Distributed Data Mining on Grids: Services, Tools, and Applications

Mario Cannataro, *Associate Member, IEEE*, Antonio Congiusta, *Associate Member, IEEE*, Andrea Pugliese, *Student Member, IEEE*, Domenico Talia, *Associate Member, IEEE*, and Paolo Trunfio, *Student Member, IEEE*

***Abstract*—Data mining algorithms are widely used today for the analysis of large corporate and scientific datasets stored in databases and data archives. Industry, science, and commerce fields often need to analyze very large datasets maintained over geographically distributed sites by using the computational power of distributed and parallel systems. The grid can play a significant role in providing an effective computational support for distributed knowledge discovery applications. For the development of data mining applications on grids we designed a system called *KNOWLEDGE GRID*. This paper describes the *KNOWLEDGE GRID* framework and presents the toolset provided by the *KNOWLEDGE GRID* for implementing distributed knowledge discovery. The paper discusses how to design and implement data mining applications by using the *KNOWLEDGE GRID* tools starting from searching grid resources, composing software and data components, and executing the resulting data mining process on a grid. Some performance results are also discussed.**

***Index Terms*—Grid computing, grid programming, grid scheduling, knowledge grid, data mining.**

I. Introduction

TODAY large amounts of data are collected and warehoused. Data sets are generated and stored at enormous speed in local databases, from remote sources or from the sky. At the same time, scientific simulations generating terabytes of data are performed in many laboratories. E-commerce and e-business applications store and manage huge databases about products, clients and transactions.

Unfortunately, we are much better at storing data than extracting knowledge from it. Large datasets are hard to understand and traditional techniques are infeasible for raw data. Data mining helps scientists in hypothesis formation in biology, medicine, physics, and engineering. Companies use data mining techniques to provide better, customized services and support decision making. In all these different areas, massive data collections of terabyte and petabyte scale need to be used and analyzed. Moreover, in many cases datasets must be shared by large communities of users that pool their resources from different sites belonging to a single company, or from a large number of laboratories, plants, or public organizations.

Grid computing has been proposed as a novel computational model, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. Today grids can be used as effective infrastructures for distributed high-performance computing and data processing [1]. A grid is a geographically distributed computation infrastructure composed of a set of heterogeneous machines that users can access via a single interface. Grids therefore, provide common resource-access technology and operational services across widely distributed *virtual organizations* composed of institutions or individuals that share resources.

Although originally intended for advanced science and engineering applications, grid computing has emerged as a paradigm for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations in industry and business [2]. Grid applications include the following:

- intensive simulations on remote supercomputers;
- cooperative visualization of very large scientific data sets;
- distributed processing for computationally demanding data analysis;
- coupling of scientific instruments with remote computers and data archives.

In the last five years, toolkits and software environments for implementing grid applications have become available. These include Legion [3], Condor [4], and Unicore [5]. In particular, Foster and Kesselman's Globus Toolkit [6] is the most widely used middleware in scientific and data-intensive grid applications, and is becoming a de facto standard for implementing grid systems. The toolkit addresses security, information discovery, resource and data management, communication, fault-detection, and portability issues. It does so through mechanisms, composed as bags of services, that execute operations in grid applications. Today, Globus and the other grid tools are used in many projects worldwide. Although most of these projects are in scientific and technical computing, there is a growing number of grid projects in education, industry, and commerce.

Together with the grid shift toward industry and business applications, a parallel shift toward the implementation of data grids has been registered. Data grids are designed to allow large data sets to be stored in repositories and moved with almost the same ease that small files can be moved. They represent an enhancement of computational grids, driven by the need to handle

Manuscript received December 29, 2002; revised October 11, 2003. This work was supported in part by the Italian FIRB Project "GRID.IT" under Grant RBNE01KNFP. This paper was recommended by Guest Editors H. Kargupta, S. Bandyopadhyay, and B.-H. Park.

M. Cannataro is with the Università di Catanzaro, 88100 Catanzaro, Italy. (e-mail: cannataro@unicz.it).

A. Congiusta, A. Pugliese, D. Talia, and P. Trunfio are with the DEIS, Università della Calabria, 87036 Rende (CS), Italy (e-mail: apugliese@si.deis.unical.it; congiusta@si.deis.unical.it; talia@si.deis.unical.it; trunfio@si.deis.unical.it).

Digital Object Identifier 10.1109/TSMCB.2004.836890

large data sets without repeated authentication, aiming to support the implementation of distributed data-intensive applications. Significant examples are the EU DataGrid [7], the Particle Physics Data Grid [8], the Japanese Grid DataFarm [9], and the Globus Data Grid [10] project.

Data grid middleware is central for management of data movement and replication on grids. Furthermore, in many scientific and business areas it is necessary to use tools and environments for analysis, inference and discovery over the available data. Scientists and engineers can use those environments for implementing grid-based problem solving environments for doing "virtual" scientific experiments. Analysts can follow the same approach in mining large volumes of data to support decision making. Therefore, the evolution of data grids is represented by knowledge grids offering high-level tools and models for the distributed mining and extraction of knowledge from data repositories available on the grid [11]. The development of such an infrastructure is the main goal of our research work, focused on the design and implementation of an environment for geographically distributed high-performance knowledge discovery applications called *KNOWLEDGE GRID*.

The *KNOWLEDGE GRID* is a parallel and distributed software architecture that integrates data mining techniques and grid technologies. In the *KNOWLEDGE GRID* architecture data mining tools are integrated with generic and data grid mechanisms and services. Thus the *KNOWLEDGE GRID* can be exploited to perform data mining on very large data sets available over grids, to make scientific discoveries, improve industrial processes and organization models, and uncover business valuable information.

In [12] some of us presented the system requirements and the software architecture of the *KNOWLEDGE GRID* and [13] describes a visual toolset for developing data mining applications on the *KNOWLEDGE GRID*. This paper includes a more detailed introduction to the system properties, discusses the design and execution process of applications on the *KNOWLEDGE GRID*, and presents performance results achieved running a real distributed data mining application on a Globus-based grid.

The outline of the paper is as follows. Section II briefly describes the components of the *KNOWLEDGE GRID* architecture and its main features. Sections III and IV discuss how the tools of the *KNOWLEDGE GRID* offer knowledge discovery services for designing, building, and executing distributed data mining applications. Section V presents some experimental results. Section VI discusses related work and Section VII concludes the paper.

II. KNOWLEDGE GRID

The *KNOWLEDGE GRID* architecture uses basic grid mechanisms to build specific knowledge discovery services on top of grid toolkits and services. These services can be developed in different ways using the available grid environments. The current implementation is based on the Globus Toolkit [14]. Like Globus, the *KNOWLEDGE GRID* offers global services based on the cooperation and combination of local services. We designed the *KNOWLEDGE GRID* architecture so that more specialized data mining tools are compatible with lower-level grid mechanisms and data grid services. This approach benefits from "standard" Grid services that are more and more utilized and offers an open parallel and distributed knowledge discovery architecture that can be configured on top of grid middleware in a simple way.

A. Globus Toolkit Services

The main services offered by Globus Toolkit 2 are the following:

- *Grid security infrastructure* (*GSI*). Enables secure authentication and communication over an open network providing a number of services, including mutual authentication and single sign-on run-anywhere authentication, with support for local control over access rights and mapping from global to local user identities [15]. GSI is based on public key encryption, X.509 certificates, and the *secure sockets layer* (*SSL*) communication protocol.
- *Monitoring and discovery service* (*MDS*). Provides a framework for publishing and accessing information about grid resources [16] by using the *lightweight directory access protocol* (*LDAP*) as a uniform interface to such information. MDS provides two types of directory services: the *grid resource information service* (*GRIS*) and the *grid index information service* (*GIIS*). A GRIS can answer queries about the resources of a particular grid node; examples of information provided include host identity (e.g., operating systems and versions), as well as more dynamic information such as current CPU load and memory availability. A GIIS combines the information provided by a set of GRIS services managed by an organization, giving a coherent system image that can be explored or searched by grid applications.
- *Globus resource allocation manager* (*GRAM*). Provides facilities for resource allocation and process creation, monitoring, and management [17]. GRAM simplifies the use of remote systems by providing a single standard interface for requesting and using remote system resources for the execution of jobs. The most common use of GRAM is remote job submission and control, to support distributed computing applications.
- *Dynamically-updated resource online co-allocator* (*DUROC*). Manages multirequests of resources, delivers requests to different GRAMs and provides time-barrier mechanisms among jobs [18]. In Globus, a GRAM provides an interface to submit jobs on a particular set of physical resources, whereas the DUROC is used to coordinate transactions with independent GRAMs.
- *Heartbeat monitor* (*HBM*). Provides a mechanism for monitoring the state of processes [19]. The HBM is designed to detect and report the failure of processes that have identified themselves to the HBM. It allows simultaneous monitoring of both Globus system processes and application processes associated with user computations. The HBM also provides notification of process status exception events, so that recovery actions can be taken.
- *GridFTP*. Implements a high-performance, secure data transfer mechanism based on an extension of the FTP protocol that allows parallel data transfer, partial file transfer,

and third-party (server-to-server) data transfer, using GSI for authentication [20]. This allows grid applications to have ubiquitous, high-performance access to data in a way that is compatible with the most popular file transfer protocol in use today.

- *Replica catalog* and *replica management*. Provide facilities for managing data replicas, i.e., multiple copies of data stored in different systems to improve access across geographically-distributed grids. The replica catalog provides mappings between logical names for files and one or more copies of the files on physical storage systems; it is accessible via an associated library and a command-line tool [21]. The replica management combines the replica catalog (for keeping track of replicated files) and GridFTP (for moving data) to manage data replication [22].

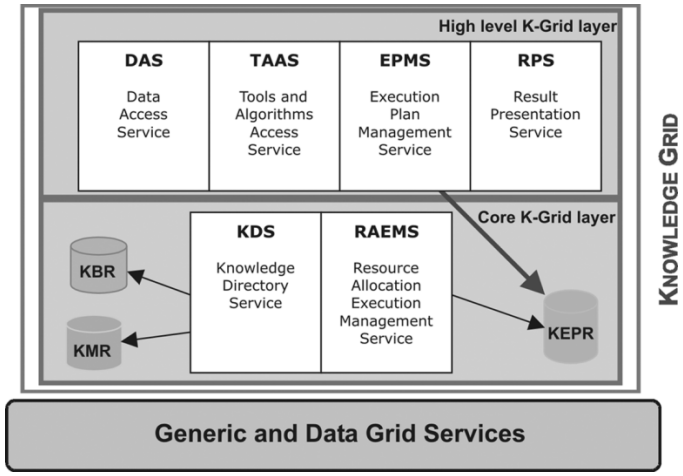

Fig. 1. *KNOWLEDGE GRID* architecture.

B. Knowledge Grid Services

The *KNOWLEDGE GRID* is composed of two hierarchic levels: the *Core K-Grid* layer and the *High level K-Grid* layer. The former refers to services directly implemented on top of generic grid services, while the latter is used to design, compose, and execute distributed knowledge discovery computations over the *KNOWLEDGE GRID*. Fig. 1 shows the layers and their components together with the *KNOWLEDGE GRID* data and metadata repositories. In the following, the term *K-Grid node* denotes a grid node implementing the *KNOWLEDGE GRID* services.

1) Core K-Grid Layer: The core K-Grid layer implements the basic services for the definition, composition and execution of a distributed knowledge discovery application over the grid. Its main goals are the management of metadata describing features of data sources, third party data mining tools, data management, and data visualization tools and algorithms. Moreover, this layer coordinates the application execution by attempting to fulfill the application requirements on the available grid resources.

The Core K-Grid layer comprises two main services

- The *Knowledge Directory Service* (*KDS*) extends the basic globus monitoring and discovery service and manages metadata describing data and tools used in the *KNOWLEDGE GRID*. These include:
  - Repositories of data to be mined (data sources).
  - Tools and algorithms used to extract, filter and manipulate data; tools to mine data and visualize and store mining results.
  - Distributed *execution plans*. An execution plan is an abstract description of a distributed data mining application, that is a graph describing the interaction and data flow between data sources, data mining tools, visualization tools, and result storage facilities.
  - Knowledge obtained as result of the mining process, i.e., learned models and discovered patterns.

  All metadata are represented in eXtensible Markup Language (XML) documents and stored in a *knowledge metadata repository (KMR)*.

  Whereas, it would be infeasible to maintain the data to be mined in an ad hoc repository, it could be useful to maintain a repository of the discovered knowledge. This information is therefore stored in a *knowledge base repository* (*KBR*), and the associated metadata are managed by the KDS. The KDS is thus used not only to search and access raw data, but also to find previously discovered knowledge that can be used to compare the output of a given mining computation with different data sources, or to apply data mining tools in an incremental way.

  Data management, analysis and visualization tools are usually pre-existent to the *KNOWLEDGE GRID* (i.e., they reside into file systems or code libraries). Finally, the *knowledge execution plan repository* (*KEPR*) stores the execution plans of data mining processes.
- The *resource allocation and execution management service* (*RAEMS*) is used to find a suitable mapping between an execution plan and the available resources, with the goal of satisfying application requirements (computing power, storage, memory, database, compiler, network bandwidth and latency) and grid constraints. After the execution plan activation, this service manages and coordinates the application execution. In the current *KNOWLEDGE GRID* implementation, instead of using the KDS and the Globus MDS services, this layer is directly based on the Globus resource allocation manager (GRAM) services. Resource requests of each data mining job are expressed using the Globus Resource Specification Language (RSL) [23]. The analysis and processing of the execution plan will generate global resource requests that in turn are translated into RSL requests for local GRAMs.

2) High Level K-Grid Layer: The high-level K-Grid layer includes services used to compose, validate, and execute a parallel and distributed knowledge discovery computation. Moreover, the layer offers services to store and analyze the discovered knowledge. The main services are the following:

- *Data access service* (*DAS*)

  The data access service is responsible for searching, selecting, extracting, transforming, and delivering data to be mined. Search and selection are based on the core KDS

service. On the basis of the user requirements and constraints, the data access service automates (or assists the user in) searching and finding data sources to be analyzed by DM tools.

- *Tools and algorithms access service* (*TAAS*)

  This service is responsible for searching, selecting, and downloading data mining tools and algorithms. As before, the metadata regarding their availability, location, and configuration are stored in the KMR and managed by the KDS, whereas the tools and algorithms are stored in the local storage facility of each K-Grid node. A node wishing to export data mining tools to other users has to publish them using the KDS services, which store the metadata in the local portion of the KMR.
- *Execution plan management service* (*EPMS*)

  An execution plan is represented by a graph describing the interaction and data flows among resources. In simple cases, a user can directly design the execution plan by using a visual composition tool where programs are connected to data sources. However, due to the variety of results produced by DAS and TAAS, different execution plans can be yielded, in terms of data and tools location, strategies to move or stage intermediate results, and so on. Thus, the execution plan management service is implemented by a semi-automatic tool that takes data and programs selected by the user, and generates an abstract execution plan describing the designed computation to be mapped onto concrete grid resources (see Section III-D). Execution plans are stored in the knowledge execution plan repository (KEPR).
- *Results Presentation Service* (*RPS*)

  Result visualization is important in the knowledge discovery process to help users in the interpretation of the discovered patterns. This service specifies how to generate, present and visualize the knowledge models extracted (e.g., association rules, clustering models, classification models), after storing them in the Knowledge Base Repository. The result metadata are stored in the KMR to be managed by the KDS.

III. Design of Grid Data Mining Applications

Fig. 2 shows the steps of the design process of distributed knowledge discovery applications on the *Knowledge Grid*. The design process starts by searching and selecting the resources needed to compose the application. This step is accomplished by means of DAS and TAAS tools that analyze the XML metadata documents representing the available resources of the participant K-Grid nodes, stored into their KMRs. Such analysis attempts to find specific information about useful resources (e.g., software implementing a specific data mining algorithm, particular data sources, etc.). It is performed on the basis of search parameters and selection filters chosen by the user. Metadata about the resources selected for the computation are then stored into the *task metadata repository* (*TMR*), a local storage space that contains information about resources (computational nodes, data sources and software) selected to perform a computation.

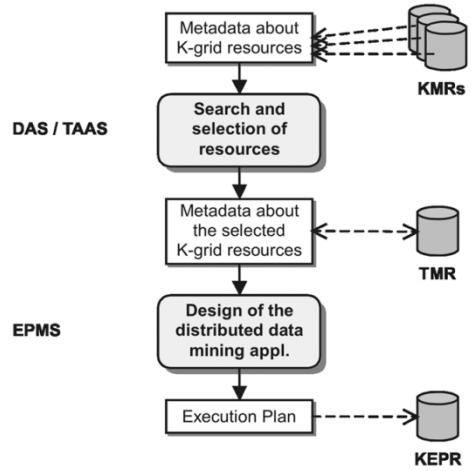

Fig. 2. Design process of a data mining computation.

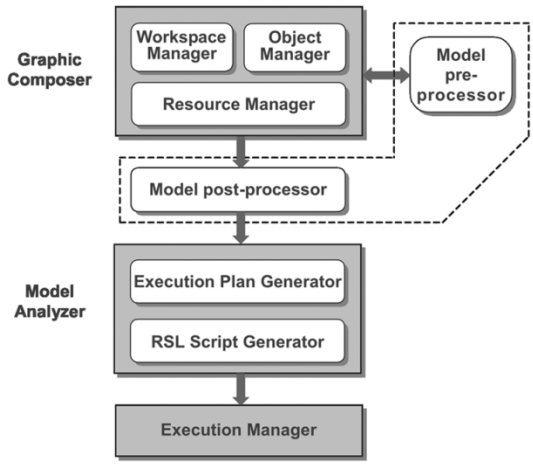

Fig. 3. VEGA software modules.

The design of a data mining computation is performed by means of the EPMS. For allowing a user to build the computation in a simple way, we developed a toolset named visual environment for grid applications (VEGA). The VEGA architecture is shown in Fig. 3. VEGA integrates functionalities of the EPMS and other K-Grid services. In particular, it provides the following EPMS operations:

- *task composition*, i.e., definition of the entities involved in the computation and specification of the relationships among them;
- *consistency checking* of the planned computation;
- *generation* of the execution plan.

A. Task Composition

The task composition phase is performed by means of a graphical interface (see Fig. 4), which provides a user with a set of graphical objects representing the grid nodes and the resources (e.g., data sets, data mining tools) available on them. These objects can be composed through visual facilities that allow a user to insert links among them and produce a graphical representation of the computation.

In particular, such phase is implemented by the following software components:

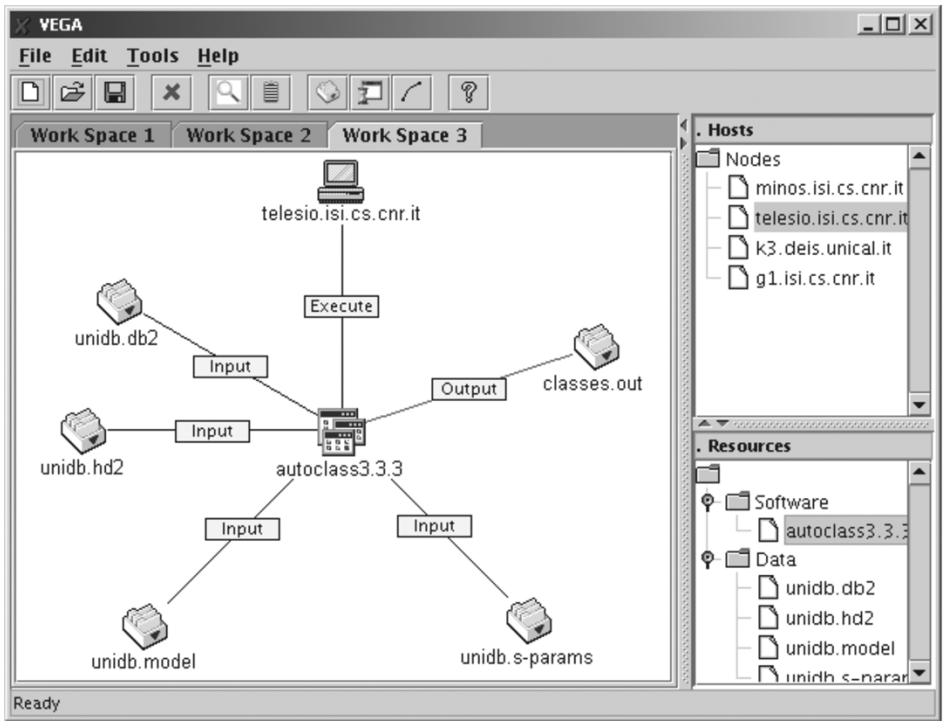

Fig. 4. Visual interface of VEGA.

- *resource manager*;
- *object manager*;
- *workspace manager*.

The resource manager supports the browsing of 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 a user can explore resources of each one by clicking on its label. Resources are displayed, by categories, into the *resources* panel.

The object manager deals with the graphical objects during the visual composition. Each graphical object is associated with information about the related resources; such information is used for the creation of an internal model and for the execution plan generation. The object manager handles three kinds of objects: data, software and hosts. It allows the user to drag the objects presented in the hosts and resources panels (on the right-hand side of Fig. 4) into a *workspace*. Then, a user can link those objects 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, program execution and input and output relationships. The object manager performs the labeling of the links and the attribution of the other associated properties. 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 indicate input and output data of a program. For each link type it is possible to set related parameters (e.g., protocol and destination path of the data transfer, job-manager of the execution, etc.).

A complex computation is composed of several jobs. The design environment is organized in different workspaces. Jobs in a given workspace are intended to be executed concurrently, whereas workspaces are executed sequentially. To this end, a priority relationship between workspaces is maintained.

We describe here an example that shows in detail the task composition process. A user logged on K-Grid node `g1.isi.cs.cnr.it` aims to perform a data mining application on the data set `Unidb`, stored on the same node. The application is composed of two data mining steps: clustering and classification. The data set are to be clustered using three different algorithms, running in parallel on copies of the data set. The clustering results are then analyzed by a classification algorithm that will be executed in parallel on three different nodes, generating three classification models of the same data set. Finally, the three different models will be shown to the user that will select the more accurate ones. The user has located K-Grid nodes `k1.deis.unical.it`, `k2.deis.unical.it`, and `k3.deis.unical.it` offering the clustering algorithms K-Means [24], Intelligent Miner [25] and AutoClass [26], respectively, and node `g2.isi.cs.cnr.it` that offers the C5.0 classifier [27].

Figs. 5–8 show the sequence of the four following workspaces composed by the user to design such computation:

- *Workspace 1* (Fig. 5). The dataset `Unidb` (which is located on node `g1`) and the classifier C5.0 (which is located on `g2`) are copied to nodes `k1`, `k2`, and `k3`.
- *Workspace 2* (Fig. 6). On node `k1`, dataset `Unidb` is analyzed by K-Means producing the output `K-Means.out`; on `k2`, dataset `Unidb` is analyzed by IMiner that produces `Iminer.out`; on `k3`, dataset `Unidb` is analyzed by AutoClass producing its results as `AutoClass.out`.
- *Workspace 3* (Fig. 7). On node `k1`, `K-Means.out` is analyzed by C5.0 producing the output `K-Means_c5.out`; on `k2`, `IMiner.out` is analyzed by C5.0 producing

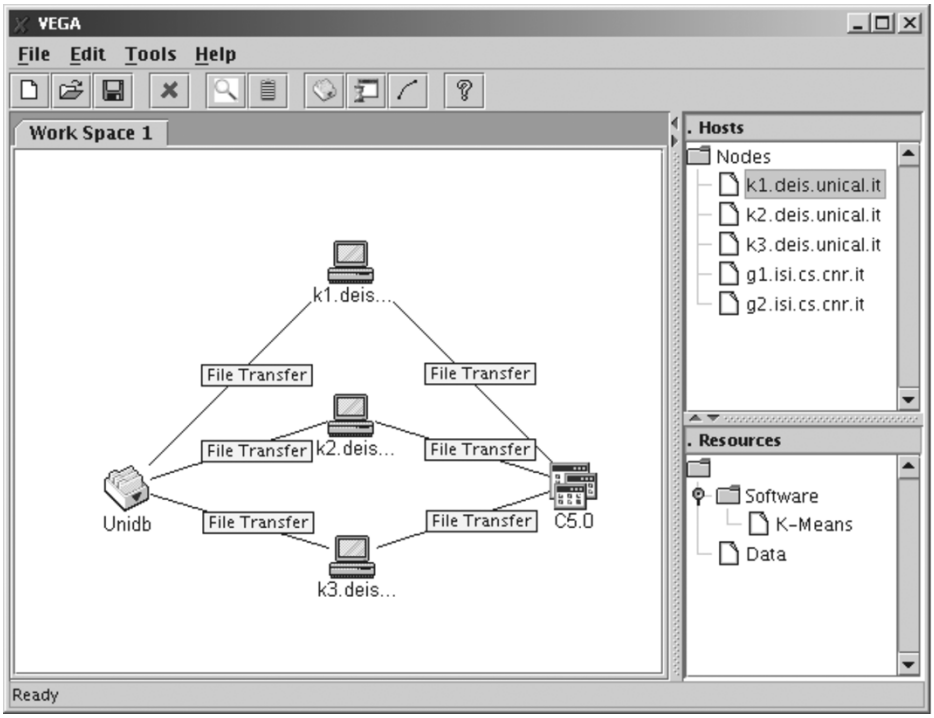

Fig. 5. Workspace 1.

`Iminer_c5.out`; on `k3`, `AutoClass.out` is analyzed by C5.0 producing `AutoClass_c5.out`.

- *Workspace 4* (Fig. 8). The results `K-Means_c5.out`, `IMiner_c5.out` and `AutoClass_c5.out` are moved from `k1`, `k2` and `k3` to `g1`.

Since the set of workspaces represents a unique logical computation, the workspace manager must handle 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 available when a user starts to compose a subsequent workspace, because all the workspaces are processed for the execution only at the end of the design phase.

The workspace manager recognizes such a situation during the composition of a workspace, and generates and makes available the needed *virtual resources* to the successive workspaces. For instance, in *workspace 1* (Fig. 5) the dataset `Unidb` is copied to node `k1.deis.unical.it`, then a new metadata document is created for `Unidb` and stored in the `k1.deis.unical.it` portion of the TMR. That document is marked as temporary until the data transfer is actually performed. However, in *workspace 2* (Fig. 6), the dataset `Unidb` is displayed as already available under the resources of `k1.deis.unical.it`.

B. 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: the *model preprocessor* and the *model postprocessor*.

The preprocessing of the computation model takes place during the graphical composition. The model preprocessor checks the consistency of composition, allowing, with a context-sensitive control, to create links only if they represent actions that can be actually executed. For instance, it allows the user to insert an input or output link only 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 for catching errors that cannot be recognized during the preprocessing phase. For example, it verifies if a workspace contains at least one host.

C. Execution Plan Generation

In this phase the computation model is translated into an execution plan represented by an XML document. This task is performed by the *execution plan generator*.

Basically, the execution plan generator is a parser that analyzes the computation model produced during the graphical composition, and generates its equivalent XML representation. When invoked, the execution plan generator performs its task by 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, containing neither 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. Fig. 9 shows an extract of the execution plan for the example described above.

The execution plan gives a list of tasks and task links, which are specified using the XML tags `Task` and `TaskLink`, respectively. The `label` attribute for a `Task` element identifies each basic task in the execution plan, and is used for 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

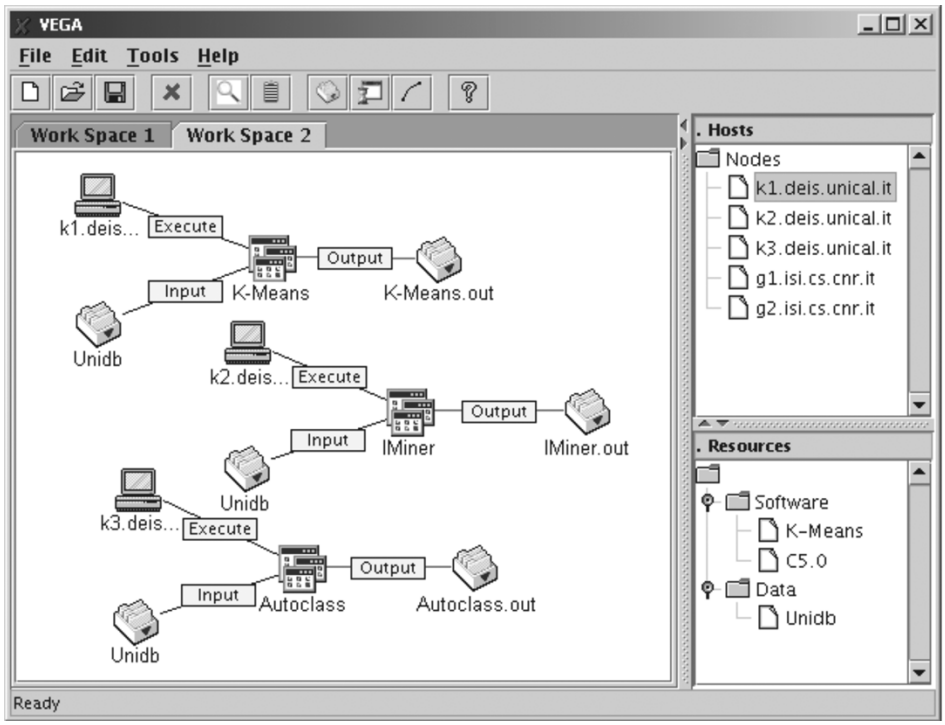

Fig. 6. Workspace 2.

task. For instance, the task identified by the `ws1_dt2` label contains a `DataTransfer` element, indicating that it is a data transfer task. The `DataTransfer` element specifies `Source` and `Destination`. The `href` attributes of such elements specify the location of metadata about source and destination objects.

In this example, metadata about the source of data transfer in the `ws1_dt2` task are provided by the `Unidb.xml` file stored in the directory named `g1.isi.cs.cnr.it` of the TMR, whereas metadata about destination are provided by the `Unidb.xml` file stored in the `k2.deis.unical.it` portion of the same TMR. The first of such XML documents provides metadata about the `Unidb` dataset when stored on `g1.isi.cs.cnr.it`, whereas the second one provides metadata about `Unidb` when, after the data transfer, it is stored on `k2.deis.unical.it`. The `TaskLink` elements represent relationships among tasks of the execution plan. For instance, the shown `TaskLink` indicates that task `ws2_c2` follows `ws1_dt2`, as specified by its `from` and `to` attributes.

The *KNOWLEDGE GRID* also offers the users a set of services for transparent location, retrieval, and access to data sources and software tools on the grid. Such transparency support has a cost, as it involves production, publishing, retrieval, and updating of resource metadata. Moreover, navigating inside metadata requires accessing local or remote repositories, thus resulting in CPU and transmission overheads.

An additional issue concerns *control* transparency, that is the abstraction of data mining applications from the available physical grid resources. Such transparency is achieved in the *KNOWLEDGE GRID* through a precompilation of the execution plans with respect to a set of hosts including also *abstract hosts*. The execution plans are later mapped and scheduled against the available grid resources mapping *abstract hosts* to concrete ones. In Section III-D, we detail our approach to application scheduling.

D. Application-Oriented Scheduling

In real grid applications it is generally infeasible to specify all the application requirements at the time of their composition. As said before, we are currently adding to the *KNOWLEDGE GRID* programming model the possibility to define and use abstract hosts, i.e., hosts whose characteristics are only partially specified, and that can be matched to different concrete ones. The assignment of abstract jobs (i.e., those involving abstract hosts) to concrete hosts is performed by a *scheduler*, which is part of the resource allocation and execution management service. The scheduler's task is to examine execution plans comprising abstract jobs and, on the basis of knowledge or prediction about computational and input/output (I/O) costs, yield *schedules* (i.e., assignments along with timing constraints) with the goal of improving applications' performances. Furthermore, the scheduler is able to adapt generated schedules to new information about job status and available resources.

The scheduler offers an open Java interface allowing the specification of user-defined scheduling policies. Moreover, users can provide their own way to estimate computational and I/O costs, i.e., computation times of software components as a function of input, processing host, and time; communication times for data transfers as a function of source and destination hosts, size of data to be transferred, and time; output sizes as a function of software and input. The currently provided functionalities are the following:

- *Scheduling strategy*. Several strategies are provided (discussed in the remainder).

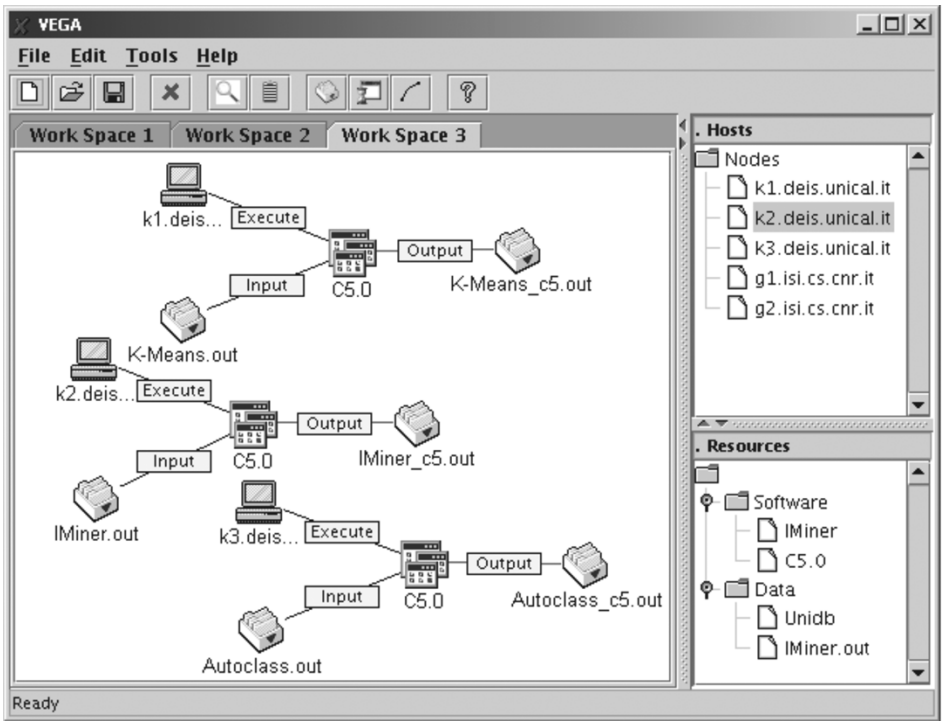

Fig. 7. Workspace 3.

- *Scheduling process*. A dynamic rescheduling scheme is adopted, where schedules are computed initially and then, during applications' execution, they are recomputed as a consequence of the completion of all jobs preceding an unassigned job in the current schedule; important performance variations; job failures.
- *Computational cost estimation*. The scheduler adopts the *Network Weather Service* (*NWS*) [28] as its information source about current and future CPU availability, a *sampling method* for evaluating the processing requirements of software components [29], and based on that it computes the cost formula $\cos t(d,s,h,t) = (req(d,s)/perf(h) \cdot avail(h,t))$ where $req(d,s)$ represents the processing requirements of software s run on data set d (with respect to a reference host), $perf(h)$ is the no-load performance of host h, and $avail(h,t)$ is the fraction of processing cycles reserved on host h at time t.
- *I/O cost estimation*. The scheduler adopts the NWS as well, and directly employs information about bandwidth and latency to build I/O cost estimates.
- *Output size estimation*. The scheduler makes use of user-provided descriptions of the relationships between input and output sizes of software components.

Scheduling strategies are implemented by a *mapper* component. The mapper's input, besides the abstract execution plan and resource descriptions, comprises the three cost estimation functions. The mapper's output consists of an assignment of abstract jobs to hosts, and a timing function associating each job with the time at which it must be started during the application execution. Computed schedules may be partial, i.e., comprise unassigned jobs (called pending), to be scheduled subsequently, but they must meet several strict requirements. First, resource constraints must be satisfied, i.e., each job has to be really executable (taking into account the properties of software, data and host composing it). Precedence constraints must be satisfied as well, i.e., if a job j_i precedes another job j_k in the input plan, then j_k's starting time has to be chosen after j_i's completion (and after other possible data movement operations). Finally, the overall completion time must be as low as possible.

The mapper deals with a very challenging problem. Even if we assume that the mapper is in control of an entire completely-connected resource pool, invariant and composed of resources having identical performances, both w.r.t. processing units and network links [30], the resulting problem (which is a generalization of the *precedence-constrained scheduling* problem) is *NP*-hard if more than one host is considered [31]. Therefore, exact optimal techniques, such as integer-linear or constraint programming, seem not to be usable as they incur in an exponential duration of the scheduling process. More suitable approaches tackle the problem heuristically, and several interesting proposals have appeared, both dealing with sets of independent jobs ([29], [32], [33]) and entire applications ([34]–[36]).

The mapper, before applying its heuristics, performs a preprocessing phase with the objective of reducing the size of the search space. The preprocessing phase comprises the following steps:

1) The abstract plan is reduced by eliminating jobs whose input size is not known neither it can be suitably estimated. This reduction is feasible since we deal with data-intensive applications.
2) The abstract plan is still reduced to comprise only *entry* jobs (i.e., jobs that have all inputs ready at scheduling time) and jobs that depend on them, up to a certain depth p.

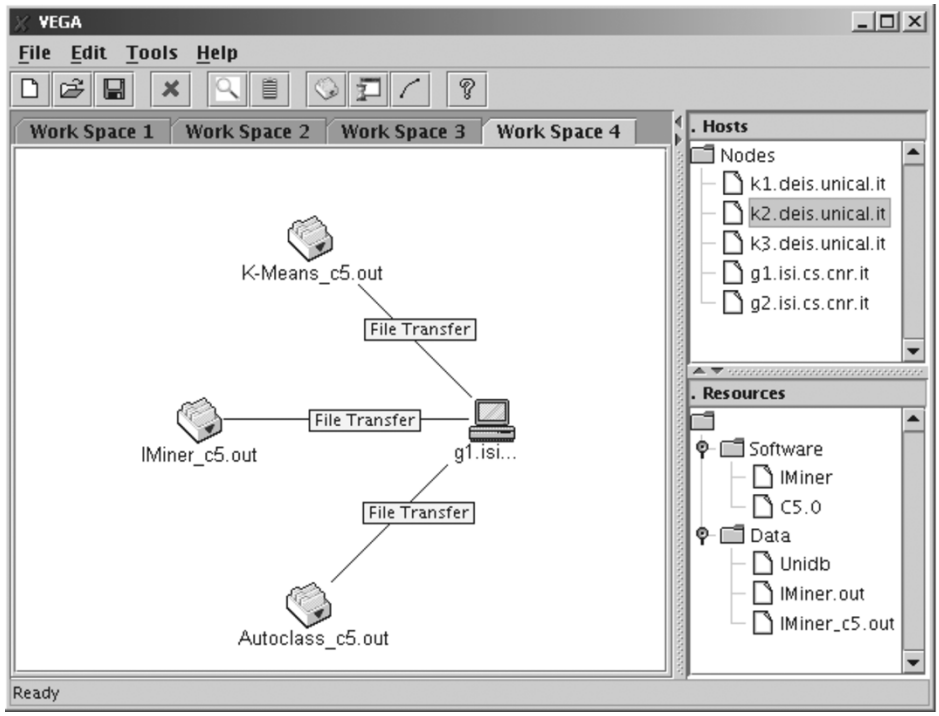

Fig. 8. Workspace 4.

```
<ExecutionPlan>
 ...
 <Task ep:label="ws1_dt2">
  <DataTransfer>
   <Source ep:href="g1../Unidb.xml"
           ep:title="Unidb on g1.isi.cs.cnr.it"/>
   <Destination ep:href="k2../Unidb.xml"
                ep:title="Unidb on k2.deis.unical.it"/>
   ...
  </DataTransfer>
 </Task>
 ...
 <Task ep:label="ws2_c2">
  <Computation>
   <Program ep:href="k2../IMiner.xml"
            ep:title="IMiner on k2.deis.unical.it"/>
   <Input ep:href="k2../Unidb.xml"
          ep:title="Unidb on k2.deis.unical.it"/>
   ...
   <Output ep:href="k2../IMiner.out.xml"
           ep:title="IMiner.out on k2.deis.unical.it"/>
  </Computation>
 </Task>
 ...
 <TaskLink ep:from="ws1_dt2" ep:to="ws2_c2"/>
 ...
</ExecutionPlan>
```

Fig. 9. Extract of an execution plan.

3) In the spirit of [34], [36], groups of "similar" hosts, in terms of computing power and network distance, are formed.
4) Finally, for each abstract job, the set of possible hosts on which that job can be scheduled is built, by looking at resource descriptions, in order to consider only feasible assignments.

After having reduced the search space, the mapper employs one (or more) of the following heuristics:

- *Min-Min*. The Min-Min heuristic at the first step evaluates the earliest completion times of entry jobs over available hosts, and schedules a job to the host that allows the earliest completion time. Then, at each step, the heuristic schedules a job that becomes ready to be run only after the completion of some jobs that have been scheduled at the previous steps (we refer to these as to *ready* jobs). The process terminates when all jobs have been scheduled.
- *Max-Min*. This heuristic works in the same fashion as the Min-Min one, except that at each step it assigns the job to the host that incurs in the maximum completion time.
- *Minimum completion time*. This heuristic works in the same fashion as the Max-Min one, except that it examines only one randomly-chosen job at a time. The job is extracted from the set of entry jobs at the first step, and subsequently from the set of ready jobs.
- *Opportunistic load balancing*. This heuristic assigns each considered job to the first host that becomes idle; the steps are the same as those of MCT.
- *Simulated annealing*. This heuristic is based on the idea of starting from an initial solution, and then performing moves into the search space, from a current solution to one belonging to its "neighborhood." This heuristic allows moves toward worse solutions according to a probability distribution that decreases with time.
- *Tabu search*. This heuristic is based on the idea of marking recently examined solutions and moving in their neighborhood without re-examining them for a certain amount of time, until a stop criterion is reached.
- *Genetic algorithms*. This heuristic uses a set of current solutions (*population*) and mimics the natural evolution of the population through suitable breeding and recombination mechanisms.

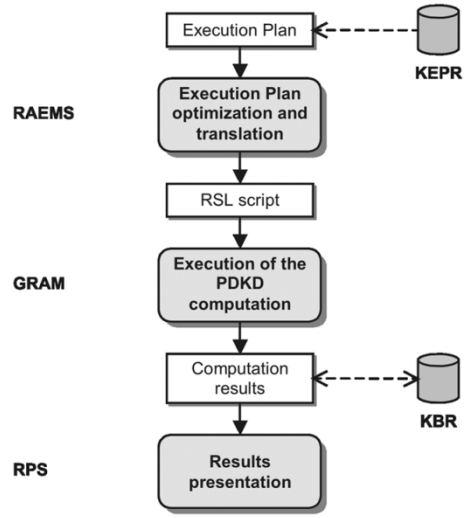

Fig. 10. Execution process of a data mining computation.

We have planned a thorough experimental evaluation of our scheduler. We will consider different approaches to cost estimation and different scheduling strategies, in order to assess their performance with respect to classical measures, such as distance from the optimal solution and *turnaround* time (i.e., total execution time, comprising the scheduling activity itself), but also in terms of other desirable properties, such as robustness w.r.t. unpredictable changes and stability.

Furthermore, we plan to integrate ontologies within the KDS, in order to possibly find partial matches between resource requests and descriptions, and also to give the possibility to have abstract software, data sources etc. Ontologies may help the scheduling task as it is often difficult to find identities between resource requests and descriptions. Ontologies can be created to explicitly describe resources and job requests allowing to perform semantic matching using terms defined in those ontologies. Resource descriptions and job requests may be expressed as concepts of ontologies while matchmaking rules allow to specify when a resource matches a job description (i.e., a request). Thus, to realize ontology-based request/resource matchmaking, an ontology describing requests, properties of the request (such as the request's owner), characteristics of the request (e.g., the type of the job requested) and resource requirements should be provided. Moreover, an ontology capturing the resource authorization and usage policies (e.g., a set of accounts that are authorized to access a specified computer system) is also needed. We are currently developing an ontology for the data mining domain that classifies data mining software tools. It will be used to simplify the development of distributed knowledge discovery applications on the grid, by offering to a domain expert a reference model for the different kind of data mining tasks, methodologies, and software available to solve a given problem, helping a user in finding the most appropriate solution. It can also be used as a starting point for matchmaking [37], [38].

IV. Execution of Grid Data Mining Applications

Fig. 10 shows the steps of the execution of a data mining computation on the grid. The execution plan optimization and translation is performed by means of the RAEMS, whose basic functionalities are provided by the VEGA components and by the scheduler (see Fig. 3).

Currently, VEGA integrates an *RSL generator* module, which produces an Resource Specification Language (RSL) script that can be directly submitted to the Globus resource allocation manager (GRAM) of a grid node running the Globus Toolkit. In opposition with the XML execution plan, the RSL script entirely describes 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 parameters, etc.). Fig. 11 shows an extract of a sample RSL script.

The execution of the computation is performed by means of the VEGA execution manager module. The execution manager allows the system to authenticate a user to the grid, by using the Globus grid security infrastructure (GSI) services, and submits 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. Finally, the execution manager collects the results of the distributed data mining computation and passes them to the RPS that, in turn, presents them to the user.

As it appears evident from the discussed example, it could often be necessary to move a large amount of data across nodes to perform a distributed data mining applications over the grid. To optimize data movements and prevent both wasted bandwidth and computational inefficiency, our approach adopts convenient scheduling strategies for software, datasets, and models movement (see Section III).

At the same time, ad hoc protocols and tools are crucial to perform efficient data transfer along heterogeneous networks having different latencies and bandwidths. To this end, in [39] we proposed a system to enhance the use of the GridFTP protocol for efficient data transfer on the grid. Such system is based on an algorithm that, on the basis of historical file transfer data, selects the appropriate GridFTP parameters for a required transfer session. Among other projects for distributed data analysis, DataSpace proposes a significant system to address efficient data access and transfer over the grid [40]. DataSpace is a Web services based infrastructure for exploring, analyzing, and mining remote and distributed data. DataSpace applications employ a protocol for working with remote and distributed data called DataSpace transfer protocol (DSTP). DSTP simplifies working with data by providing direct support for common operations, such as working with attributes, keys and metadata. The DSTP protocol can be layered over specialized high performance transport protocols such as SABUL [41], that allows DataSpace applications to effectively work on wide-area high-performance networks.

Finally, it is worth mentioning that security and privacy issues are critical in distributed environments such as grids. In this paper we did not address how privacy considerations can prevent the execution of distributed data mining applications on the *Knowledge Grid*. In fact, a typical issue in decentralized data mining, where data are distributed among two or more nodes, is how these nodes can cooperate to learn a global model without revealing their individual data. Several approaches and systems to allow privacy-preserving data mining have been pro-

```
+
...
(&(resourceManagerContact=g1.isi.cs.cnr.it)
  (subjobStartType=strict-barrier)
  (label=ws1_dt2)
  (executable=$(GLOBUS_LOCATION)/bin/globus-url-copy)
  (arguments=-vb -notpt gsiftp://g1.isi.cs.cnr.it/.../Unidb
             gsiftp://k2.deis.unical.it/.../Unidb
   )
)
...
(&(resourceManagerContact=k2.deis.unical.it)
  (subjobStartType=strict-barrier)
  (label=ws2_c2)
  (executable=.../IMiner)
  ...
   )
)
...
```

Fig. 11. Extract of a sample RSL script.

posed (see, for example, [42]). Even though we did not tackle this issue so far, an effort to integrate a set of standard privacy services into the *KNOWLEDGE GRID* will be investigated in the near future. The KDS can be used not only to search and access raw data, but also to find prediscovered knowledge that can be used to compare the output of a given *KNOWLEDGE GRID* computation when varying data, or to apply data mining tools in an incremental way.

V. EXPERIMENTAL RESULTS

To evaluate the efficiency of the *KNOWLEDGE GRID* prototype, we carried out a performance analysis of a classification task for intrusion detection of network data. To this end, a number of independent classifiers have been first computed by applying in parallel the same learning algorithm over a set of distributed *training sets*, generated through a random partitioning of the overall data set. Afterwards, the best classifier has been chosen by means of a *voting* operation taking into account evaluation criteria like computation time, error rate, confusion matrix, etc.

The training sets on which to apply the mining process have been extracted from a dataset with a size of 712 MBytes, containing about five million records produced by a TCP dump carried out during seven weeks of network monitoring. The C4.5 data mining tool has been used to generate a classifier based on decision trees.

After partitioning, each training set has been moved to a node of the grid providing the C4.5 data mining software. The induction of the decision trees has been performed in parallel on each node and the results have been next moved back to the starting node to execute the voting operation and the validation of the chosen model against a testing set.

The application has been designed using the VEGA visual environment as a set of workspaces reflecting the steps that compose the entire data mining application as described above. Several runs of the application have been performed to test the application and measure the execution times. In the following we present a comparison of the experimental results obtained on a single node (sequential case) with those of the distributed execution over three and eight nodes, respectively.

In order to improve transfers, data sets have been compressed before moving them to destination nodes; this permitted us to transfer files from 94% to 97% smaller in size with respect to original data sets.

Table I shows the execution times obtained during the experiments. Notice that the compression, data transfer, and decompression steps were not needed in the execution on one single node. It should be mentioned that, even when more than one node is used, partitioning and compression phases are still executed on a single machine, whereas decompression and computation phases are executed in parallel.

The experiments were performed on grid nodes of an early deployment of the SP3 Italian national grid funded by MIUR. Machines hardware was ranging from dual Pentium III 800 MHz workstations to Pentium 4 1500 MHz PCs.

Figs. 12 and 13 show, respectively, execution times and speedup achieved with the different configurations shown in Table I.It should be observed that, with 8 nodes, the speedup of the computational time is slightly superlinear. This is due to several factors, among which 1) machine heterogeneity, 2) caching effects, and 3) random partitioning of the dataset, possibly introducing inhomogeneity among the training sets. The total time suffers of the overhead added by the compression/decompression steps and data transfers. However, a total speedup factor of about two has been achieved employing three nodes and a speedup of about five was obtained by using eight nodes.

These results show how the use of the *KNOWLEDGE GRID* may bring several benefits to the implementation of distributed knowledge discovery applications both in terms of data analysis distribution and scalability results.

VI. RELATED WORK

As discussed in our recent work [12], [13] and, in different ways, in works of Berman [43] and Johnston [44], the creation of knowledge grids on top of computational grids middleware is the enabling condition to allow and favor the development of knowledge discovery processes. In general terms, a knowledge grid is an abstract problem solving environment that allows a user to express a problem using his/her domain specific knowledge and is able to translate it to the computational and data analysis operations of the underlying, real, problem solving system.

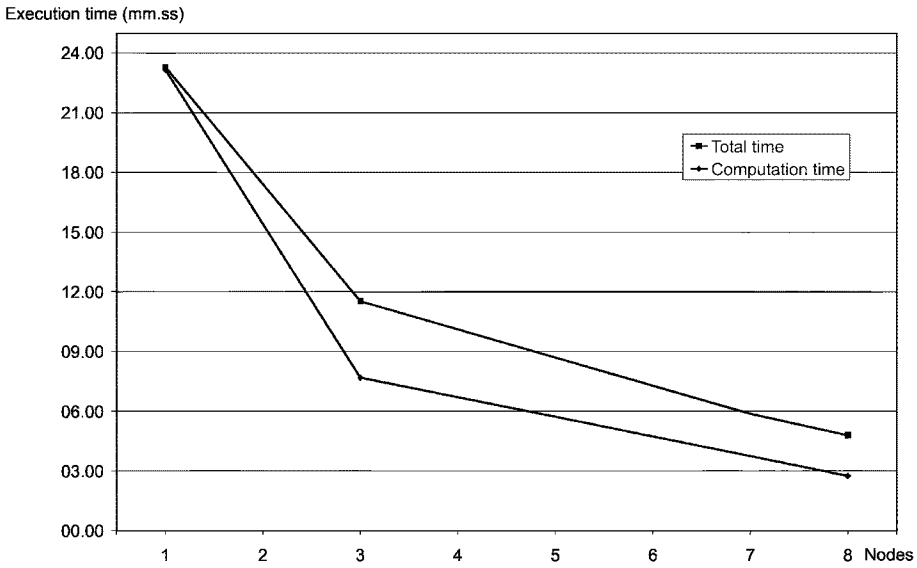

Fig. 12. Comparison of execution times.

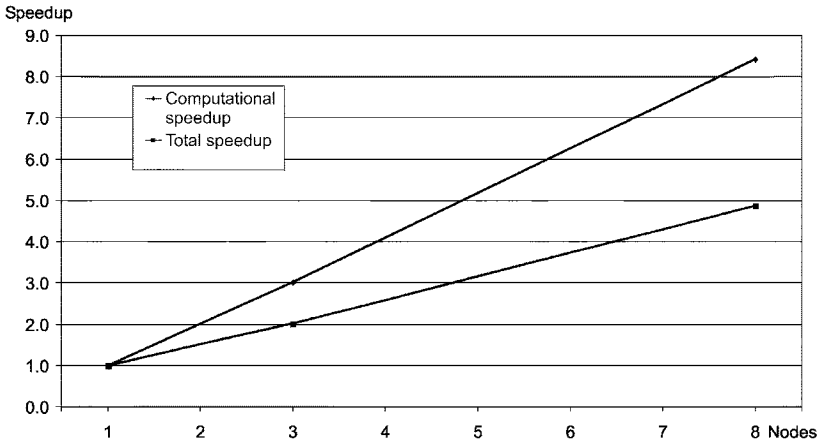

Fig. 13. Achieved speedup.

TABLE I
COMPARISON OF THE EXECUTION TIMES

No. of nodes	Dataset size per node	EXECUTION TIMES (mm:ss)					
		Partitio-ning	Compres-sion	Data transfer	Decom-pression	Compu-tation	TOTAL
1	180 Mb	00:08	0	0	0	23:10	**23:18**
3	60 Mb	00:09	01:13	01:33	00:56	07:41	**11:32**
8	23 Mb	00:12	00:37	00:55	00:18	02:45	**04:47**

A main issue in the implementation of such environments is a general, clear representation and manipulation of the knowledge base that is used to translate moderately abstract queries into sets of computations and data analysis that resolve the query. A second, important issue, regards the integration of two main characteristics of knowledge grids: the ability to synthesize data to provide useful and usable information and the ability to perform sophisticated large-scale computation leveraging the grid infrastructure.

Whereas some data mining systems that support high-performance distributed data mining recently appeared (see [45] and, for a short review, also [12]), there are really a few projects attempting to build knowledge grids on top of computational grids. More specifically, many parallel and distributed data mining systems operate on clusters of computers or over the Internet, but none of those, to the best of our knowledge, makes use of the computational grid basic services (e.g., authentication, data access, communication and security services). On the other hand, emerging knowledge grids can be roughly classified as domain-specific knowledge grids (e.g., TeraGrid, ADaM), and domain-independent knowledge grids. The *KNOWLEDGE GRID* we designed is one of the first attempts to build a domain-independent knowledge discovery environment on the grid.

In the rest of this section we shortly review the most significant grid-based projects/systems discussing differences and common aspects with respect to our *KNOWLEDGE GRID* system.

The TeraGrid project is building a powerful grid infrastructure, connecting four main sites in USA (San Diego Supercomputer Center, National Center for Supercomputing Applications, Caltech and Argonne National Lab), that will provide access to tera-scale amounts of data [43]. The most challenging application on the TeraGrid will be the synthesis of knowledge from very large scientific data sets. The development of knowledge

synthesis tools and services will enable the TeraGrid to operate as a knowledge grid. A first application is the establishment of the Biomedical Informatics Research Network to allow brain researchers at geographically distributed advanced imaging centers to share data acquired using different techniques and subjects. Such application makes a full use of a distributed data grid with hundreds of terabytes of data online, enabling the TeraGrid to be used fully as a knowledge grid in the biomedical domain. The use of the *KNOWLEDGE GRID* services can be potentially effective in these applications [11].

The algorithm development and mining (ADaM) system is an agent-based data mining framework developed at the University of Alabama in Huntsville, used to mine in parallel hydrology data from four sites [46]. The system comprises a mining engine and a daemon-controlled database. The database contains information about the data to be mined, including its type and location. A user provides the mining engine with a mining plan (i.e., a sequential list of mining operations that are to be performed along with any parameters that may be required for each mining operation). The mining engine consults the database in order to find out where the data to be mined is stored and then applies the mining plan to the set of data that has been identified. Each mining operation is represented as a shared-library file (one file per operation). In the grid version of ADaM the database and its associated daemon reside on a processor distinct from the one on which the mining engine operates. Data are managed at multiple sites through SRB/MCAT and GridFTP. This system uses a design approach similar to the *KNOWLEDGE GRID*, but the system architecture is simpler and the system purpose is limited to the application range for which the system has been designed.

The Discovery Net is a newly announced Engineering and Physical Sciences Research Council project (EPSRC), at Imperial College [47]. Its main goal is to design, develop and implement an infrastructure to support real time processing, interaction, integration, visualization and mining of massive amounts of time critical data generated by high throughput devices. The knowledge discovery process will be applied to raw and processed data from biotechnology, pharmacogenomic, remote sensing, and renewable energy data. The DNET architecture aims to develop high throughput sensing (HTS) applications by using the Kensington Discovery Platform on top of the Globus services. In this case the rationale is to port a Java-based distributed data mining system to grid platforms using the Globus Toolkit. The main question mark here is how the pre-existent system can adapt to grid mechanisms and policies.

Finally, the National Center for Data Mining (NCDM) at the University of Illinois at Chicago (UIC) is developing some significant testbeds on knowledge discovery over grids [48]:

- The Terra Wide Data Mining Testbed is an infrastructure built on top of DataSpace for the remote analysis, distributed mining, and real time exploration of scientific, engineering, business, and other complex data. Terra testbed uses the DataSpace predictive modeling and scoring protocol working with "events," which are abstractions representing new bits of information assumed to arrive in a real time stream. DataSpace supports an open standard called the Data Transformation Markup Language (DTML) for updating profiles with new events in real time or near real time.
- The Terabyte Challenge Testbed is an open, distributed testbed for DataSpace tools, services, and protocols. It consists of ten sites distributed over three continents connected by high performance links. It has been instrumented for network measurements and provides a platform for experimental discovery of scientific, engineering, business, and e-business data. The testbed includes a variety of distributed data mining applications, including the analysis of climate data, astronomical data, network data, web data, and business data.
- The Global Discovery Network is a collaboration between the National Center for Data Mining (Laboratory for Advanced Computing) and the Discovery Net. The new Global Discovery Network will link the Discovery Net to the Terra Wide Data Mining Testbed to create a combined global testbed with a critical mass of data. The Global Discovery Network is the first global high performance testbed for remote data analysis and distributed data mining and holds the promise of providing scientists and engineers easier ways to work with distributed data.

In summary, these emerging knowledge discovery-oriented grids are almost all facing specific application domains. Our system, besides being independent of the application domain, adopts specifically designed tools for the management of knowledge discovery results that allow a user to evaluate and compare different knowledge models and allow for the transparent integration of parallel and sequential data mining tools and algorithms.

VII. CONCLUSION AND FUTURE WORK

Parallel and distributed data mining suites and computational grid systems are two critical elements of future high-performance computing environments for e-science (data-intensive experiments), e-business (distributed online services), and virtual organizations support (virtual teams, virtual enterprises).

The grid infrastructure is growing up very quickly and is going to be more and more complete and complex both in the number of tools and in the variety of supported applications. Along this direction, grid services are shifting from generic computation-oriented services to high-level information management and knowledge discovery services.

Knowledge grids will enable entirely new classes of advanced applications for dealing with the data deluge. Their integration is a challenge whose achievements could produce many benefits in several application areas. Grids are coupling compute-oriented services with data-oriented and high-level information management services. This trend enlarges the grid application scenario and offers opportunities for high-performance distributed knowledge-based systems and services such as data mining and knowledge discovery.

The *KNOWLEDGE GRID* system we discussed here is a significant component of this trend. It integrates and completes the data grid services by supporting distributed data analysis and knowledge discovery and knowledge management services [43].

After introducing the general architecture of the *KNOWLEDGE GRID*, we presented VEGA, a tool that implements some services provided by the *KNOWLEDGE GRID*, and discussed step-by-step how a user can utilize VEGA to compose and execute a knowledge discovery computation in a simple way. Experimental results obtained from the execution of a distributed data mining application on a grid by using the *KNOWLEDGE GRID* have been presented. They demonstrate the feasibility of the proposed approach and show how the *KNOWLEDGE GRID* system can exploit the grid infrastructure for developing complex knowledge discovery applications.

ACKNOWLEDGMENT

The authors wish to thank the organizations involved in the MIUR SP3 project that offered their machines for running the experiments discussed in this paper.

REFERENCES

[1] I. Foster and C. Kesselman, "The anatomy of the grid: Enabling scalable virtual organizations. S. Tuecke," *Int. J. Supercomput. Applicat.*, vol. 15, no. 3, 2001.
[2] Building the Grid: An Integrated Services and Toolkit Architecture for Next Generation Networked Applications, I. Foster. (2000). [Online]. Available: http://www.gridforum.org/building_the_grid.htm
[3] Legion [Online]. Available: http://legion.virginia.edu
[4] Condor [Online]. Available: http://www.cs.wisc.edu/condor
[5] Unicore [Online]. Available: http://www.unicore.org
[6] The Globus Toolkit [Online]. Available: http://www.globus.org/toolkit
[7] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, "Data management in an international data grid project," in *Proc. IEEE/ACM Int. Workshop Grid Computing Grid*, Dec. 2000, pp. 77–90.
[8] P. Avery and I. Foster. (2001) GriPhyN Project Description. [Online]. Available: http://www.griphyn.org/info/index.html
[9] Y. Morita *et al.*, "Grid data farm for atlas simulation data challenges," in *Proc. Int. Conf. Computing High Energy Nuclear Physics*, 2001, pp. 699–701.
[10] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The data grid: Toward an architecture for the distributed management and analysis of large scientific datasets," *J. Netw. Comput. Appl.*, vol. 23, pp. 187–200, 2001.
[11] F. Berman, private communication, Nov. 2001.
[12] M. Cannataro and D. Talia, "The knowledge grid," *Commun. ACM*, vol. 46, no. 1, pp. 89–93, 2003.
[13] M. Cannataro, A. Congiusta, D. Talia, and P. Trunfio, "A data mining toolset for distributed high-performance platforms," in *Proc. Conf. Data Mining*, Bologna, Italy, 2002.
[14] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *Int. J. Supercomput. Applicat.*, vol. 11, pp. 115–128, 1997.
[15] The Grid Security Infrastructure. The Globus Project. [Online]. Available: http://www.globus.org/security
[16] The Monitoring and Discovery Service. The Globus Project. [Online]. Available: http://www.globus.org/mds
[17] The Globus Resource Allocation Manager. The Globus Project. [Online]. Available: http://www.globus.org/gram
[18] The Dynamically-Updated Request Online Coallocator (DUROC). The Globus Project. [Online]. Available: http://www.globus.org/duroc
[19] The Globus Heartbeat Monitor Specification v1.0. The Globus Project. [Online]. Available: http://www.globus.org/hbm/heartbeat_spec.html
[20] The GridFTP protocol. The Globus Project. [Online]. Available: http://www.globus.org/datagrid/gridftp.html
[21] The Globus Replica Catalog. The Globus Project. [Online]. Available: http://www.globus.org/datagrid/replica-catalog.html
[22] The Globus Replica Management API. The Globus Project. [Online]. Available: http://www.globus.org/datagrid/replica-management.html
[23] The Globus Resource Specification Language RSL v1.0. The Globus Project. [Online]. Available: http://www.globus.org/gram/rsl_spec1.html
[24] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Symp. Mathematical Statistics Probability*, 1967, pp. 281–297.
[25] Intelligent Miner [Online]. Available: http://www.software.ibm.com/data/iminer/
[26] P. Cheeseman and J. Stutz, "Bayesian classification (autoclass): Theory and results," in *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. Cambridge, MA: MIT Press, 1996, pp. 61–83.
[27] J. R. Quinlan. (2002) See5/C5.0, version 1.16. [Online]. Available: http://www.rulequest.com/see5-info.html
[28] R. Wolski, N. Spring, and J. Hayes, "The network weather service: A distributed resource performance forecasting service for metacomputing," *Future Gener. Comput. Syst.*, vol. 15, 1999.
[29] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "Scheduling high performance data mining tasks on a data grid environment," *Europar*, 2002.
[30] Y. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, 1999.
[31] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.
[32] D. Arnold, H. Casanova, and J. Dongarra, "Innovation of the netsolve grid computing system," *Concur. Comput.*, 2002.
[33] A. Abraham, R. Buyya, and B. Nath, "Nature's heuristics for scheduling jobs on computational grids," in *Proc. IEEE 8th Int. Conf. Advanced Computing Communications*, 2000.
[34] H. Dail, H. Casanova, and F. Berman, "A Modular Scheduling Approach for Grid Application Development Environments," UCSD, 2002.
[35] H. J. Siegel and S. Ali, "Techniques for mapping tasks to machines in heterogeneous computing systems," *J. Syst. Architect.*, vol. 46, 2000.
[36] M. Mika, G. Waligora, and J. Weglarz, "A metaheuristic approach to scheduling workflow jobs on a grid," in *Grid Resource Management*, J. Nabrzyski, J. Schopf, and J. Weglarz, Eds. Norwell, MA: Kluwer, 2003.
[37] M. Cannataro and C. Comito, "A data mining ontology for grid programming," in *Proc. 1st Int. Woprkshop Semantics Peer-to Peer Grid Computing (SemPGrid)*, 2003, pp. 113–134.
[38] H. Tangmunarunkit, S. Decler, and C. Kesselman, "Ontology-resource matching in the grid the grid meets the semantic web," in *Proc. 1st Int. Workshop Semantics Peer-to-Peer Grid Computing*, pp. 85–101.
[39] M. Cannataro, C. Mastroianni, D. Talia, and P. Trunfio, "Evaluating and enhancing the use of the GridFTP protocol for efficient data transfer on the grid," in *Proc. 10th Euro PVM/MPI*, vol. 2840, Venice, Italy, 2003, pp. 619–628.
[40] DataSpace [Online]. Available: http://www.dataspaceweb.net/
[41] Y. Gu, X. Hong, M. Mazzucco, and R. Grossman, "SABUL: A high performance data transfer protocol," *IEEE Commun. Lett.*, submitted for publication.
[42] C. Clifton *et al.*, "Tools for privacy preserving distributed data mining," *ACM SIGKDD Explorations Newsletter*, vol. 4, no. 2, pp. 28–34, 2002.
[43] F. Berman, "From teragrid to knowledge grid," *Commun. ACM*, vol. 44, no. 11, pp. 27–28, 2001.
[44] W. E. Johnston, "Computational and data grids in large-scale science and engineering," Future Generation Computer Systems, 2002, to be published.
[45] *Advances in Distributed and Parallel Knowledge Discovery*, H. Kargupta and P. Chan, Eds., MIT Press, Cambridge, MA, 2000.
[46] T. Hinke and J. Novonty, "Data mining on NASA's information power grid," in *Proc. 9th IEEE Int. Symp. High Performance Distributed Computing*, 2000.
[47] Y. Guo. (2002) Discovery Net. [Online]. Available: http://www.lesc.ic.ac.uk/projects/dnet.html
[48] (2002) Testbeds. National Center for Data Mining, Laboratory for Advanced Computing, Univ. Illinois, Chicago. [Online]. Available: http://www.ncdm.uic.edu/testbeds.htm

Mario Cannataro (A'94) received the Laurea Degree (*cum laude*) in computing engineering from the University of Calabria, Italy, in 1993.

He is an Associate Professor of Computer Engineering at University "Magna Græcia" of Catanzaro, Italy, and a co-founder of Exeura. His current research interests include grid computing, bioinformatics and proteomics, grid-based problem solving environments, and adaptive hypermedia systems. He published a book and more than 100 papers in international journals and conference proceedings. He is serving as a program committee member of several conferences.

Prof. Cannataro is a member of ACM.

Antonio Congiusta (S'01–A'01) received the Laurea degree in computer engineering from the University of Calabria, Rende, Italy, in 2002. He is currently pursuing the Ph.D. degree in systems and computer engineering from the same university.

From 2002 to 2003, he was a Research Fellow at the Institute of High Performance Computing and Networking of the Italian National Research Council (ICAR-CNR), working on data mining applications on the grid. His research interests are focussed on grid programming environments, grid services, and workflow based grid systems.

Andrea Pugliese (S'03) received the Laurea degree (*cum laude*) in computer engineering from the University of Calabria, Rende, Italy, in 2000, and is currently pursuing the Ph.D. degree in systems and computer engineering from the same university.

His current research interests include architectures and strategies for Grid scheduling, grid services, and semistructured data.

Mr. Pugliese is a student member of the ACM.

Domenico Talia (A'94) received the Laurea degree in physics from the University of Calabria, Rende, Italy.

He is a Professor of computer science at the Faculty of Engineering, University of Calabria, Rende, Italy. His main research interests include grid computing, parallel computation, parallel data mining, parallel programming languages, cellular automata, computational science, and peer-to-peer computing. He is a Member of the editorial boards of the IEEE Computer Society Press, the Parallel and Distributed Practices journal, the Future Generation Computer Systems journal, and the Web Intelligence and Agent Systems International journal. In addition he a member of the advisory board of Euro-Par conference series and a member of the advisory committee of the IEEE Task Force on Cluster Computing (TFCC). He served as a distinguished speaker in the IEEE Computer Society Chapter Tutorials Program and in the IEEE Computer Society Distinguished Visitors Program. He was guest editor of special issues of IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Parallel Computing and Future Generation Computer Systems and he is serving as a program committee member of several conferences. He published three books and more than 130 papers in international journals such as *Communications of the ACM*, *Computer*, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, and conference proceedings.

Dr. Talia is a member of the ACM.

Paolo Trunfio (S'03) received the Laurea degree in computer engineering from the University of Calabria, Rende, Italy, in 2001. He is currently pursuing the Ph.D. degree in systems and computer engineering from the same university.

He collaborated with the Institute of High Perfomance Computing and Networking of the Italian National Research Council (ICAR-CNR) in the area of grid computing. His current research interests include parallel and distributed computing and peer-to-peer systems.