A data mining toolset for distributed high-performance platforms

M. Cannataro[1], A. Congiusta[2], D. Talia[2] & P. Trunfio[2]
[1] *ICAR-CNR, Rende (CS), Italy*
[2] *DEIS, University of Calabria, Rende (CS), Italy*

Abstract

Today a large number of scientific and commercial applications often require to analyse large data sets maintained over geographically distributed sites by using the computational power of distributed high-performance environments. Advances in networking technology and computational infrastructure made it possible to construct large-scale distributed computing platforms, called *computational grids*, that provide dependable, consistent, and pervasive access to high-end computational resources. Grids can play a significant role in providing an effective computational support for distributed data mining applications. Currently we are developing a software system for geographically distributed knowledge discovery applications called KNOWLEDGE GRID, which is designed on top of computational grid mechanisms, provided by grid environments such as *Globus*. In this paper we present an integrated toolset named *VEGA* (*Visual Environment for Grid Applications*), which allows a Knowledge Grid user to develop and execute distributed data mining computations in a simple and effective way.

1 Introduction

In many industrial, scientific and commercial applications, it is often necessary to mine large distributed data sets by using the computational power of distributed high-performance computers. Advances in networking technology and computational infrastructure made it possible to design *computational grids* as large-scale distributed computing platforms that provide dependable, consistent, and pervasive access to high-end computational resources. The term *grid* refers to an emerging infrastructure that enables the integrated use of remote computers, databases, scientific instruments, and other resources [1]. Grid applications often involve large amounts of computing and data. For these reasons, the grids can play a significant role in providing an effective computational support for distributed data mining applications.

We designed a software architecture for geographically distributed knowledge discovery applications called KNOWLEDGE GRID [2], which is designed on top of computational grid mechanisms, provided by grid environments such as Globus. The KNOWLEDGE GRID uses the basic grid services such as communication, authentication, information, and resource management to build more specific distributed data mining tools and services. This paper presents a KNOWLEDGE GRID toolset, named VEGA, for the composition and execution of distributed data mining computations over a Globus-based grid. Such toolset allows a user to build the computation starting from a set of useful remote resources (e.g., computational nodes, sources of data, data mining suites, etc.). Such resources, which are located and selected by means of grid information services, are presented to the user as a set of objects. The user can compose those objects using common visual facilities, to form a graphic representation of her/his data mining computations. The toolset validates and translates this graphic representation into an execution plan, which is then processed and effectively executed on the grid by means of Globus resource management tools.

2. The KNOWLEDGE GRID architecture

The KNOWLEDGE GRID architecture [2] (see Figure 1) is designed on top of computational grid mechanisms, provided by grid environments such as Globus [1]. The KNOWLEDGE GRID uses the basic grid services such as communication, authentication, information, and resource management to build more specific parallel and distributed knowledge discovery (PDKD) tools and services.

The KNOWLEDGE GRID services are organized into two layers: *core K-grid layer*, which is built on top of generic grid services, and *high level K-grid layer*, which is implemented over the core layer.

The core K-grid layer comprises two basic services: the *Knowledge Directory Service* (*KDS*) and the *Resources Allocation and Execution Management Service* (*RAEMS*). The KDS manages the metadata describing the characteristics of relevant objects for PDKD applications, such as data sources, data mining software, results of computations, data and results manipulation tools, execution plans, etc. The information managed by the KDS is stored into three ad hoc repositories: the metadata describing features of data, software and tools, coded in XML documents, are stored in a *Knowledge Metadata Repository* (*KMR*), the information about the knowledge discovered after a PDKD computation is stored in a *Knowledge Base Repository* (*KBR*), whereas the *Knowledge Execution Plan Repository* (*KEPR*) stores the execution plans describing PDKD applications over the grid. The goal of RAEMS is to find a mapping between an execution plan and available resources on the grid, satisfying user, data and algorithms requirements and constraints.

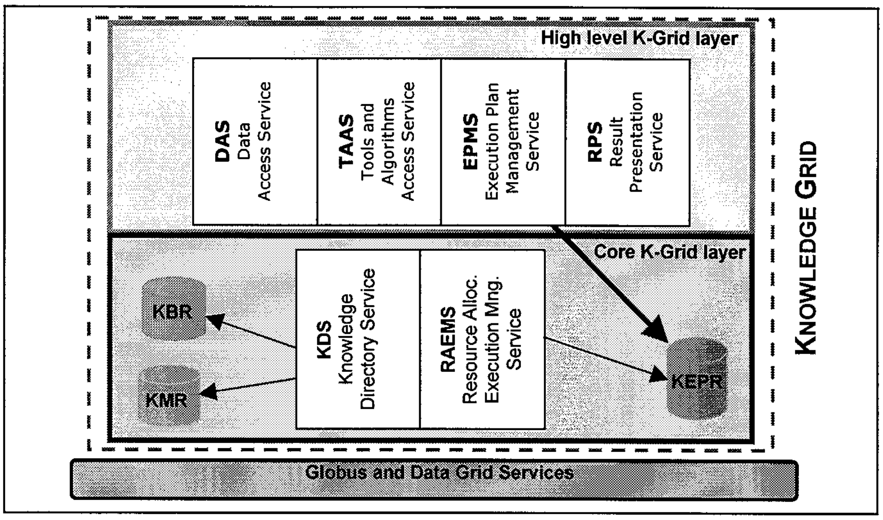
High level K-Grid layer
DAS Data Access Service
TAAS Tools and Algorithms Access Service
EPMS Execution Plan Management Service
RPS Result Presentation Service
Core K-Grid layer
KBR
KMR
KDS Knowledge Directory Service
RAEMS Resource Alloc. Execution Mng. Service
KEPR
KNOWLEDGE GRID
Globus and Data Grid Services

Figure 1: The KNOWLEDGE GRID architecture.

The high level K-grid layer comprises the services used to build and execute PDKD computations over the grid. The *Data Access Service* (*DAS*) is used for the search, selection, extraction, transformation and delivery of data to be mined. The *Tools and Algorithms Access Service* (*TAAS*) is responsible for search, selection, and download of data mining tools and algorithms. The *Execution Plan Management Service* (*EPMS*) is used to generate a set of different possible execution plans, starting from the data and the programs selected by the user. Execution plans are stored in the KEPR to allow the implementation of iterative knowledge discovery processes, e.g., periodical analysis of the same data sources varying in time. The *Results Presentation Service* (*RPS*) specifies how to generate, present and visualise the PDKD results (rules, associations, models, classification, etc.), and offers methods to store in different formats these results in the KBR.

3. Task building and execution process

The design and the execution of computations on the KNOWLEDGE GRID is performed as showed in Figure 2. The operations start searching data and programs to be used in the data mining process.

The search of resources is accomplished by means of the DAS and TAAS tools, analyzing the XML metadata documents stored into the KMR of the participant grid nodes. Such analysis attempts to find specific information about useful resources (e.g., a desired software, datasets regarding a specific argument, etc.), and is conducted on the basis of the search parameters and selection filters chosen by the user. The useful metadata (i.e., those satisfying the searching and filtering criteria) are then stored into the *Task Metadata Repository* (*TMR*), a local storage space that contains information about resources (nodes, data sources and algorithms) selected to perform a computation. The TMR is organized as a set of directories: each one is named with the fully qualified hostname of a grid node, and contains metadata files about resources of that node.

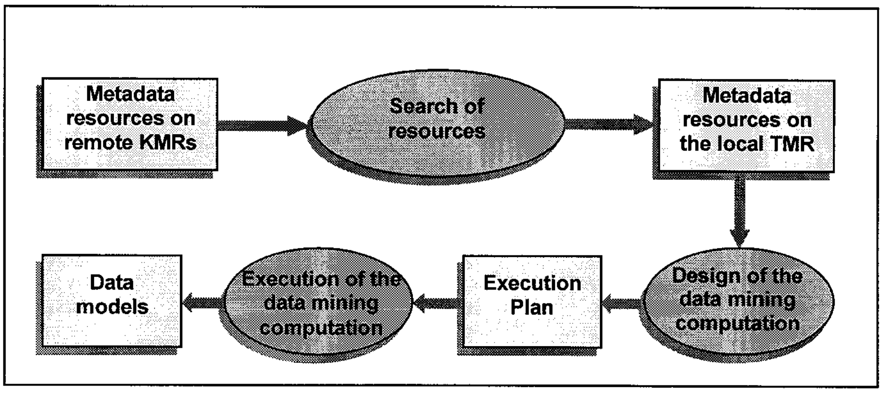
Metadata resources on remote KMRs
Search of resources
Metadata resources on the local TMR
Design of the data mining computation
Execution Plan
Execution of the data mining computation
Data models

Figure 2: Design and execution steps of an application on the KNOWLEDGE GRID.

The design of the computation is performed by means of the EPMS tools, which permits to generate an execution plan for the planned computation. The execution plan (an XML document) can be stored into the KEPR, and processed for its execution by the RAEMS tools. The RAEMS uses in turn the services provided by the resource allocation manager of the underlying grid. After the execution the results are stored in the KBR. A user can visualise and analyse such results using the RPS tools. In the next section we describe in detail an environment integrating functionalities of both EPMS and RAEMS services.

4. Visual Environment for Grid Applications

In order to allow a KNOWLEDGE GRID user to develop and execute applications in a simple and useful manner, we developed an integrated environment named *VEGA* (*Visual Environment for Grid Applications*), which software architecture is depicted in Figure 3. VEGA includes a set of tools, allowing to perform the following operations:

- *task composing*, i.e., definition of the entities involved in the computation and specification of the relations among them;
- *checking* of the consistency of the planned task;
- *generation* of the execution plan for the task;
- *execution* of the generated execution plan through the resource allocation manager of the underlying grid.

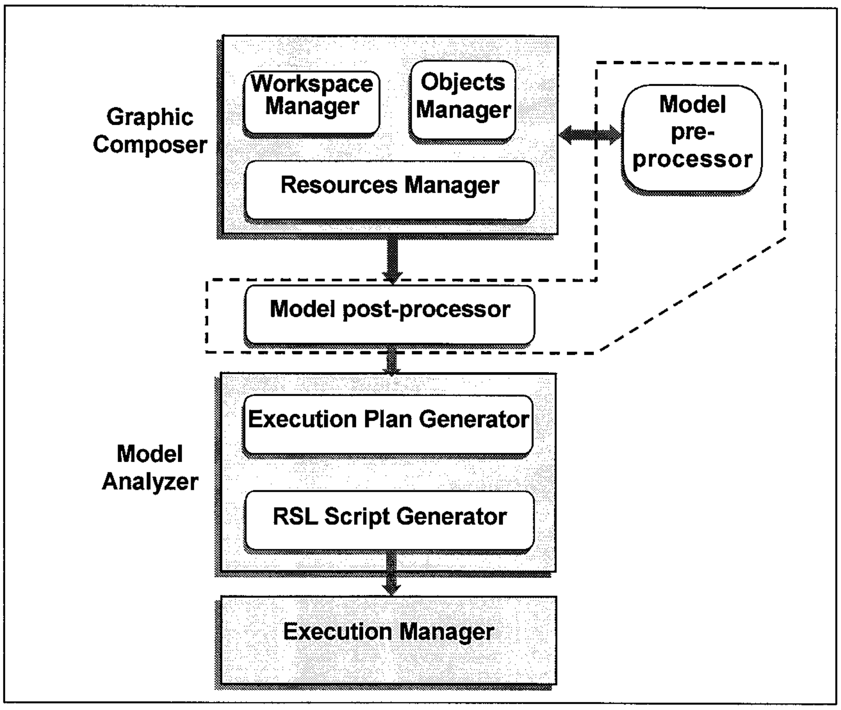

Figure 3: The VEGA software modules.

4.1 Task composing

The task composing phase is performed by means of a graphical interface (see Figure 4), which provides the user with a set of graphical objects representing the resources (datasets, data mining tools, grid nodes). This objects can be composed using visual facilities which consent to insert links among them, forming a graphical representation of the computation. In particular, such phase is realized by the following software components:

- *Workspace Manager*,
- *Resource Manager*, and
- *Object Manager*.

A complex computation is composed of several jobs. The design environment is organized in *workspaces*. Jobs present in a given workspace are intended to be executed concurrently, whereas workspaces are executed sequentially. To this end is maintained a priority relationship between the workspaces which reflects the order of their creation. In addition, the Workspace Manager manages an internal model of the graphical representation showed to the user. Since the set of workspaces represents a unique logical computation, the Workspace Manager must handle with the case in which a task in a given workspace needs to operate on resources generated by tasks in previous workspaces. Such resources are not physically generated by a given workspace in the moment in which the user start to compose the next workspace of the same computation, because all the workspaces are processed for the execution only at the end of the design session. Thus, the Workspace Manager recognizes such a situation during the composition of a workspace, generates the needed virtual resources and make them available through the Resource Manager to all the following workspaces. For instance, if in the *workspace 1* a software component *S* is transferred to a node *N*, a new metadata document is created for *S* and stored in the directory *N* of the TMR. That document that specifies the new location of *S* is marked as temporary until the data transfer is performed. However, if a *workspace 2* is opened in the same session (i.e. it is scheduled after workspace 1), the software *S* is displayed as already available under the resources of *N*. The Workspace Manager allows also to store a graphical composition in a binary file, which can be next retrieved for modifications by the user.

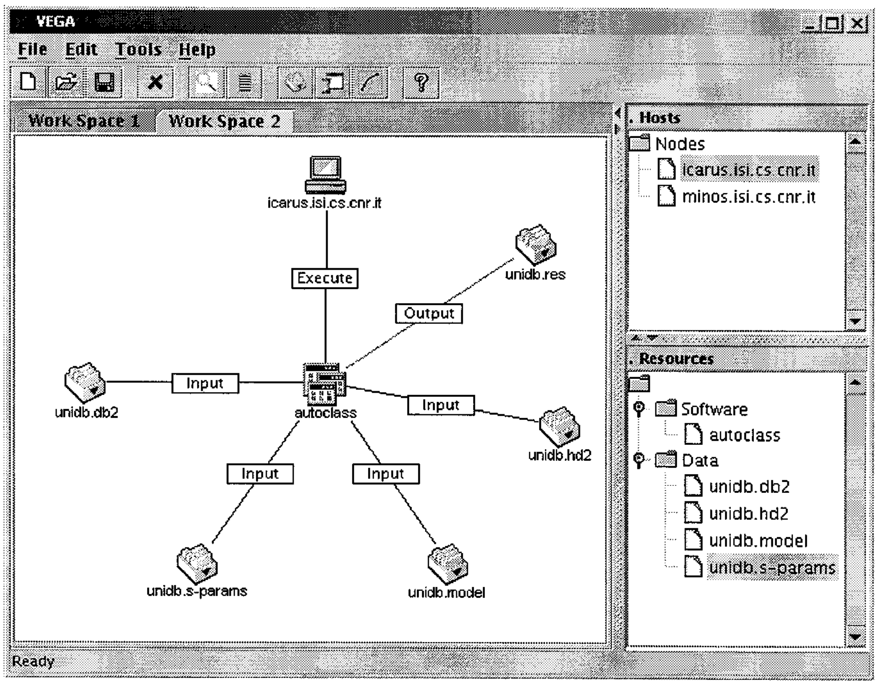

Figure 4: The VEGA user interface

The Resource Manager permits to browse the TMR in order to search and choose the resources to be used in the computation. Selected hosts are displayed into the *Hosts* panel, and the user can explore resources of each one by clicking on its label. That resources are displayed by categories into the *Resources* panel.

The Object Manager deals with the graphical objects during the realization of the visual composition. Each graphical object is associated with information about the related resources; such information is used for the creation of the internal model and for the execution plan generation. The Object Manager handles three kind of objects: data, software and hosts. It allows the user to drag the objects presented in the *Hosts* and *Resources* panels into the active workspace (on the right of Figure 4). After this, the user can link that objects in order to indicate the interaction between them. During the composition phase the objects can be involved in several operations, such insertion and movement in a workspace, selection, linking with other objects, etc. Links can represent different actions, such as data transfer, programs execution and input and output relations. The Object Manager performs the labeling of the links and the attribution of the others properties characterizing them. The *data transfer* link is used to move resources among different locations of the grid. The *execute* link is used to run an application on a grid host, the *input* and *output* links are used to respectively indicate input and output of a program. For each link type is possible to set related parameters (e.g., protocol and destination path of the data transfer, job-manager of the execution, etc.).

4.2 Task consistency checking

The goal of this phase is to obtain a correct and consistent model of the computation. The validation process is performed by means of two components:

- *Model pre-processor* and
- *Model post-processor*.

The pre-processing of the computation model takes place during the graphical composition. The Model pre-processor verifies the composition consistency, allowing, with a context-sensitive control, to create links only if they represent actions that can be effectively executed. For instance, it allows to insert only an input or output link between a software object and a data object, but it does not allow to insert an execution link between a host object and a data object.

The checking is completed by the Model post-processor, which is responsible to catch error occurrences that cannot be recognized during the pre-processing phase. For example, it indicates if the graphical composition in a workspace does not contain at least one host.

4.3 Execution plan generation

In this phase the computation model is translated into a generic execution plan (represented by an XML document), and/or into an RSL script. There are two software modules that accomplish these tasks:

- *XML Generator* and
- *RSL Generator*

Basically, the XML Generator is a parser that analyses the computation model produced during the graphical composition, and is able to generate its equivalent XML representation. When invoked, the XML Generator performs its goal taking into account the properties of the involved resources and the parameters of the links. The XML execution plan describes a data mining computation at a high level, neither containing physical information about resources (which are identified by metadata references), nor about status and current availability of such resources. In fact, specific information about the involved resources will be included in the RSL generation phase, when the computation model is translated in this language. Figure 5 shows an excerpt of a sample execution plan.

```
<ExecutionPlan>
 ...
 <Task ep:label="ws1_dt4">
  <DataTransfer>
   <Source ep:href="minos../unidb_db2.xml"
           ep:title="unidb.db2 on minos.isi.cs.cnr.it"/>
   <Destination ep:href="icarus../unidb_db2.xml"
                ep:title="unidb.db2 on icarus.isi.cs.cnr.it"/>
   ...
  </DataTransfer>
 </Task>
 ...
 <Task ep:label="ws2_c1">
  <Computation>
   <Program ep:href="icarus../autoclass3-3-3.xml"
            ep:title="autoclass on icarus.isi.cs.cnr.it"/>
   <Input ep:href="icarus../unidb_db2.xml"
          ep:title="unidb.db2 on icarus.isi.cs.cnr.it"/>
   ...
   <Output ep:href="icarus../classes.xml"
           ep:title="Classes on icarus.isi.cs.cnr.it"/>
  </Computation>
 </Task>
 ...
 <TaskLink ep:from="ws1_dt4" ep:to="ws2_c1"/>
 ...
</ExecutionPlan>
```

Figure 5: The extract of a sample execution plan.

The execution plan gives a list of tasks and task links, which are specified using respectively the XML tags `Task` and `TaskLink`. The `label` attribute for `Task` element identifies each basic task in the execution plan, and is used in linking various basic tasks to form the overall task flow. Each `Task` element contains a task-specific sub-element, which indicates the parameters of the particular represented task. For instance, the task identified by the `ws1_dt4` label contains a `DataTransfer` element, indicating that it is a data transfer task. The `DataTransfer` element specifies `Source` and `Destination` of the data transfer. The `href` attributes of such elements specify the location of metadata about source and destination objects. In this example, metadata about source of data transfer in the `ws1_dt4` task are provided by the `unidb_db2.xml` file stored in the directory named `minos.isi.cs.cnr.it` of the TMR, whereas metadata about destination are provided by the `unidb_db2.xml` file stored in the directory named `icarus.isi.cs.cnr.it` of the same TMR. The first of such XML documents provides metadata about the `unidb.db2` data set when stored on `minos.isi.cs.cnr.it`, whereas the second one provides metadata about `unidb.db2` when, after the data transfer, is stored on `icarus.isi.cs.cnr.it`. The `TaskLink` elements represent the relations among tasks of the execution plan. For instance, the showed `TaskLink` indicates that the task flow proceeds from the task `ws1_dt4` to the task `ws2_c1`, as specified by its `from` and `to` attributes.

The RSL Generator produces an RSL script that can be directly submitted to the GRAM (Globus Resource Allocation Manager) of a grid node running Globus. The RSL (Resource Specification Language) is a structured language by which resource requirements and parameters can be outlined by a user [3]. In opposition with the XML execution plan, the RSL script describes entirely an instance of the designed computation, i.e., it specifies all the physical information needed for the execution (e.g., name and location of resources, software parameter, etc.). Figure 6 shows an extract of a sample RSL script.

```
+
...
(&(resourceManagerContact=minos.isi.cs.cnr.it)
  (subjobStartType=strict-barrier)
  (label=ws1_dt4)
  (executable=$(GLOBUS_LOCATION)/bin/globus-url-copy)
  (arguments=-vb -notpt gsiftp://minos.isi.cs.cnr.it/.../unidb.db2
              gsiftp://icarus.isi.cs.cnr.it/.../unidb.db2
   )
)
...
(&(resourceManagerContact=icarus.isi.cs.cnr.it)
  (subjobStartType=strict-barrier)
  (label=ws2_c1)
  (executable=.../autoclass)
  (arguments=-search .../unidb.db2  .../unidb.hd2 .../unidb.model
              ...
   )
)
...
```

Figure 6: The extract of a sample RSL script.

4.4 Execution of the computation

The execution of the computation is performed by means of the Execution Manager module. The Execution Manager allows the system to authenticate a user to the grid, by using the Globus GSI (Grid Security Infrastructure) services, and submit the RSL script to the Globus GRAM for its execution. The Execution

Manager is also responsible of the monitoring of the jobs that compose the overall data mining computation during their life cycle.

5. Conclusions

The deluge of data available today in several formats requires intelligent and efficient tools to analyze them and extract models that are useful and undertandable. In the latest years several efforts have been devoted to the design and implementation of parallel and distributed data mining systems that can speed up the knowledge discovery process on large and/or distributed data sets [4]. Among different parallel and distributed computing paradigms, the grid is emerging as a high-performance and higly decentralized infrastructure. Traditional and novel applications can benefit from the use of computational grids as a distributed platform for supporting complex applications.

In this paper we presented the VEGA toolset for the composition of data mining applications on the KNOWLEDGE GRID environment. We discussed how such toolset allows a user to build the computation starting from a set of useful remote resources such as computational nodes, sources of data, and data mining algorithms. Such resources, which are located and selected by means of grid information services, are presented to the user as a set of objects. The user can compose those objects using common visual facilities, to form a graphic representation of her/his data mining computations. The toolset validates and translates this graphic representation into an execution plan, which is then processed and effectively executed on the grid by means of Globus resource management tools.

References

[1] Foster, I. & Kesselman, C. Globus: a metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications*, **11**, pp. 115-128, 1997.
[2] Cannataro, M., Talia, D., & Trunfio, P. KNOWLEDGE GRID: High Performance Knowledge Discovery Services on the Grid. *Proc. GRID 2001*, LNCS, pp. 38-50, Springer-Verlag, 2001.
[3] The Globus Project, The Globus Resource Specification Language RSL v1.0, available at http://www.globus.org/gram/rsl_spec1.html
[4] Kargupta, H. & Chan P. (eds). *Advances in Distributed and Parallel Knowledge Discovery*, AAAI/MIT Press, 2000.