines the specific local minimum that will be found by the algorithm, and it determines the number of the k-means iterations. In order to avoid the impact of the initial choice of the centroids on our timing measurements, for a fixed dataset, identical starting cluster centroids as well as a common maximum number of iterations are used.

We evaluated how the WSRF overhead contributes to the total execution time of the algorithms exploiting it. The WSRF overhead comes from all the steps to be executed to provide the algorithms as WSRF services, that is, resource creation, notification subscription, task submission, results notification, and resource destruction steps. In order to make a complete picture of the performances, we explored three experimental parameters. In fact, we evaluated the performance varying: (i) the number of nodes n; (ii) the number of attributes d; and (iii) the number of clusters k.

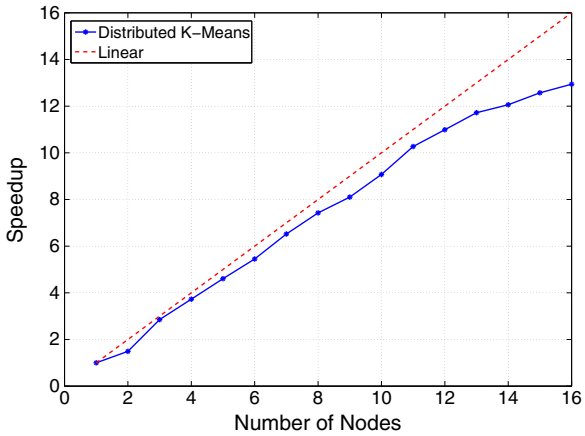
First, let us show the performance of the k-means algorithm by using a different number of nodes n. In order to perform such kind of experiments, we used the dataset *CoverType* as data source for the algorithm by fixing a common number of iterations ($maxIterations = 50$) and a common number of clusters ($numClusters = 7$) for all the experiments. Figure 8(a) shows two curves: the total *execution time* and the total *notification time*. The first one is the total time taken by the execution of the distributed algorithm, that is, the time elapsed from the submission of the task by the client (to the GlobalMiner-WS) until the final result is returned to it (by notification). The second one is the total time spent for the delivery of the results by the notification; in particular, it is the sum of the times spent for the local model notifications to the GlobalMiner-WS (summed up on all the iterations) and the time spent for the delivery of the global model notification (one time, containing the final result) to the client. As it will be described in the following, the other WSRF-overhead contributions resulted so low that, we guess, it is not interesting to report them in this plot. By observing the figure, we can notice an appreciable improvement in the performance when the number of nodes involved in the mining process increases: the total time execution decreases from 8379 s (obtained with one node) to 647 s (obtained with 16 nodes). Moreover, it is interesting to notice that the notification time is very low with respect to the execution time. This means that the portion of the time spent for the delivery of the computed models affects the total execution time in a very limited way. In addition to the execution time, another interesting evaluation index is the *relative speedup* that is the ratio of the execution time on one node to the execution time on n nodes. Figure 8(b) shows the achieved execution speedup: it ranges from 1.49 (using two nodes) to 12.95 (using 16 nodes). Finally, the *efficiency* is another metric to evaluate the scalability of the system. It is formally defined as the ratio of the relative speedup to the number of nodes used [24, 25]. In a few words, it is a value, between 0 and 1, estimating how well-utilized the nodes are in solving the problem, compared with the effort wasted in communication and synchronization. Table I reports the *efficiency* measured for the distributed k-means in the experiments. The 0.8 value for the efficiency on 16 nodes shows that all 16 computing nodes are exploited very well.

As anticipated earlier, another interesting issue is to evaluate how much the WSRF overhead contributes to the total execution time. In Figure 9(a), in addition to the computation time (that is,

DOI: 10.1002/cpe



(a) Execution and notification results time for different number of local nodes.



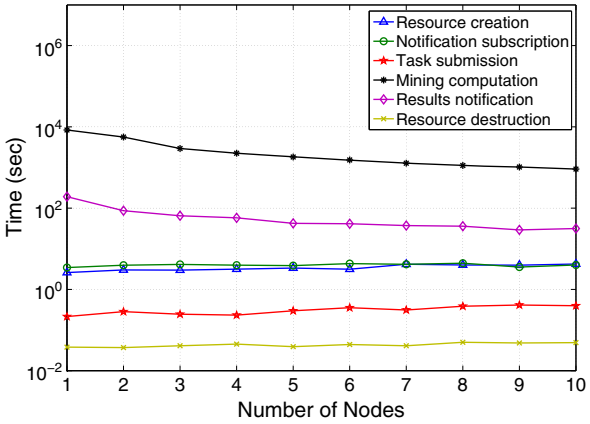
(b) Speedup values for different number of local nodes.

Figure 8. Distributed k-means performance.

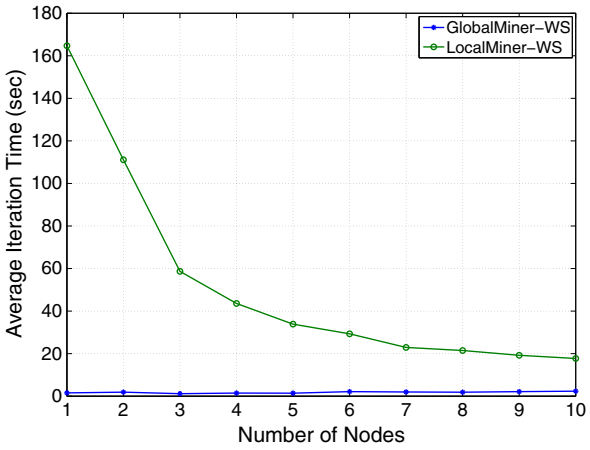
Table I. Distributed k-means efficiency.

Number of nodes	**Efficiency**
1	1
2	0.75
3	0.95
4	0.93
5	0.92
6	0.91
7	0.93
8	0.92
9	0.90
10	0.91
11	0.93
12	0.91
13	0.90
14	0.86
15	0.83
16	0.80

DOI: 10.1002/cpe



(a) Execution time of the different steps for different number of local nodes, logarithmic scale.



(b) Average execution time of an iteration for different number of local nodes.

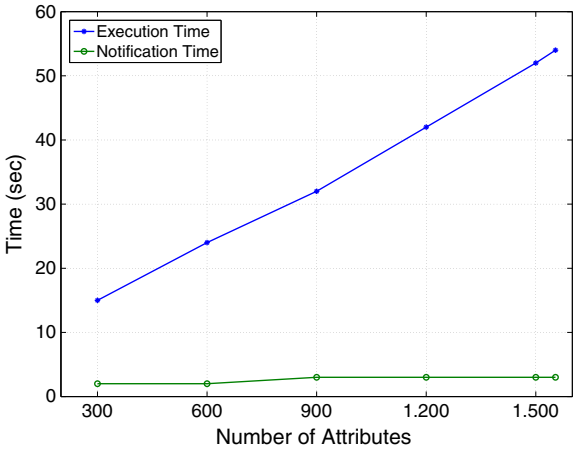
Figure 9. Distributed k-means performance.

the time spent for only mining computation), the execution times of the WSRF specific steps are reported. It is clearly shown that the computation time and results notification time are the two most time-consuming steps, whereas all the other WSRF steps (resource creation, notification subscription, task submission, resource destruction) need short execution time. From the plot in a logarithmic scale, results show that all the WSRF steps, with the exception of the notification, are negligible with respect to the computation time. Moreover, focusing on the computation time, we notice that it decreases as the number of the local nodes involved in the computation increases. This can be better appreciated by observing Figure 9(b), where the average computation time for a single iteration (on a local node and the coordinator node, respectively) is shown. Whereas execution time of the coordinator (the GlobalMiner-WS) seems to be unaffected by the number of local nodes involved, time of a single iteration on a local node (the LocalMiner-WS) decreases as the number of local nodes involved increases. This is justified by the fact that, in order to compute the local statistics at the current iteration, the local node has to compute the distance from each element (stored on the local dataset) to each cluster centroid, whose time complexity is $O(K \cdot \frac{|D|}{n})$ (or $O(K \cdot |D_i|)$, where $|D_i| = |D|/n$ is the size of the dataset stored on the $i-th$ local site). Indeed, in the specific case, to calculate the local statistics on each local site, the LocalMiner-WSs process datasets whose sizes range from 72 MB (when $n = 1$ and the dataset contains 581 K tuples) to 7.2 MB (when $n = 10$ and the dataset stores 58.1 K tuples), and this strongly reduces the computational time of each iteration.

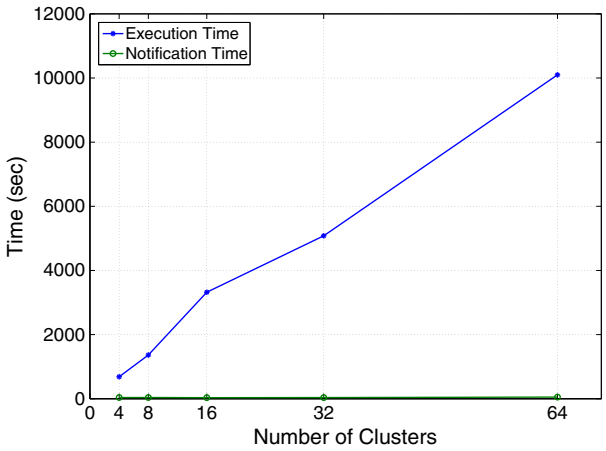
DOI: 10.1002/cpe

As a second evaluation criterion, we studied the system behavior when the number of attributes changes. For such kind of experiments, we used the dataset *Internet Advertising* as data source for the algorithm. Starting from the original dataset, six different datasets were built for different number of attributes. In particular, the values $d = 300, 600, 900, 1200, 1500,$ and1554 are used, where the last one refers to the dataset composed of the original number of attributes. We fixed the number of iterations ($maxIterations = 10$) and the number of clusters ($numClusters = 2$) for all the experiments. Figure 10(a) shows the execution time and notification time with different numbers of attributes. As expected, the execution time linearly increases with respect to the number of attributes (because the number of floating point operations to be computed for the distance calculations are directly proportional to the dimensionality of the dataset). On the contrary, we want to point out that the notification time increases very slowly with the increment of the attributes. This shows that, even though high-dimensional datasets are processed, the notification step takes a relatively low and constant time.

In a third evaluation, we studied the performance of the k-means exploiting the framework when the number of clusters changes. We used the dataset *CoverType* as data source for the algorithm by fixing the number of iterations ($maxIterations = 50$) and executed the tests by using eight nodes. Figure 10(b) shows how the execution time and notification time vary when the number of clusters



(a) Execution time and Notification time fo different number of attributes, on 3 Nodes.



(b) Execution time and Notification time fo different number of clusters, on 8 Nodes.

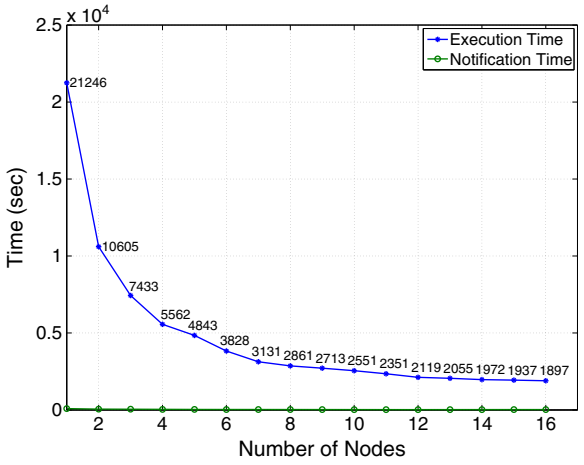
Figure 10. Distributed k-means performance for different number of attributes and clusters.

DOI: 10.1002/cpe

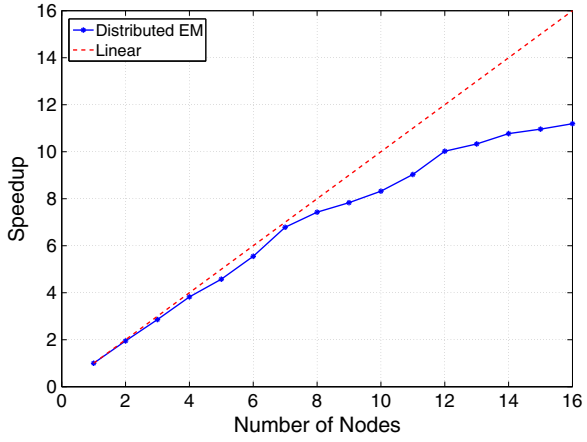
increases. Also in this case, the notification time increases very slowly when the number of clusters grows and contributes to the total execution time in a very limited amount.

4.1.2. Distributed expectation maximization. Similarly, to the tests performed on the distributed k-means, various experiments have been carried out by executing the distributed EM exploiting the proposed framework. Also in this case, because the initial choice of the model parameters and mixing probabilities could influence the behavior of the computation, identical centers, covariance matrices, and mixing probabilities (as well as a common maximum number of iterations) for a fixed dataset have been chosen.

The first goal of such experiments was the evaluation of the framework scalability and the WSRF overhead introduced by the framework. For such tests, the *CoverType* dataset has been fixed as data source for the algorithm, as well as a common number of iterations (*maxIterations* = 10) and clusters (*numClusters* = 7) have been used. The *execution time* and *notification time* trends are shown in Figure 11(a), where an appreciable improvement can be noticed when the number of local nodes is increased. In particular, the total execution time decreases from 21,246 s (obtained using one local node) to 2551 s (obtained with 16 nodes). Also the speedup, plotted in the Figure 11(b), shows a good shape, ranging from 1.94 (using two nodes) to 11.19 (using 16 nodes). Finally, Table II reports the *efficiency*, which is 93% on eight nodes and 69% on 16 nodes.



(a) Execution and notification results time fo different number of local nodes.



(b) Speedup values for different number of local nodes.

Figure 11. Distributed EM performance.

DOI: 10.1002/cpe

Table II. Distributed EM efficiency.

Number of nodes	Efficiency
1	1
2	0.97
3	0.95
4	0.96
5	0.91
6	0.92
7	0.97
8	0.93
9	0.87
10	0.83
11	0.82
12	0.83
13	0.79
14	0.76
15	0.73
16	0.69

As we have carried out in the evaluation of the k-means, we have compared the elapsed time for data mining computation and that spent as WSRF overhead. In Figure 12(a), such different times are reported for increasing values of the number of nodes. As it is clear, the time spent for the execution of all the WSRF steps is a minimal part with respect to that spent in the mining computation.

As a final step, we show the performance (on *Internet Advertising* dataset) when the number of attributes varies. Figure 12(b) shows how the execution time and notification time vary when the number of attributes increases. Also in this case, the notification time grows very slowly when the number of attributes increases and contributes to the total execution time in a very limited amount.

4.2. Qualitative analysis

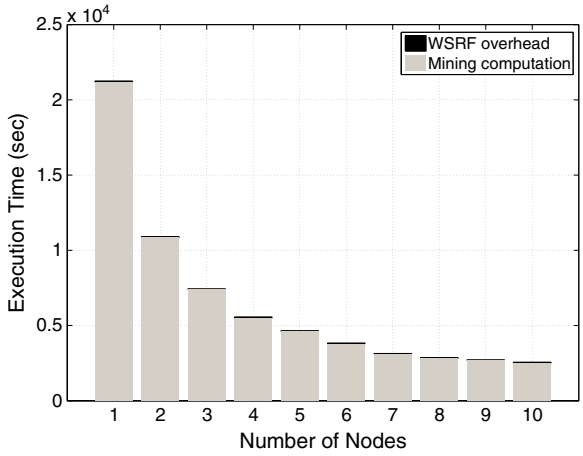
In this subsection, we discuss, from a qualitative viewpoint, the results obtained by evaluating the framework. As a preliminary consideration, it is worth noting that the two algorithms exploited in the experiments build the same models that could be extracted by sequential (centralized) algorithms; in other words, the framework has produced the same data mining models when a different number of nodes is used.

An important aspect that we want to deal with is that the framework is general enough to perform preliminary initialization steps (if needed) in addition to mining processes. In particular, for the k-means, we have implemented different techniques for the initialization of the centroids, and we will show the obtained results in terms of accuracy. In order to do that, we adopt two external criteria, the *error rate* and the *F-index*. The *error rate* is an index aimed at evaluating the inaccuracy (imprecision) of the classification performed by the clustering process, whereas the *F-index* quantifies the compactness of the discovered model. Clearly, low values of the first index and high values of the second one are evidence of good-quality clustering. The experiments have been performed on the *Sonar*[‖], *Vehicle*[**] and *Iris*[††] datasets of the UCI Repository.

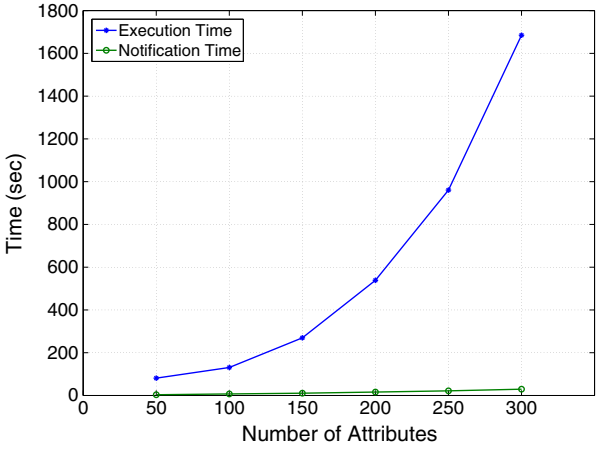
It is well known that the k-means algorithm suffers from initial starting conditions effects (initialization and instance order effects). In order to achieve better results in terms of quality of the final result and number of iterations, various initialization techniques have been proposed in the literature [26–29]. In this section, we do not propose a new technique for centroid initialization, but we want to show that the proposed framework is sufficiently general to let the user use different initialization methods, if they are implemented. To this purpose, we have chosen three initialization methods for

[‖]http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar2C+Mines+vs.+Rocks)
[**]http://archive.ics.uci.edu/ml/datasets/Statlog+(Vehicle+Silhouettes)
[††]http://archive.ics.uci.edu/ml/datasets/Iris

DOI: 10.1002/cpe



(a) Computation time and WSRF overhead for different number of nodes.



(b) Execution time and Notification time for different number of attributes, on 3 Nodes.

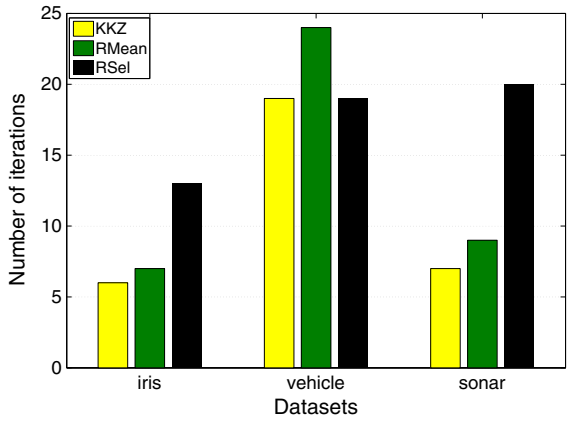
Figure 12. Distributed EM performance.

the k-means, and we have exploited them in our implementation. In the following, a brief description of such techniques is reported. For more details, the reader could refer to [29].

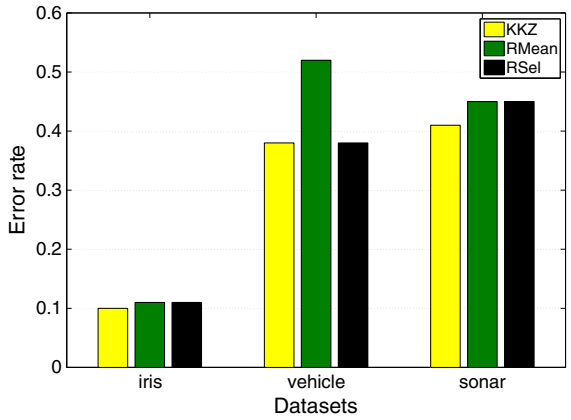
The *R-SEL* adopts uniformly random input samples as the seed clusters by choosing the first randomly selected K input instances (where K is the number of clusters); it was one of the earliest methods proposed in the literature, and it is roughly known as random initialization. The *KKZ* is a method that utilizes the sorted pairwise distances for initialization. Initially, the data object with the maximal norm is chosen to be the center of the cluster C_1. Then, the rest of the representative instances are selected according to the heuristic rule of collecting the closest element to c_k, for each centroid c_k yet chosen, and setting as new centroid the element with the maximum distance. The method terminates when K centroids have been selected. The *R-MEAN*, based on a Gaussian distribution of a given mean and variance, generates an array of K indices identifying the K elements selected from the dataset as initial centroids.

We have implemented all three of these initialization methods of the k-means, and we have also performed experiments on some datasets by varying the initialization technique. Figure 13 shows some qualitative results by exploiting three local nodes. In particular, we are interested in the evaluation of the *convergence rate*, that is, the number of iterations needed for the algorithm termination and the quality of the results, that is, the *error rate* and the *F-Index*. We have noticed that the KKZ

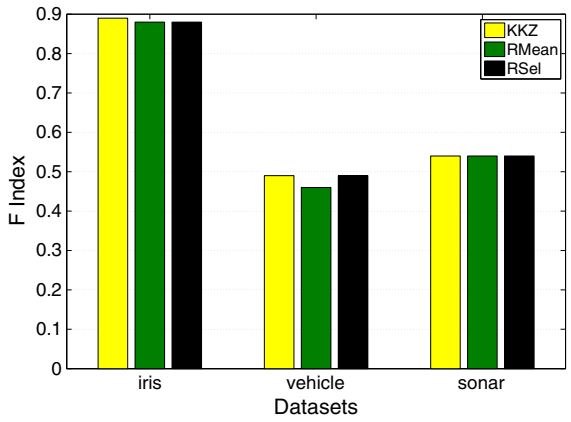
DOI: 10.1002/cpe



(a) Number of iterations.



(b) Error rate.



(c) F Index.

Figure 13. Distributed k-means quality.

initialization obtains better results with respect to both the R-SEL and R-MEAN. In fact, on all of the three datasets, the KKZ induces the algorithm to achieve the convergence in the lowest number of iterations, as well as to obtain low error rate and high values of F-index.

4.3. Prototype implementation: technical details

This section provides some technical details about the implementation of the framework and some low-level specifications on its usage.

DOI: 10.1002/cpe

The framework prototype has been implemented in Java. The two grid services have been developed by using the Java WSRF library provided by the WS-Core, a component of the Globus Toolkit 4 [23]. SOAP over HTTP have been used as default message format and transport protocol. All services are fully OGSA 1.0 compliant and can be deployed on pre-installed Globus Toolkit 4.0.x (GT4) stand-alone containers. As it can be seen in Figure 2, the deployment and usage of the framework contemplate the presence of various nodes distributed in the grid. These nodes are distinguished in two categories: the global node that performs coordination tasks, and the local nodes that execute the local mining tasks on the datasets that are located on them. The usage of the framework needs the deployment of a *GlobalMiner-WS* on the central node and a certain number of *LocalMiner-WS* on the local nodes.

From the implementation viewpoint, each service can be ideally distinguished in two components: the *Web service* component and the *algorithm library* component. In both types of nodes, the *Web service* component acts as a WSRF-compliant interface for the invocation of the operations, whereas the *algorithm library* is a code library providing the algorithms to be executed (at the global and local level). In other words, the *Web service* components take care of the WSRF-specific stuff and mechanisms, (i.e., the resource creation/destruction, notification subscription and delivery, operation invocation, etc.) and execute the mining tasks by invoking the corresponding algorithms provided in the *GAL/LAL*. We point out that the components are completely decoupled and independent; whenever a new algorithm is added to the library, it just needs the update of the code library (e.g., it does not need to redeploy the service on the GT4 container).

The proposed architecture is able to execute different classes of mining algorithms and not just clustering tasks. The framework is easily extensible, that is, other mining algorithms could be simply added by providing the algorithmic logic of the coordinator and the local worker to the algorithm library. For example, whereas a developer wants to add an *X-algorithm* to the framework, it has to develop the *X-GlobalAlgorithm* and the *X-LocalAlgorithm* by implementing the generic *GlobalAlgorithm* and *LocalAlgorithm* interfaces provided in the libraries (see Figure 5). Such software will integrate the *GAL* and *LAL*.

In order to guarantee generality to the framework, all the resource properties used in the services are string-based. Actually, whereas a new algorithm has to be added to the framework, it is desirable to utilize a standard representation (such as PMML) to reduce implementation complexity (even though the discovered structures and models could be stored in every representation the developer wants). As a consequence of this, the information delivered by notification (mining models, statistics, etc.) is serialized as strings. Moreover, in order to minimize the delivery time of notifications (in some cases carrying large amount of information), the messages are compressed before their delivery and are decompressed at destination. This is carried out both for the local-global and for global-client notification by using the Java Zip library: it reduces the delivery time by more than 35%.

Finally, the parameters passed to the `submitGlobalTask` and `submitLocalTask` operations are generic *TaskDescriptor* objects containing the task parameters (algorithm name and parameters, initialization requests, etc.) needed for the task execution. Currently, the datasets on the local nodes must be stored in the file system as flat file. Adding support for retrieving data from a database could be carried out by OGSA-DAI without adding particular technical difficulties to the developer.

5. RELATED WORK

Recently, several DDM systems exploiting the grid infrastructure have been designed and implemented [30]. Some of them (FAEHIM [31], GridWeka [32], Weka4WS) have been designed as modifications/extensions of *Weka* [33], a well-known toolkit for machine learning and data mining. In Weka, the overall data mining process takes place on a single machine, whereas in distributed extensions, a data mining process can be executed concurrently in a distributed environment. Other grid-based data mining frameworks have been designed and implemented from scratch, such as the Knowledge Grid, GridMiner, Discovery Net, and DataMiningGrid.

Weka4WS [34] is a framework that extends the Weka toolkit for supporting DDM on grid environments through WSRF Web services. In such a way, DDM tasks can be concurrently executed

DOI: 10.1002/cpe

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 also executed on remote grid resources. To enable remote invocation, all the data mining algorithms provided by the Weka library are exposed as Web services, which can be easily deployed on the available grid nodes. To achieve integration and interoperability with standard grid environments, Weka4WS has been developed by using the WSRF Java library provided by GT4 [23].

Discovery Net [35] is a system allowing users to integrate data analysis software and data sources made available by third parties. The building blocks are the so-called knowledge discovery services, distinguished in computation services and data services. Discovery Net provides services, mechanisms, and tools for specifying knowledge discovery processes. The functionalities of Discovery Net can be accessed through an interface exposed as an OGSA-compliant grid service. However, Discovery Net currently uses an early implementation of OGSA (namely, the Open Grid Services Infrastructure (OGSI)), which has been replaced by WSRF because of lack of compatibility with standard Web service technologies.

The *GridMiner-Core* [36] is a framework that supports development and runtime execution of data mining and data preprocessing applications running in grid environments. It aims at data mining application developers that want an easy-to-use framework supporting them in either porting their existing applications or developing new applications that run in grid environments, without the necessity to handle grid-specific stuff. In other words, it provides the core data mining functionality, that is in fact, delegated to external toolkits, and, in this way plugged homogeneously into the system. More in detail, the system allows the user: (i) to collect and aggregate information about available services and provides querying interfaces for resource discovery and monitoring; (ii) to submit jobs described by activity description (modeled as XML document) and monitor their execution status; and (iii) to access and manipulate various kinds of data sources and allocate/assign resources to grid application.

The *Knowledge Grid* [37] is an environment providing knowledge discovery services for a wide range of high performance distributed applications. Such a framework makes use of basic grid services to build more specific services supporting distributed knowledge discovery in databases on the grid. Such services allow users to implement knowledge discovery applications that involve data, software, and computational resources available from distributed grid sites. The *Knowledge Grid* supports data mining activities by providing mechanisms and higher level services for searching resources, representing, creating, and managing knowledge discovery processes, and for composing existing data transfer services and data mining services in a structured manner, allowing designers to plan, store, document, verify, share, and re-execute their workflows as well as manage their output results. Previous work on the Knowledge Grid has been focused on the development of the system by using early grid middleware [38], as well as the design and evaluation of distributed KDD applications [39]. The WSRF has been adopted for reimplementing the Knowledge Grid services following the SOA approach [40].

The *DataMiningGrid* [41] software is an environment suitable for executing data analysis and knowledge discovery tasks in a wide range of different application sectors, including the automotive, biological and medical, environmental and ICT sectors. Based on open-source grid middleware, it provides functionality for tasks such as data manipulation, resource brokering and application searching according to different data mining tasks and methodologies, and supporting different types of functionality for parameter sweeps. The result is a grid software with all the generic functionality of its component middleware, but with additional features that ease the development and execution of complex data mining tasks.

Recently, in [42], the Data Mining and Integration *(DMI)* architecture has been proposed, even though not yet implemented. It is a framework to perform both data mining and data integration tasks[‡‡]. It supports enactment of DMI processes across heterogeneous and distributed data resources and data mining services. It posits that a useful division can be made between the facilities established to support the definition of DMI processes and the computational infrastructure

[‡‡]Such combined activity is referred to as *DMI activity*.

DOI: 10.1002/cpe

provided to enact DMI processes. Moreover, the communication between those two divisions is restricted to requests submitted to gateway services in a canonical DMI language. Autonomous changes/evolution of data resources and services are supported by types and descriptions, which will support detection of inconsistencies and semi-automatic insertion of adaptations.

Differently from the systems discussed earlier, the architecture presented here implements a service-oriented approach that provides a suitable WSRF-compliant infrastructure for executing some specific prearranged patterns of DDM in a grid environment. As described in the Sections 2.3 and 4.3, any process exploiting the master/worker distributed architecture reported in Figure 1 could be easily integrated in our framework. Moreover, the experimental evaluation has shown that the architecture is efficient, in the sense that the WSRF overhead[§§] is very low with respect to the total computation time. This guarantees also a good scalability, as evicted from the experiments conducted in a real grid.

Here, we point out some specific differences among the presented frameworks and some systems discussed in the section. *Weka4WS* exploits the grid for executing DDM tasks as single services or composing tasks in workflows that are not prearranged for meta-learning. If a user needs to implement a DDM algorithm exploiting a master/worker schema, he/she needs to design from scratch the workflow implementing it. Thus, in this case, our framework seems to be more suitable because it provides such a prearranged pattern. A different observation has to be carried out for the *KnowledgeGrid* that is a framework for composing (and executing) workflows by arranging existing services (data transfer, data mining) available on the grid. In this sense, the proposed architecture could be used by the *KnowledgeGrid* as complex mining service (e.g., a meta-learning service) available on the grid, and that can be easily integrated in larger mining workflows. It is evident that, in such a scenario, the *KnowledgeGrid* can act as a client of the *GlobalMiner-WS*, which will execute the whole mining process by "delegating", in turn, partial processes to the available *LocalMiner-WS* services. A similar exploitation of the framework proposed in this paper could be performed by the *GridMiner* and *DataMiningGrid* systems.

Finally, in order to provide the reader with a quantitative comparison of the system proposed in the paper and the others cited here, in the following, we report a brief description of the performance results obtained by them.

Experiments of the *DataMiningGrid* framework are presented in its descriptive paper [41]. In a first scenario, the framework was employed: (i) to induce models of aquatic ecosystems from modeling knowledge and measurement data; and (ii) to simulate these models under changing different environmental conditions. The test bed includes 18 servers with the Globus Toolkit 4 and Condor installations. The experiments show that, by changing the number of machines from 1 to 10, a speedup of 6.7 was obtained. Furthermore, by increasing the number of machines by a factor of 2.5 and the workload by a factor of 2.0, a scale-up of 1.8 has been achieved. The second scenario concerns a classification of millions of documents to be labeled according to a subject. Authors state that, for a central file server with Gigabit Ethernet, a linear speedup was measured only for up to five machines.

The experimental results of the *GridMiner-Core* framework are described in [36]. In particular, the authors show the dataset size-up behavior for a single client, during a C4.5 algorithm execution on a synthetic dataset (the *Weather* dataset filled with synthetic records): the algorithm execution time grows linearly with the dataset size by 4.2 from 10,000 to 100,000 and by 7.6 from 25,000 to 250,000 records in the dataset. This was a local experiment aimed at evaluating the size-up behavior of the system. More interesting is a distributed classification algorithm denoted as DIGIDT that has been designed and implemented as pilot application to demonstrate the functionality of *GridMiner-Core*. In particular, DIGIDT builds a decision tree classifier from datasets that are distributed over several sites, and its schema contemplates the presence of one master and n slave nodes. The experiment environment was a workstation cluster whose nodes were interconnected via a 100-Mbit Ethernet. The scale-up results show a reduction of the response time from about 10,329 s (with a single worker node) to 7472 (with two worker nodes) and to 7076 s (with four worker nodes)

[§§]The whole set of steps needed to make usable the algorithms, such as WSRF service.

DOI: 10.1002/cpe

Table III. Performance comparison among some systems.

Framework	Applicative scenario	Test bed hardware configuration	Speedup
System proposed in the paper	Distributed clustering	Grid composed of 18 heterogeneous nodes	1.5 (2 nodes) 7.4 (8 nodes) 9.1 (10 nodes) 12.9 (16 nodes)
DataMiningGrid	Ecosystem modeling	Grid composed of 18 heterogeneous nodes	6.7 (10 nodes)
GridMiner-Core	Distributed classification	workstation cluster of 4 nodes	1.4 (2 nodes) 1.5 (4 nodes)
KnowledgeGrid	Classification task for intrusion detection of network data	Grid composed of 8 heterogeneous nodes	2.0 (3 nodes) 4.9 (8 nodes)
Weka4WS	Parameter sweeping on clustering task	Grid composed of 6 heterogeneous nodes	1.8 (2 nodes) 4.5 (6 nodes)

when totally 10 million records have been classified. From these values, we can deduce a speed-up of 1.4 and 1.5 when two and four nodes are exploited, respectively. No results on a higher number of nodes are reported.

A performance analysis of the *Knowledge Grid* is reported in [38], where a classification task for intrusion detection of network data has been described. In particular, a number of independent classifiers are first computed by applying in parallel the same learning algorithm (i.e., C4.5) over a set of distributed training sets and, finally, the best classifier has been chosen by means of a voting operation based on the error rate and confusion matrix. The experiments were performed on a grid composed of eight heterogenous machine nodes. The paper reports the execution times and speedup achieved with a different number of nodes. In particular, a speedup factor of 2.0 has been achieved on three nodes, and a speedup of 4.9 was obtained by using eight nodes.

Two different papers [34, 43] describe the experimental evaluation of *Weka4WS*. In [34], the framework is exploited to analyze the *Covertype* dataset by running multiple instances of the same clustering algorithm, with the goal of obtaining multiple clustering models from the same data source. In particular, Weka4WS has been used to run an application in which six independent instances of the k-means algorithm (each one typified by a different number of clusters) perform a different clustering task on the given data set. The same application has been executed using a number of computing nodes ranging from 1 to 6. The paper points out an appreciable execution time decreasing the number of computing nodes increases, and the achieved execution speedup ranged from 1.8 (using two nodes) to 4.5 (using six nodes). In a second experimental evaluation, described in [43], a dataset is analyzed in parallel on multiple grid nodes using different data mining classification algorithms. In particular, the *KDDCup99* dataset is analyzed in parallel using four different classification algorithms, that is, Decision Stump, Naive Bayes, J48, and Random Forest, with the aim of finding the "best" classifier on the basis of some evaluation criteria. The experiments that run on four nodes show that, for the largest dataset used (1000-k instances), the system achieves a speedup of 2.1 when four nodes are used. The authors observe that the speedup is not particularly high (as might be expected) because of the diversity of the classification algorithms used in the example, coupled with the heterogeneity of computing nodes.

Table III summarizes the results obtained by all these systems, by reporting for each one the applicative scenario, the environment (i.e., number of nodes), and the achieved speedup. If a system has been exploited in more experiments, the table reports the most significant result.

6. CONCLUDING REMARKS

In this paper, we have described how some DDM patterns can be implemented as mining services by exploiting the grid infrastructure. We have defined a distributed architectural model that can be exploited for different distributed mining algorithms deployed as grid services, for the analysis of

DOI: 10.1002/cpe

dispersed data sources. The implementation of two distributed clustering algorithms, k-means and expectation maximization, on the proposed architecture, as well as an experimental evaluation on real data sets have been presented. Such algorithms have been implemented and deployed as WSRF services on the Globus Toolkit, and intensive experimental evaluation on a real grid has shown good performance. In fact, experiments performed on large and high-dimensional datasets have shown good scalability of the framework in various test scenarios. Moreover, it is pointed out that the WSRF overhead contributes to the total execution time in a very limited amount.

The framework described in the paper has various peculiarities, as sketched in the following:

1. If a user needs to exploit a master/worker schema implementing DDM algorithms, the framework provides a specific prearranged pattern; thus, the design of an ad hoc structure (or script) modeling the application to be executed is not requested to the user.
2. The integration of a new algorithm is very easy; in fact, it is sufficient to implement two program interfaces, the `GlobalAlgorithm` and `LocalAlgorithm` (see Figure 5). Such software will integrate the two code libraries *GAL* and *LAL* that automatically make the new algorithm available in the system.
3. The algorithms are available as WSRF services, guaranteeing standard access and managing of stateful resources in a grid environment, allowing their invocation from Web and Cloud applications.
4. The experiments have shown that the framework achieves good scalability and efficiency; so, its usage is convenient also from the performance viewpoint.

As a future work, our further research work will investigate some additional issues. In particular, we would like to deploy other distributed mining algorithms (clustering, frequent itemsets, association rule), in order to make it more complete from an algorithmic viewpoint. Moreover, fault-tolerant strategies will be designed to handle possible disconnections of the coordinator or local nodes. Finally, a technique for the automatic choice both of the optimal value of distribution degree (number of local miners) and of the involved nodes (with respect to their hardware/software status) will be studied.

REFERENCES

1. Tan PN, Steinbach M, *et al. Introduction to Data Mining*. Addison-Wesley: Boston, Massachusetts, 2006.
2. Fu Y. Distributed data mining: an overview. *IEEE TCDP Newsletter*, 2001.
3. Johnson E, Kargupta H. Collective, hierarchical clustering from distributed, heterogeneous data. In *Large-Scale Parallel KDD Systems*, Zaki M, Ho C (eds). Springer–Verlag: San Diego, CA, 2000; 217–249.
4. Zaki M. Parallel and distributed association mining: a survey. *IEEE Concurrency* 1999; **7**:14–25.
5. Distributed Data Mining Bibliography. (Available from: http://www.csee.umbc.edu/~hillol/DDMBIB/ [16 July 2008]).
6. Prodromidis AL, Chan PK, *et al*. 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.
7. Kargupta H, Park B, *et al*. 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.
8. Foster I, Kesselman C, *et al*. 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.
9. Samatova NF, Ostrouchov G, *et al*. RACHET: an efficient cover-based merging of clustering hierarchies from distributed datasets. *Distributed and Parallel Databases* 2002; **11**:157–180.
10. Januzaj E, Kriegel HP, *et al*. Scalable density-based distributed clustering. *Proc. 8th European Conference on Principles and Practice of Knowledge Discovery*, 2004.
11. Zhang B, Hsu M, *et al*. Accurate recasting of parameter estimation algorithms using sufficient statistics for efficient parallel speed-up: demonstrated for center-based data clustering algorithms. *Proc. 4th European Conference on Principles and Practices of Knowledge Discovery in Databases*, 2000.
12. Dhillon I, Modha D. A data-clustering on distributed memory processors. *Proc. KDD Workshop on High Performance Knowledge Discovery*, 1999.
13. Hall L, Chawla N, *et al*. Combining decision trees learned in parallel. *In Working Notes of the KDD-97 Workshop on Distributed Data Mining*, 1997.
14. Agrawal R, Shafer JC. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering* 1996; **8**(6):962–969.

DOI: 10.1002/cpe

15. Zaiane OR, El-Hajj M, *et al*. Fast parallel association rule mining without candidacy generation. *Proc. IEEE International Conference on Data Mining*, 2001.
16. Taniar MZ, Smith D, *et al*. ODAM: an optimized distributed association rule mining algorithm. *IEEE Distributed Systems Online* 2004; **5**(3):1541–4922.
17. Cesario E, Talia D. Distributed data mining models as services on the grid. *Proc. 10th International Workshop on High Performance Data Mining*, 2008.
18. Czajkowski K, Ferguson DF, *et al*. The WS-Resource Framework Version 1.0 2004. (Available from: http://www.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf [16 July 2008]).
19. Sotomayor B, Childers L. *Globus Toolkit 4: Programming Java Services*. Morgan Kaufmann: San Francisco, CA, 2006.
20. McConnell SM, Skillicorn DB. Building predictors from vertically distributed data. *Proc. 2004 Conference of the Centre for Advanced Studies on Collaborative Research*, 2004.
21. Sayal M, Scheuermann P. A distributed clustering algorithm for Web based access patterns. *Proc. KDD Workshop on Distributed and Parallel Knowledge Discovery*, 2000.
22. Gu D. Distributed EM algorithm for Gaussian mixtures in sensor networks. *IEEE Transactions on Neural Networks* 2008; **19**(7):1154–1166.
23. The Globus Toolkit. (Available from: http://www.globus.org/toolkit/ [16 July 2008]).
24. Grama A, Gupta A, *et al*. Isoefficiency: measuring the scalability of parallel algorithms and architectures. *IEEE Journal of Parallel and Distributed Technology: Systems and Technology* 1993; **1**(3):12–21.
25. Kumar V, Gupta A. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing* 1994; **22**(3):379–391.
26. Bradley PS, Fayyad UM. Refining initial points for k-means clustering. *Proc. International Conference on Machine Learning*, 1998.
27. Pizzuti C, Talia D, *et al*. A divise initialisation method for clustering algorithms. *Proc. European Conference on Principles of Data Mining and Knowledge Discovery*, 1999.
28. Pena JM, Lozano JA, *et al*. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters* 1999; **20**(10):1027–1040.
29. He J, Lan M, *et al*. Initialization of cluster refinement algorithms: a review and comparative study. *Proc. IEEE International Joint Conference on Neural Networks*, 2004.
30. Talia D, Trunfio P. How distributed data mining tasks can thrive as knowledge services. *Communications of the ACM* 2010; **53**(7):132–137.
31. Shaikh AA, Rana OF, *et al*. Web services composition for distributed data mining. *Proc. Workshop on Web and Grid Services for Scientific Data Analysis*, 2005.
32. Khoussainov R, Zuo X, *et al*. Grid-enabled Weka: a toolkit for machine learning on the grid. *ERCIM News* 2004; **59**.
33. Witten IH, Frank E. *Data Mining: Practical Machine Learning Tools with Java Implementations*. Morgan Kaufmann: New York, NY, 2000.
34. Talia D, Trunfio P, *et al*. The Weka4WS framework for distributed data mining in service-oriented grids. *Concurrency and Computation: Practice and Experience* 2008; **20**(16):1933–1951.
35. AlSairafi S, Emmanouil FS, *et al*. The design of Discovery Net: towards open grid services for knowledge discovery. *International Journal of High Performance Computing Applications* 2003; **17**(3):297–315.
36. Hofer J, Brezany P. DIGIDT: distributed classifier construction in the grid data mining framework GridMiner-Core. *Proc. Workshop on Data Mining and the Grid*, 2004.
37. Cannataro M, Talia D. The Knowledge Grid. *Communications of the ACM* 2003; **46**(1):89–93.
38. Cannataro M, Congiusta A, *et al*. Distributed data mining on grids: services, tools, and applications. *IEEE Transactions on Systems, Man, and Cybernetics Part B* 2004; **34**(6):2451–2465.
39. Bueti G, Congiusta A, *et al*. Developing distributed data mining applications in the Knowledge Grid framework. *Proc. International Meeting on High Performance Computing for Computational Science*, 2004.
40. Congiusta A, Talia D, *et al*. Distributed data mining services leveraging WSRF. *Future Generation Computer Systems* 2007; **23**(1):34–41.
41. Stankovskia V, Swainb M, *et al*. Grid-enabling data mining applications with DataMiningGrid: an architectural perspective. *Future Generation Computer Systems* 2008; **24**(4):259–279.
42. Atkinson MP, van Hemert JI, *et al*. A distributed architecture for data mining and integration. *Proc. 2nd International Workshop on Data-Aware Distributed Computing*, 2009.
43. Lackovic M, Talia D, *et al*. Service oriented KDD: a framework for grid data mining workflows. *Proc. 10th International Workshop on High Performance Data Mining (HPDM 2008), In Conjunction with ICDM'08*, 2008.

DOI: 10.1002/cpe