The DataGrid Workload Management System: Challenges And Results

G. Avellino, S. Beco, B. Cantalupo, A. Maraschini, F. Pacini, M. Sottilaro and A. Terracina
DATAMAT S.p.A., Via Laurentina 760, I-00143 Roma, Italy

D. Colling
Physics Department, Imperial College London, Prince Consort Road, London, SW7 2BW, United Kingdom

F. Giacomini and E. Ronchieri
INFN CNAF, Viale Berti Pichat 6/2 , I-40127 Bologna, Italy

A. Gianelle, R. Peluso and M. Sgaravatto
INFN Sezione di Padova, Via Marzolo 8, I-35131 Padova, Italy

A. Guarise, R. Piro and A. Werbrouck
INFN Sezione di Torino, Via P. Giuria 1, I-10125 Torino, Italy

D. Kouřil, A. Křenek, L. Matyska, M. Mulač, J. Pospíšil, M. Ruda, Z. Salvet, J. Sitera, J. Škrabal and M. Voců
CESNET z.s.p.o., Zikova 4, 160 00 Praha 6, Czech Republic

M. Mezzadri and F. Prelz
INFN Sezione di Milano, Via Celoria, 16, I-20133 Milano, Italy

S. Monforte and M. Pappalardo
INFN Sezione di Catania, Via S. Sofia 64, I-95123 Catania, Italy

Abstract. The workload management task of the DataGrid project was mandated to define and implement a suitable architecture for distributed scheduling and resource management in a Grid environment. The resulting artifact was the design and implementation of a Grid Workload Management, implementing a super-scheduler with the distinguishing feature of taking into account also data access requirements when scheduling jobs to the available Grid resources. Many novel issues in various fields were faced: resource management, resource reservation and co-allocation, Grid accounting, etc. In this paper the architecture and the functionality provided by the DataGrid Workload Management System are presented.

Keywords: Grid scheduling, distributed resource management, Grid workload management

1. Introduction

The workload management task of the DataGrid project [1] was mandated to define and implement a suitable architecture for distributed scheduling and resource management in a Grid environment. The aggressive schedule of the project along with the particular thrust on meeting the schedules and needs of the collaborating HEP (High Energy Physics), Earth Observation and Bioscience projects, required, besides new and original developments, an effective collaboration with other projects (such as Globus [13] and Condor [20]) and a design built on different existing tools. Many novel issues in many different fields were faced: resource management, scheduling, resource reservation and co-allocation, distributed accounting, etc.

The result of all these activities was the design and implementation of a Grid Workload Management System (WMS). This implements a super-scheduler [27] with the distinguishing feature of taking data access into account when scheduling jobs to the available Grid resources. This WMS has been proved to successfully address many aspects of the problem of scheduling and resource management in a Grid environment. The success of this software can be measured by its deployment and use in environment outside of the DataGrid testbeds.

This paper discusses the architecture and the functionality provided by this Workload Management System. Section 2 discusses the architecture of the WMS and its components and describes the basic functionality. In section 3 more advanced functionality provided by the WMS is reported. Section 4 discusses about results and achievements, in particular in term of deployment and use of the WMS, and of its measured performance. In section 5 comparisons with related work are presented. Section 6 concludes the paper.

2. The DataGrid Workload Management System

The architecture of the DataGrid Workload Management System is represented in Fig. 1.

The *User Interface* (UI), discussed in more detail in Sect. 2.1 is the component that allows users to access the functionality offered by the Workload Management System. In particular it allows jobs described in a *Job Description Language* (JDL) to be submitted.

The *Network Server* (NS) is a generic network daemon, responsible for accepting incoming requests from the UI (e.g. job submission, job removal), which are validated and passed to the Workload Manager.

The *Workload Manager* is the core component of the Workload Management System: given a valid request, it has to take the appropriate actions to satisfy it. To do so, it may need support from other components, which are specific to the different request types.

The *MatchMaker* (or *Resource Broker*, RB) is one of the classes offering support to the Workload Manager. It provides a matchmaking

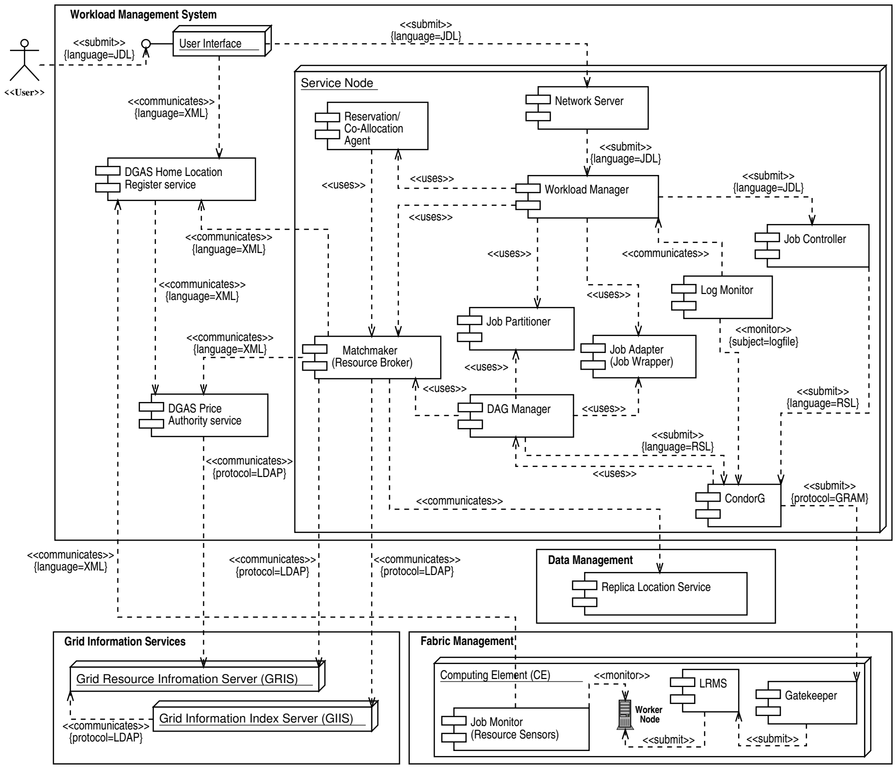

Figure 1. WMS component and deployment diagram.

service: given a request (e.g. for a job submission, or for a resource reservation request), it finds the resources that best match it. Details on the Resource Broker design and implementation are discussed in Sect. 2.2.

The *Job Adapter* (JA) is responsible for making the final "touches" to the job before it is passed to the Job Submission Services (discussed in Sect. 2.3) for the actual submission. In particular it is responsible for creating the appropriate execution environment in the Computing Element (CE) worker node where the execution takes place.

The *Logging and Bookkeeping* (L&B) service is another component of the WMS: it has not been represented in the figure for simplicity. It stores logging and bookkeeping information concerning events generated by the various components of the WMS. Using this information, the L&B service, discussed in Sect. 2.4, keeps a state machine view of each job.

The other components of the WMS represented in the figure are discussed in the following sections.

Determining the hierarchy and distribution of WMS instances (the Service Nodes, with respect to Fig. 1) is not a trivial issue. On one extreme a single, central WMS would provide optimal resource allocation fairness, but poses scalability problems that are very hard to address with commodity hardware. On the other extreme one could imagine a dedicated WMS (some sort of Grid-browser) for each end-user, giving freedom in the choice of resource, but also causing races for distributed resources that cannot be fairly arbitrated. The intermediate solution usually deployed was to provide a WMS for each Virtual Organisation (VO) participating in the Grid. In order to be able to experiment with different choices, however, the WMS was designed to work both at the "personal" and at the VO level.

2.1. The User Interface and the Job Description Language

A Workload Management System has to provide users with suitable means to express their needs, which have then to be forwarded and interpreted by the Grid middleware. Therefore mechanisms to represent customers and resources of the system are needed, i.e.:

- the characteristics of jobs (executable name and size, parameters, number of instances to consider, standard input/output/error files etc.)
- the resources (CPUs, network, storage, etc.) required for the processing and their properties (architecture types, network bandwidth, latency, assessment of required computational power, etc.)

This information has then to be provided to the actual workload management software layer, through an appropriate API or a specific, flexible, easy to use (Graphical) User Interface.

A high-level, user-oriented DataGrid Job Description Language (JDL), to describe both jobs and resources, was therefore defined. In the DataGrid JDL based on the Condor ClassAd language [25], the central construct is a record-like structure, the *classad*, composed of a finite number of distinct attribute names mapped to expressions. These ads conform to a protocol that states that every description should include expressions named *Requirements* and *Rank*, which denote the requirements and preferences of the advertising entity. Two entity descriptions match if each ad has an attribute (Requirements) that evaluates to `true` in the context of the other ad.

For the convenience of the reader, we quote from [25] some of the features of this framework that make it particularly fitting for our application:

- The use of a semi-structured data model, so no specific schema is required for the resources description, allowing it to work naturally in a heterogeneous environment.
- The query language folded into the data model. Therefore requirements (i.e. queries) may be expressed as attributes of the job description.
- The possibility to arbitrarily nest descriptions, leading to a natural language for expressing resources and jobs aggregates or co-allocation requests.

The JDL defined for the DataGrid WMS provides attributes to support:

- the definition and specification of batch, interactive, MPI-based, checkpointable and partitionable jobs;
- the definition of aggregates of jobs with dependencies (Direct Acyclic Graphs, DAGs);
- the specification of constraints to be satisfied by the selected computing and storage resources, including also data access requirements;
- the specification of preferences for choosing among multiple suitable resources (ranking expressions).

Despite its extensibility, the JDL encompasses a set of predefined attributes having a special meaning for the underlying components of the Workload Management System.

The Requirements and Rank expressions are built using the *resources attributes*, which represent the characteristics and status of the resources. These resources attributes are not part of the predefined set of attributes for the JDL as their naming and meaning depends on the adopted Information Service schema [3]. This independence of the JDL from the resources information schema allows targeting for the submission resources that are described by different Information Services without any changes in the job description language itself. Hereafter follows an example of the JDL used to describe a simple job:

```
[
    Type = "Job";
    Executable = "/bin/bash";
    StdOutput = "std.out";
    StdError = "std.err";
```

```
    Arguments = "./sim010.sh";
    Environment = "GATE_BIN=/usr/local/bin";
    OutputSandbox = {"std.out","std.err","Brain_radioth000.root"};
    InputData = { "lfn:BrainTotal", "lfn:EyeTotal" };
    DataAccessProtocol = { "gridftp" };
    OutputSE = "grid011.pd.infn.it";
    InputSandbox = { "sim010.sh", "macro000.mac", "status000.rndm"};
    rank = -other.GlueCEStateEstimatedResponseTime;
    requirements = other.GlueCEStateFreeCPUs >= 2;
]
```

After having described the characteristics and requirements of their applications, users expect, without caring about the complexity of the Grid middleware software, to be enabled to submit them to the Grid and monitor their execution. In the DataGrid WMS this is accomplished by means of the User Interface (UI) component. The UI provides access to all services of the WMS and includes all the public interfaces exposed by the WMS. The UI allows access to all the job management related functionality: job submission, job cancellation, job status and output retrieval, listing of the resources suitable to run a specific job, etc. All this functionality is made available through a Python command line interface and an API providing C++ and Java bindings.

The JDL and the command line User Interface provide a very powerful means to interact with the Grid. However JDL-based definitions of jobs can become quite complex, since users to some extent are required to know some details of the definition language. Moreover the increasing number of available options, and the added complexity, can make the learning curve of the command line UI steeper. In order to relieve users from this burden, a set of flexible and easily configurable Java graphical components were provided.

2.2. The Resource Broker

As introduced in Sect. 2, the Resource Broker, RB (or MatchMaker) is one of the classes offering support to the Workload Manager. It provides a matchmaking service, which relies on the ClassAds mechanisms [25] provided by the Condor project : given a job submission or resource reservation request (represented by a JDL expression), it finds the resources that best match this request. In particular it is responsible for finding the resource that best matches the requirements and preferences of a submitted user job, considering also the current distribution of load on the Grid. The distinguishing feature of this DataGrid super-scheduler is the ability to also take into account the data access requirements that are typical of many applications.

Actually the Resource Broker can be decomposed in three sub-modules:

- a sub-module responsible for performing the matchmaking, therefore returning all the resources suitable for that JDL expression;
- a sub-module responsible for performing the ranking of matches resources, therefore returning just the "best" resource suitable for that JDL expression;
- a sub-module implementing the chosen scheduling strategy, which can be based on the services of the previous two modules; this is easily pluggable and replaceable with other ones implementing different scheduling strategies.

In order to achieve its goal, the Resource Broker interacts with other Grid services. In particular, as shown in Fig. 1, it interacts with the Information Services (to know the status of the available computing and storage resources) and with the Data Management Services (to know where the required data are physically available in the Grid). This information is cached and converted to JDL for symmetric matching to the user requests.

The default implemented scheduling policy (replaceable with different ones) is to submit to a resource where the submitting user has proper authorization, that matches the characteristics specified in the job JDL, and where the specified input data (and possibly the chosen output storage resource) are determined to be "close enough", therefore minimizing the overall access cost to the required data. In this matchmaking framework not only computing resource, but also storage resource information can be taken into account. For example it is possible to request that a job must be submitted to a computing resource close to a storage system where enough storage space is available. This is an application of a more general match-making technique sometimes referred to as *gang-matching* [26].

2.3. The Job Submission Services

The Job Controller (JC) is the component of the workload management system responsible for the actual job management operations, issued on request of the Workload Manager. In particular the JC is responsible for managing the job submission and job removal requests.

The JC is essentially a wrapper around the CondorG system [15], which leverages the security and resource access protocols as supported within the Globus Toolkit [13] (namely the Globus GSI and GRAM services), and the intra-domain resource management methods of Condor [20].

CondorG was chosen when implementing the job submission services of the DataGrid WMS for several reasons, in particular:

- CondorG relies on the Globus GRAM protocol, which is a de-facto standard for Grid resource access.
- The reliable two-phase commit protocol used by CondorG for job management operations.
- The reliability of the system: CondorG keeps a persistent (crash proof) queue of jobs, used as a persistent database storing information concerning active jobs.
- The need for interoperability with the US Grid projects, of which CondorG is an important component.

2.4. The Logging and Bookkeeping Service

The L&B service collects and manages job-related data from the WMS components, which are not directly accessible by the user otherwise, to provide a global view on job states, as well as aggregate information on compound jobs.

The data are collected in terms of L&B *events* of the following types:

- *Register.* The very first event of each job. It records the basic information on the job type, its definition, owner, etc.
- *Transfer, Accept, Refuse, EnQueue, DeQueue, Call, Return.* Passing job control between WMS components over network connections, through reliable job queues, and by calling a procedure.
- *Match, Run, Done, Cancel, Resubmit, Abort.* Other important points of the job life (finding a matching CE etc.).
- *UserTag.* Arbitrary user's annotation of the job (`key = value`).

All the events carry common information (JobId, timestamp, originating component, ...), as well as event-type specific attributes, e. g. where the job is transfered, or its exit code.

Main components of the event delivery infrastructure are shown in Fig. 2. The events are posted with calls to the *L&B producer library* [19] which passes them on to the *locallogger* daemon, running physically close (preferably on the same machine) in order to avoid any sort of network problems.

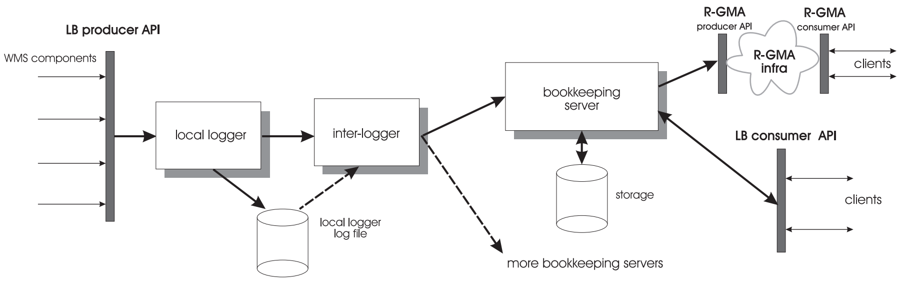

Figure 2. Block diagram of the L&B infrastructure.

Further on, event delivery is managed by the *interlogger* daemon. It takes the events from the locallogger (or its log files on crash recovery), and repeatedly tries to deliver them to the destination bookkeeping server (known from the JobId) until it eventually succeeds. The event delivery chain is therefore almost non-blocking and highly reliable.

A counterpart to the standard delivery is the *synchronous mode* (used under specific circumstances, e. g. for the *Register* event, when the resulting performance penalty is acceptable). In this case a reverse confirmation channel is established so that the logging call does not return until the event is accepted by the server.

Besides being stored, events are processed by the L&B server to give a higher level view — the *job state* seen by the user. The following states are identified:

– *Submitted, Waiting, Ready, Scheduled.* 4 stages of job preparation, i. e. accepted by WMS, transfered to RB, being assigned a target CE, and transfered to the CE queue.

– *Running.* The job actually started execution.

– *Done, Cancelled, Aborted.* Normal and abnormal (user or system initiated) termination.

– *Cleared.* Job output was retrieved and purged.

Again, each state carries an appropriate set of attributes, e. g. job owner, job description (JDL), destination CE, reason of failure etc. In the case of DAGs (see Sect. 3.1) basic statistics on the sub-jobs (i. e., how many of them are in each state) is computed as well.

Information available in the bookkeeping server is retrieved by *L&B consumers* using two classes of queries: *job queries* which return one or more jobs including their states, and *event queries* returning raw events. The queries specify conditions on various job and event attributes (e. g. JobId, job owner, destination, various timestamps, etc., as well as specific user tags).

However, completely arbitrary queries would translate into full searching through the large L&B server database, easily overloading the server. Therefore the L&B database can be *indexed* according to selected job attributes (including user tags). The server rejects queries which cannot make use of the indices. Selectivity of individual attributes is highly specific for a given users' community, hence the actual index configuration is configurable and left upon the server administrator.

Finally, L&B server is capable of pushing job state changes into the R-GMA [10] monitoring infrastructure. Careful setup of the infrastructure leads to considerable optimizations in the L&B data distribution towards the end users, namely notifications on job state changes instead of otherwise necessary frequent database polling.

2.5. Security in the Workload Management System

WMS has access to a lot of Grid resources and often processes users' sensitive data. Therefore special attention was paid to security issues during designing and implementing the WMS components.

Network communications between all the WMS components are mutually authenticated using the Grid Security Infrastructure (GSI [14]), which also provides integrity protection of the protocols. GSI is based on a Public Key Infrastructure and use of short-time PKI credentials (*proxies*), which offer a secure way for single sign-on and credential delegation. Each user must present a valid GSI credential at each interaction with the WMS, and possibly delegate it to the WMS. Each job submitted to WMS must be accompanied with a delegated credential so the job owner is always identifiable and the job itself is able to access authenticated services. The WMS components can use either their own credential or a delegated user credential when acting on the user's behalf, e. g. while submitting a job to a target CE.

Even though each job submitted to the WMS has a valid GSI credential, the overall job lifetime can easily exceed the lifetime of the credential, which is usually a few hours. Using credentials with longer lifetime does not solve the problem since the overall job lifetime is unpredictable and the security is decreased with long lived credentials.

To address the problem the WMS offers the *Proxy Renewal Service*, which allows the WMS to keep the job's credential valid for all its lifetime. The service is built upon the MyProxy Service [23], a credential storage system that allows users to store and maintain their long-lifetime credentials, and which issues users' short-time credentials to a small, well-defined set of clients, including the WMS. Running as part of the WMS, the Proxy Renewal service registers the certificate of a submitted job and, by periodically contacting the MyProxy Service, keeps the proxy certificate valid.

The proxy renewal mechanism ensures that jobs submitted to the WMS have valid credentials throughout their lifetime, while preserving the advantages of short-time proxy certificates. The proxy renewal process is performed in a controlled way, with only a small set of trusted services allowed to renew proxies. All transactions concerning proxy renewal are logged and can be analyzed in the future for possible misuse. Since management of user's proxy certificates in the repository is completely controlled by the user, they may stop renewal at any time by removing their credential from the repository.

The WMS has also integrated support for the Virtual Organization Management System (VOMS [2]), which allows to specify and manage sets of groups or roles within a particular VO and is part of the DataGrid authorization framework. Various WMS components use this information for either making authorization decisions (e. g. access control to data in the L&B service) or finding out details about users' VOs (e. g. jobs owned by a given user can only be submitted to CEs available to that user).

3. Advanced functionality

Besides allowing the management of simple, sequential batch jobs, as described in the previous section, the DataGrid WMS also provides some advanced capabilities: mechanisms for Grid accounting, a framework for resource reservation and co-allocation, the possibility to submit parallel, interactive, checkpointable, partitionable jobs, , etc. Some of this functionality are described in the following subsections.

3.1. Jobs with Dependencies

The concept of a "job" only rarely corresponds to an executable to be run on a worker node, reading some input and producing some output. Often a job is better expressed as a set of interdependent tasks, collaborating to achieve a common goal. It is then desirable that a Grid implementation provides mechanisms to manage an arbitrary complex workflow.

In its simplest form a workflow can be expressed as a Directed Acyclic Graph (DAG), where nodes represent jobs and arcs represent temporal dependencies between any two of them. For example an arc between node A and node B means that the job associated to node B can start only when the job associated with node A has finished (successfully).

Within the WMS the support for DAGs is implemented on top of a tool called *DAGMan*, developed within the Condor project. DAGMan is a meta-scheduler whose purpose is to navigate through the graph, processing the nodes that happen to be free of dependencies. For each DAG submitted to CondorG, a DAGMan process is locally spawned.

In the DataGrid WMS, the specification of a DAG node has been inherited from DAGMan, which consists of three parts: a PRE script, the user's job and a POST script.

When DAGMan processes a node, it first executes the PRE script. If this succeeds DAGMan submits the user's job to CondorG and keeps monitoring it. When the job finishes, either successfully or unsuccessfully, the POST script is run. A node is considered successful if all the three parts are successful.

The external interface that the WMS offers to manage DAGs is independent of the DAGMan based implementation. Three aspects contribute to this interface: the language used to specify a DAG, the API provided to manage a DAG and the state machine representing the status of a DAG during its lifetime.

To express DAGs the customary Job Description Language introduced in Sect. 2.1 is considered. For example, the DAG shown in Fig. 3.1 can be expressed as follows:

```
[
  type = "dag";
  nodes = [
    nodeA = [ file = "A.jdl" ];
    nodeB = [ file = "B.jdl" ];
    nodeC = [ file = "C.jdl" ];
    nodeD = [ file = "D.jdl" ];
    dependencies = {
      {nodeA, {nodeB, nodeC}},
      {{nodeB, nodeC}, nodeD}
    }
  ]
]
```

where `A.jdl`, `B.jdl`, `C.jdl` and `D.jdl` are files containing the description of the respective jobs. The contents of those files can be directly embedded in the DAG description.

Within the WMS a DAG is considered as a single unit of work. As such it has its own entries in the Logging and Bookkeeping Service, its passing through the system generates events that feed the standard state machine for jobs, it can be queried for status, it can be can-

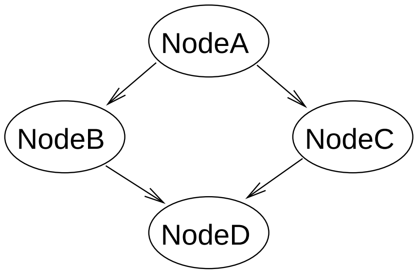

Figure 3. A diamond shaped DAG

celled, etc. Moreover its nodes, being themselves jobs, can be managed independently.

As a rule of thumb, the later the scheduling decision for a job is taken the better. Following this guideline the scheduling of the DAG nodes is not performed when the DAG is submitted (*eager scheduling*) but is deferred until the single node is ready, i.e. free of dependencies (*lazy scheduling*) [16]. In principle the decision can be deferred to the moment a certain worker node is actually free (*very lazy scheduling*) but this approach raises other problems, so it was not pursued further in the project.

3.2. Job checkpointing

Checkpoint is a service to facilitate automated recovery and continuation of interrupted computations with the aid of periodically recorded checkpoint data. Checkpointing is essential in particular for long-running computations to minimize lost time and other costs incurred by system failures. Checkpointing also enables job preemption, job migration and other interruptions to calculations [28].

In the DataGrid WMS an application level checkpointing service has been implemented: state saving is explicitly part of the application. A proper Grid checkpointing API was provided, to allow applications to be instrumented to save, at any moment during the execution of a job, the state of the process (represented by a list of <var, value> pairs) and eventually to restore a previously saved state, to restart the computation from these checkpoint data.

In the framework of the DataGrid Workload Management System, this Grid checkpoint service is basically used to enable failure recovery to applications: if a job, instrumented with the provided Grid checkpointing APIs (and which therefore while running saves from time to time its intermediate results) fails, the Workload Management System automatically reschedules the job and resubmits it to another compatible resource. When the job restarts its execution, the last saved state is automatically retrieved, and therefore the computation is restarted from that point. If the failure can not be detected by the Grid middleware, the user can retrieve a saved state for her job (usually the last saved one), and resubmit it specifying that it must start from these retrieved checkpoint data.

The functionality to persistently save the state of a job and to retrieve a previously saved state, in the DataGrid WMS is provided by the L&B service: as described in Sect. 2.4, among the various attributes associated for each job, also its job states (checkpoints) get stored here.

3.3. Interactive Jobs

The interactive job support in the DataGrid WMS relies on the *bypass* software [29] (more specifically on the Grid Console module) developed by the Condor group. Grid Console is an implementation of an interposition agent system: a software module which sits between the application layer and the operating system layer, and grabs control and manipulates the results when such system calls are invoked by the user application. It consists of two software components: the Grid Console Agent and the Grid Console Shadow. The Grid Console Agent intercepts reads and writes on stdin, stdout, and stderr, while all other operations are left untouched. Reads and writes on these streams are forwarded to the Grid Console Shadow for execution.

Relying on these mechanisms, the interactive job support implemented in the DataGrid WMS allows the job standard streams to be redirected from the worker node where the execution is taking place to a remote machine, usually the User Interface node, so that the user can interact with the running job.

Fig. 4 shows the flows for interactive jobs in the DataGrid WMS.

The *Job Shadow* is a DataGrid customized Grid Console Shadow running on the UI machine, while the *Job Agent*, installed on the worker node where the job runs, consists of a set of processes linked against a slightly modified Grid Console Interposition Agent. The Job Agent gets automatically staged and installed on this executing node when an interactive job is submitted for execution.

When the interactive jobs runs, its standard streams are therefore sent to the Shadow machine (usually the UI node).

Reliable mechanisms have been designed and implemented: if the UI crashes or the user disconnects from it, the Job Agent writes the output locally and tries to reconnect. This is retried for a while before giving up and aborting the job.

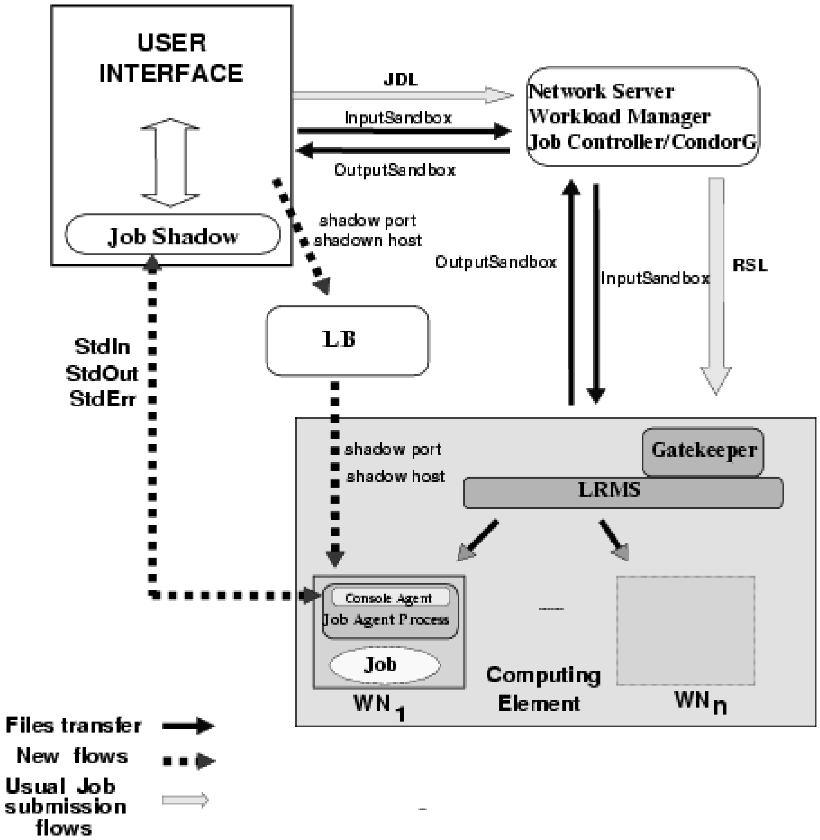

Figure 4. Interactive job flow.

3.4. Grid Accounting

One of the objectives when designing and implementing the DataGrid WMS was to provide an accounting system that traces the resource usage by the authorized Grid users. Instead of using a traditional approach—simply recording usage metrics for user jobs—we implemented an economy-based approach (users "pay" virtual credits for resource usage) in order to support an economic brokering of Grid resources. The main goal of economic brokering is to help in balancing the overall workload by pricing resources according to their current state (for example, queue length). In an economic context, the goal of resource exchange fairness implies not only a fair exchange of computational energy [24], but also concerns the temporal value of Grid resources.[1]

It is widely believed that an economic approach offers natural self-regulating mechanisms that can help with allocating resources to Grid users in a fair and effective manner by balancing demand and supply, i.e. reaching market equilibrium[2] as long term optimization. It may also help in balancing the incoming workload among the participating Grid resources ("just-in-time" scheduling as short term optimization).

The fundamental services of which the DataGrid Accounting System (DGAS) is composed are the so-called *Home Location Register* (HLR) service—responsible for resource usage accounting—and the *Price Authority* (PA) service—responsible for resource pricing, as shown in Fig. 1. DGAS services are based on the assumption of distributed and decentralized control, to assure scalability, and the existence of trust-relationships between all Virtual Organizations that are collaborating in the Grid, and thus DGAS allows to have an arbitrary topology of HLR and PA servers.[3]

Each HLR ("bank") server manages a subset of Grid users and resources by maintaining their respective account and transaction (job) information in a relational database. User and resource accounts are updated after each job execution using resource usage records that are provided by the executing computing resource. In this context, the authorization to use a specific resource should depend not only on the authorization policy of the resource owner, but also on the availability of user credits, thus accounting and authorization are tightly bound in our model.

The single PA servers furnish upon request valid price quotations for the resources within their administrative domain. The current implementation of the PA service adopts a per-resource pricing, where resource prices depend only on the current state of the resource. Each PA server keeps a history of resource prices, including their respective GMT timestamps, so that the prices that were valid at job submission time can be retrieved even after they have been updated.

Each Grid user should receive the amount of credits he needs from the management of her Virtual Organization, that redistributes the credits earned by its resources among its users. If necessary, an exchange

[1] The price reflects the value or true worth of a good if it brings about an equilibrium between demand and supply.

[2] Note that the existence of a competitive equilibrium—a market equilibrium that maximizes all agents' utilities—is guaranteed only under restrictive conditions that in actual markets are not met.

[3] We propose to have one HLR and one PA per Virtual Organization.

rate between Grid credits and real currencies might be established in order to make it feasible for large computing centers to share their resources even if their users don't require Grid services in return, as well as allowing small laboratories or research groups to utilize these services even if they cannot contribute own resources.

4. Results and achievements

The Workload Management System, whose architecture and functionality have been described in the previous sections, has been implemented and deployed in the DataGrid testbed. A first working WMS [4], with limited functionality, was released in the first phase of the project. This first WMS was then reviewed and complemented, to address some shortcomings that emerged in the first DataGrid testbed (in particular some scalability and reliability problems) and to provide some new functionality ([5], [6]).

The WMS software has not been deployed only in the DataGrid testbed. It was also used (and it is still used) in several other environments, such as:

- The LCG Grid.
  The DataGrid WMS software has been deployed in the LCG Grid infrastructure, the Grid of the LCG project implemented to meet the unprecedented computing needs of the LHC HEP experiments. This LCG infrastructure currently spans about 20 sites (in Europe, America and Asia), but the number of sites and resources keeps growing rapidly.

- The DataTAG testbed.
  In this testbed in particular the interoperability with US Grid domains was successfully addressed: job submissions using the DataGrid WMS, including to US resources, was demonstrated.

- The GRID.IT infrastructure.
  The DataGrid WMS software has been deployed in the Italian GRID.IT infrastructure. This now comprises 13 sites, but it is growing to include other resources belonging to different Italian scientific organizations. Currently the GRID.IT Grid is used to run typical HEP applications, but in the next future new users and new applications (applications in the domains of volcano seismology, climate model development, genomics, proteomics and astrophysics) will be involved.

– The CrossGrid testbeds.
  The DataGrid WMS software has also been used by the CrossGrid project, where the WMS software has been customized and enhanced according to the specific requirements of the project.

Extensive and stress testing was performed, in the first phase by the DataGrid applications groups in their quasi-production testing activities [8]. Now the services on a single WMS service node, deployed in the LCG Grid infrastructure, are routinely tested: storms of 2000 requests from 20 concurrent users are submitted daily to certify the reliability and robustness of the system.

5. Related Work

The DataGrid WMS is characterized by the complex coverage of the many facets of the workload management.

While increased number of Grid related projects and activities uses portals as the primary input point into the Grid (see, e.g. [30] or the NPACI HotPage, `https://hotpage.npaci.edu`), the DataGrid command line user interface proves very useful when large job sets are to be submitted. It also enables off-line preparation of job submission, requiring on-line connectivity only at the actual job submission time.

The GridLab project attempts to use multicriteria scheduling strategies [18] which require a super-scheduler with full knowledge and control over the resources. DAGman/Condor-G is used in GriPhyN [12] as a low-level scheduler, complex workflows are taken care by the Pegasus subsystem. General workflow scheduling problem studied in [21] poses too restrictive conditions on the knowledge of the current Grid state required by the scheduler that it still must be tested within some production Grid environment (currently within the GridLab project). GrADS [11] uses a simple launch-time scheduler which can block, accompanied with a metascheduler, capable of negotiating scheduling contracts. The approach adopted by the DataGrid WMS while being multicriterial and supporting workflows, performs well even in the presence of outdated information and does not rely on full knowledge of the Grid state.

GrADS also supports job rescheduling via swapping, tailored to MPI jobs. The DataGrid WMS handles job recovery in presence of failures via its job partitioning and checkpointing support. Working at the application level, it can support generic jobs, overcoming global synchronization problems faced by automated checkpointing support for parallel (MPI) jobs [9].

Other Grid projects are developing infrastructures implementing economic oriented Grid accounting (like the GASA GridBank system [7]) while other groups are focusing mainly on reporting usage metrics [17]. The DataGrid DGAS, while being developed mainly to implement economic accounting, is meant to provide also resource usage metrics.

6. Conclusions and Future Directions

In this paper the details of a Workload Management System, designed and implemented in the context of the DataGrid project, were presented. Feedback on this software system, deployed and used in many environments besides the DataGrid testbed, has been quite good, not only in terms of the provided functionality, but also for the performance and robustness of the software.

In any case, although significant results have been achieved, the problem of distributed processing of data-driven jobs using standard, general purpose components is of course not completely solved. Many areas appear to still require attention, such as:

- The organization of the resource information: the collection of the various information concerning Grid resources, the organization of this information, its indexing, and its possible optimization for match-making use are still an open issue.

- Resource access: mechanisms providing access to computing resources as they become available, needed to accommodate the process of match-making/brokering at the latest possible time, should be provided.

- The handling of data requirements: the co-location of the job and at least the bulk of data to be accessed, the management of disk space (allocation and cleanup) and the staging of any output need further investigations.

- Distributed super-scheduling: mechanisms for scheduling between multiple Resource Brokers, to increase the scalability and reliability of the Workload Management System, and to avoid possible bottlenecks, should be investigated.

Moreover the evolution of the set of middleware components towards emerging standards, based on service oriented architectures, is another activity were future work will have to focus.

Acknowledgements

DataGrid is a project funded by the European Commission under contract IST-2000-25182.

We also acknowledge the national funding agencies participating to DataGrid for their support of this work.

References

1. 'Home page of the DataGrid project'. `http://www.edg.org`.
2. Alfieri R. et al.: 2004, 'VOMS, an Authorization System for Virtual Organizations'. In: *Grid Computing, First European Across Grids Conference.*
3. Andreozzi, S., M. Sgaravatto, and C. Vistoli: 2003, 'Sharing a conceptual model of grid resources and services'. Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003.
4. Anglano C. et al.: 2001, 'Integrating Grid tools to build a Computing Resource Broker: activities of DataGrid WP1'. Talk from the 2001 Computing in High Energy and Nuclear Physics (CHEP01), Beijing, China, September 2001.
5. Avellino G. et al.: 2003a, 'The EU DataGrid Workload Management System: towards the second major release'. Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003.
6. Avellino G. et al.: 2003b, 'The first deployment of workload management services on the EU DataGrid Testbed: feedback on design and implementation'. Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003.
7. Barmouta, A. and R. Buyya, 'GridBank: A Grid Accounting Services Architecture (GASA) for Distributed System Sharing and Integration'. In: *Proceedings of the 17th Annual International Parallel & Distributed Processing Symposium (IPDPS 2003) Workshop on Internet Computing and E-Commerce.*
8. Bonacorsi D. et al.: 2003, 'Running CMS software on GRID Testbeds'. Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003.
9. Bronevetsky G. et al.: 2003, 'Automated aplication-level checkpointing of MPI programs'. In: *ACM Symposium on Principles and Practice ofParallel Programming.*
10. Cooke A. et al.: 2003, 'Relational Grid Monitoring Architecture (R-GMA)'. Presented at UK e-Science All-Hands meeting, Nottingham, UK. `https://edms.cern.ch/file/400756/1/rgma.pdf`.
11. Dail H. et al.: 2003, 'Scheduling in the Grid Application Development Software Project'. in [22], pp. 73–98.
12. Deelman, E., J. Blythe, Y. Gil, and C. Kesselman: 2003, 'Workflow Management in GriPhyn'. in [22], pp. 99–118.
13. Foster, I. and C. Kesselman: 1999, 'The Globus Toolkit'. In: I. Foster and C. Kesselman (eds.): *The Grid: Blueprint for a New Computing Infrastructure.* San Francisco, CA: Morgan Kaufmann, pp. 259–278. Chap. 11.
14. Foster, I., C. Kesselman, G. Tsudik, and S. Tuecke: 1998, 'A Security Architecture for Computational Grids'. In: *Proc. 5th ACM Conference on Computer and Communications Security Conference.*

15. Frey, J., T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke: 2001, 'Condor-G: A Computation Management Agent for Multi-Institutional Grids'. In: *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*. San Francisco, California, pp. 7–9.
16. Giacomini, F., F. Prelz, M. Sgaravatto, I. Terekhov, G. Garzoglio, and T. Tannenbaum: 2002, 'Planning on the Grid: A Status Report'. Technical Report INFN-TC-02/26, INFN-GRID Project.
17. Jackson S. et al.: 2003, 'Charter of the Global Grid Forum Usage Records Working Group'. Technical report.
18. Kurowski, K., J. Nabrzyski, A. Oleksiak, and J. Weglarz: 2003, 'Multicriteria Aspects of the Grid Resource Management'. in [22], pp. 271–294.
19. Křenek, A. and Z. Salvet: 2003, 'L&B API Reference'. DataGrid-01-TED-0139. `http://lindir.ics.muni.cz/dg_public/`.
20. Litzkow, M., M. Livny, and M. Mutka: 1988, 'Condor - A Hunter of Idle Workstations'. In: *Proceedings of the 8th International Conference of Distributed Computing Systems*.
21. Milka, M., G. Waligora, and J. Weglarz: 2003, 'A Metaheuristic Approach to Scheduling Workflow Jobs on a Grid'. in [22], pp. 295–320.
22. Nabryzski, J., J. Schopf, and J. Weglarz (eds.): 2003, *Grid Resource Management*. Kluwer Academic, Norwell, MA.
23. Novotny, J., S. Tuecke, and V. Welch.: 2001, 'An Online Credential Repository for the Grid: MyProxy.'. In: *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*.
24. Piro, R., A. Guarise, and A. Werbrouck: 2003, 'An Economy-based Accounting Infrastructure for the DataGrid'. In: *Proc. of the 4th International Workshop on Grid Computing (Grid2003)*. Phoenix, Arizona, USA.
25. Raman, R., M. Livny, and M. Solomon: 1998, 'Matchmaking: Distributed Resource Management for High Throughput Computing'. In: *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*. Chicago, IL.
26. Raman, R., M. Livny, and M. Solomon: 2000, 'Resource Management through Multilateral Matchmaking'. In: *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*. Pittsburgh, PA, pp. 290–291.
27. Schopf, J. M.: 2001, 'Ten Actions When SuperScheduling'. Technical Report GFD-I.4, Global Grid Forum, Scheduling Working Group.
28. Simmel D. et al.: 2004, 'Charter of the Global Grid Forum Grid Checkpoint Recovery Working Group'. Technical report.
29. Thain, D. and M. Livny: 2000, 'Bypass: A Tool for Building Split Execution Systems'. In: *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*. Pittsburgh, PA, pp. 79–85.
30. Thomas M. P. et al.: 2001, 'The GridPort Toolkit: a System for Building Grid Portals'. In: *Proc. of the Tenth IEEE International Symposium on High Performance Distributed Computing*.