Mapping "heavy" scientific applications on a lightweight grid infrastructure.

Lazar Kirchev[1]
Minko Blyantov[1]
Vasil Georgiev[1]
Kiril Boyanov[1]
Ian Taylor[2]
Andrew Harrison[2]
Stavros Isaiadis[3]
Vladimir Getov[3]
Natalia Currle-Linde[4]

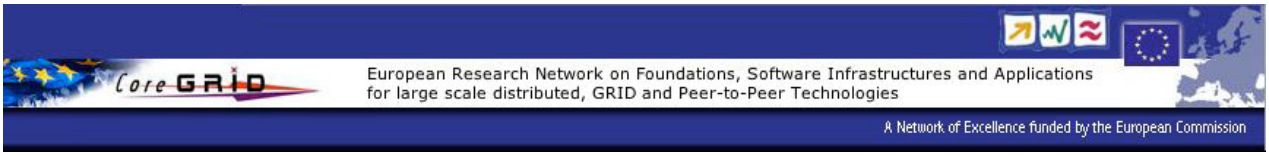
[1] Institute on Parallel Processing – Bulgarian Academy of Sciences
[2] University of Cardiff, UK
[3] Harrow School of Computer Sciences, University of Westminster
[4] High Performance Computing Center Stuttgart, Germany

This is a reproduction of CoreGRID Technical Report Number TR-0024, January 25, 2006 and is reprinted here with permission.

The report is available on the CoreGRID website, at:

http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0024.pdf

---



Mapping "Heavy" Scientific Applications on a Lightweight Grid Infrastructure

Lazar Kirchev[#], Minko Blyantov[#], Vasil Georgiev[#], Kiril Boyanov[#], Ian Taylor, Andrew Harrison*, Stavros Isaiadis[†], Vladimir Getov[†], Natalia Currle-Linde[‡]*

[#]Institute on Parallel Processing – Bulgarian Academy of Sciences
e-mail: [lkirchev, mblyantov, vasko, boyanov]@acad.bg;
**University of Cardiff, UK*
e-mail: [a.b.harrison, ian.j.taylor]@cs.cardiff.ac.uk;
[†]University of Westminster, UK
e-mail: [s.isaiadis, v.s.getov]@wmin.ac.uk;
[‡]High Performance Computing Center Stuttgart, Germany
e-mail: linde@hlrs.de;

CoreGRID Technical Report
Number TR-0024

January 25, 2006

Institute on Grid Systems, Tools and Environments

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme
Project no. FP6-00426

Mapping "Heavy" Scientific Applications on a Lightweight Grid Infrastructure

Lazar Kirchev[#], Minko Blyantov[#], Vasil Georgiev[#], Kiril Boyanov[#], Ian Taylor, Andrew Harrison*, Stavros Isaiadis[†], Vladimir Getov[†], Natalia Currle-Linde[‡]*

[#]*Institute on Parallel Processing – Bulgarian Academy of Sciences*
e-mail: [lkirchev, mblyantov, vasko, boyanov]@acad.bg;
**University of Cardiff, UK*
e-mail: [a.b.harrison, ian.j.taylor]@cs.cardiff.ac.uk;
[†]*University of Westminster, UK*
e-mail: [s.isaiadis, v.s.getov]@wmin.ac.uk;
[‡]*High Performance Computing Center Stuttgart, Germany*
e-mail: linde@hlrs.de;

CoreGRID TR-0024
January 25, 2006

Abstract

An effective security and resource management is required for the provisioning and sharing of resources while keeping autonomy and stability of their environments. The models for security control and resource provisioning and sharing are to be implemented by closely-connected modules which interact tightly with information service and stand on role-based security model. These services are local to the cluster level but all of them have the ability to interact with the according neighbours in adjacent clusters. User, security, resource and information management have their representatives on global (grid) level and they are able to interact with the same services in cluster level in order to insure global centralized control. Global services are only viable when the system is used on global or extra-grid level. In this paper we present the combined architecture of security and resource management and their close interaction with information service in a lightweight multilevel grid system based on high-level middleware. It is important to check the performance and functionality parameters of the architecture at an early phase of architecture design. That is why here we consider the possibility for mapping of typical scientific application scenarios on several commodity computing clusters running our lightweight multilevel grid middleware. A set of representative use cases is listed in this paper along with taxonomy of the imposed requirements to the underlying infrastructure. For the purpose of exemplary mapping we choose an application scenario of molecular dynamics simulation which is a typical data- and computation-intensive asynchronous application. Here is given the technical mapping of this use case to the underlying infrastructure, paying particular attention to the possibility to implement high-level grid-unawareness for the use or application developer, to identify the compatibility of logical system and technological infrastructure used, as well as service and workflow representation.

Key words

Grid Computing, Lightweight Architecture, Web Services, Distributed Systems, User/Application Scenarios, Molecular Dynamics Simulation

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1.1 Introduction

Recently it was commonly recognized that lightweight grid middleware is suitable to diverse organizations in size and domains (while this is not always the case with the extensive production grids). After studying different approaches we have created a hierarchical architecture which enables the participation of different organizations in scale and gives them the ability to be a part of one global grid system or create their own grid infrastructure. Such systems encompass a variety of geographically distributed platforms between which communication and interaction should be enabled in a transparent manner. The architecture must provide underlying system services without imposing excessive overhead and lack of functionality. Other key requirement is that the most of the grid service features must be supported by commodity workstations with possibility to extend the infrastructure with dedicated servers (e.g. for data, graphical I/O, etc.). Our approach tries to stay in tack with these requirements and grounds on the idea that each entity in the system is a service.

In the next two sub-sections, we present an overview of related work concerning the architectural and infrastructure issues as well as user scenarios issues. Further, we present our model in the section 2. Section 3. "Architecture Components" lists and describes each system module, its place in the overall architecture and how it interacts with the environment. Section 4 presents the general scenario for the user access to different grid services. Then follows a description of the scientific application of molecular dynamics used for problem-infrastructure mapping in sec. 6. The concluding remarks and the planned work are given in the end of this paper.

1.2 Related Work Overview – Grid Architecture

There are plenty of efforts to create robust and scalable systems. ProActive is a Java library for parallel, distributed and concurrent computing [6]. It provides mobility and security in a uniform framework using a reduced set of simple primitives. ProActive masks the specific underlying tools and protocols used by supplying a comprehensive API which simplifies the programming of applications that are distributed on a LAN, on a cluster of PCs, or on Internet Grids. The library is based on an active object pattern, on top of which a component-oriented view is provided. The implementation of our grid system uses Java for system-independence and its components are service-oriented simplifying its managements and component interaction. The results presented in [3, 4, 6, 7, 8] identified for us the basic properties that the system should possess. Non-functional properties: performance, fault tolerance, security, platform independence and functional properties: access to compute resources, job spawning and scheduling, interprocess communication, application monitoring. We have designed our system with these functionalities in mind, aiming to keep the system lightweight and simple for integration, management and user access to the services.

The PROGRESS project provides grid-portal architecture for further deployments in different fields of grid enabled applications [9, 10]. It consists of grid-portal environment tools implemented as open source software packages and the PROGRESS HPC Portal testbed deployment. The software architecture consists of middleware (e.g. Globus, Sun Grid Engine) as well as tools and services created within the project workpackages. The communication between these components is enabled through interfaces based on Web Services and these services are distributed within the testbed installation. We have chosen similar architecture for our solution but it differs in a couple of ways. Our system services are invisible to the user and they communicate with each other using predefined interfaces that describe the functionality of each service. The user is presented with interfaces to which the services he provides should conform to and the grid environment uses these interfaces to interact with services. These interfaces provide the means to configure the service, enter input data, start the service, monitor its execution and status and retrieve the output data. In addition to that services could be combined in order to create new services from existing ones or use meta-services provided from the framework for the same purpose.

The GridARM [2] is a dynamically extensible, scalable and adaptive system in which new protocols and tools can easily be integrated without suffering from system downtime and expensive code reorganization. In contrast to other gird projects, which are based on manual brokerage, this project provides an automated brokerage system. This automation is required especially for Grid enabled

workflows and execution environments where the brokerage process acts as a middle tier between Meta-scheduler and other Grid enabled components like Grid enabled resources and services [2]. The brokerage process is responsible to discover and allocate suitable resources for the Meta schedulers. As we aim to create a lightweight environment with a resource management service that imposes as little overhead as possible and without the need for constant human interference for tuning its performance, we agree that it is imperative to design it scalable and autonomous. The introduction of role-based security model in our architecture to be used for access control to resources eases the automatic decision taking from Resource Management Service.

1.3 Related Work Overview – Scientific User Scenarios

The aim of developing new infrastructures is not only to enable users to achieve their aims faster, cheaper, and more accurately, but to allow them to develop new ways of working that would not have been possible before. With these goals in mind, we have defined a number of key issues that user scenarios expose. Arriving at suitable strategies for handling these issues will lead to a clearer, more flexible, extensible and inclusive design.

Grid-Awareness. A major issue is whether the application the user wishes to run is Grid aware or not. From the user's perspective, most would argue, this difference should be transparent. In some cases legacy applications need to be 'gridified' without changing the behaviour from the user's point of view. In other cases they may need to be wrapped entirely, for example as a Web service.

User Interface. As Grid scenarios become more sophisticated, so the user interfaces must keep pace. These need to cope with various underlying resource/service/workflow description technologies, many of which are still evolving, in order to render grid entities. Interfaces need to be flexible but also intuitive and simple in order to handle different user types (for example 'grid aware' users may wish to define things such as resources to use while others may not). The design of resource description mechanisms therefore needs to take these issues into consideration.

Infrastructure Used. Users may expect differing grid infrastructures depending on the scenario. For example certain applications may require highly dynamic and distributed discovery environments. Others may require server-centric data repositories. The ability to behave flexibly according to users' needs in this regard is an important aspect of developing a scalable, generic grid environment.

Middleware Used. Many existing applications already rely on a middleware layer that may be grid enabled in some way. Users and developers will be reluctant to dismantle existing capabilities in order to experiment with new technologies. It is important therefore to be able to integrate these into an inclusive grid environment, enabling diverse views of a grid to co-exist.

Service Representation. With new Grid technologies moving towards the service oriented paradigm, shared views of service representation need to be developed. Furthermore, while there are existing standards of service representation and communication with a broad base of acceptance (WSDL and SOAP for example), this area is still in an evolutionary phase. Emerging technologies which are either richer or more efficient need to be able to be integrated when they achieve maturity.

Workflow Requirements. Workflow is becoming more and more important in Grid user scenarios, in part due to the adoption of SOA which views the network as discreet entities providing defined services. Understanding the workflow requirements of users' scenarios will help in defining generic mechanisms for describing and implementing them.

Runtime Requirements. The ability to monitor/steer/migrate running applications is paramount in optimising not only application performance, but user performance as well. These requirements become more complex to implement as the underlying distributed topology becomes more complex.

We have elicited a number of user scenarios from projects affiliated with CoreGrid:

1. Library Level Legacy Code Wrapping (UWC and FORTH-ICS)
2. Weather forecast application (GEMLCA (http://www.cpc.wmin.ac.uk/gemlca/) and P-GRADE (http://www.lpds.sztaki.hu/pgportal/))
3. Urban traffic simulation (GEMLCA (http://www.cpc.wmin.ac.uk/gemlca/) and P-GRADE (http://www.lpds.sztaki.hu/pgportal/))

4. Hospital Information System (UOW)
5. Chemical application - using GAMESS (UPC and Universidad de Castilla La Mancha (UCLM, SPAIN)
6. GRID superscalar applications using BLAST (UPC)
7. Molecular Dynamics Simulation (HLRS)

We have chosen one particular scenario – the Molecular Dynamics Simulation (HLRS) for a number of reasons:

1. It is both computation and resource intensive.
2. It requires workflow
3. The user should not have to be conversant with grid technologies
4. It requires a sophisticated user interface
5. It requires monitoring and steering capabilities

Section 6. describes how we map the scenario to our Grid environment.

2. Service Oriented Architecture

Each participating party in our system is represented by a cluster – its grid system and all clusters are embraced in a global grid. This present us with a hierarchical multilevel architecture as suggested in models presented in [1].

The central idea of our research is to create a lightweight scalable solution with simple and effective modules. We based our model on Service Oriented Architecture paradigm. After going through different designs and similar grid systems we have identified the cornerstone services of our model and its functionalities. The entry point to the system is the grid portal. Both cluster and global layers have their portals and they are much the same in the services they provide. The system services that we have identified are:

1. Security Service/User Management
2. Resource Manager Service/Task Scheduler
3. Information Service
4. Monitoring Service
5. Accounting Service

6. Node Service

Each system service has a cluster and a global version – Fig.1. Each one performs its duty in the context of a cluster or grid. Cluster system services interact with each other within the cluster and have the ability (if they are a part of a global grid) to interact with the corresponding services from other clusters. Global grid system services provide their functionalities in terms of a collection of clusters. They interact with each corresponding cluster service to present their results in terms of global grid infrastructure.

The basic functionalities that our system provides are:

1. Library of service prototypes, meta service support and code wrappers
2. Directory of active services
3. Structured service description/advertisement
4. User profiling and accounting
5. Audit information about services, users and hardware related parameters (e.g. CPU-time consumed etc.)
6. Administrative and user GUI will be supported by cluster and grid portals

In context with the features listed above in our system each resource is accessed through a service and thus represented by a service and a task for execution is a service with the supplied data and selected and reserved resources. This makes the service model fundamental to our architecture.

The management of resources in the grid environment becomes complex as they are geographically distributed, heterogeneous in nature, owned by different individuals/organizations and have different access-and-cost models [1]. Striving to provide a simple and scalable solution we have chosen to group services by the type of resources they use. In [4] most common and basic grid resources identified are:

1. Compute
2. Storage
3. Data
4. Network
5. Software
6. Services

Our model follows these conclusions and makes these types of resources a corner stone of the resource management service. In order to afford an architecture which is scalable and has the ability to automatically control resources, tasks execution and users management we have decided to use the role-based security model as central to managing the usage of services by users and consumption of resources by services. Thus, the rights of each user in the system are identified by the roles he possesses and each service specifies the roles it needs (what resources it will use) for execution and to what user roles it grants access (Fig. 2.). This is a coarse-grain management which means that the system matches only roles for fast resource discovery and access permitting. This is shown on Table 1. – the roles follow the resources types we have listed divided into three groups by the extent to which they are used. This presents us with 18 roles.

Resource Type\Resource Usage	***Weak***	***Normal***	***Strong***
Compute	WeakComputeUsage	NormaLComputeUsage	StrongComputeUsage
Storage	WeakStorageUsage	NormaLStorageUsage	StrongStorageUsage
Data	WeakDataUsage	NormaLDataUsage	StrongDataUsage
Network	WeakNetworkUsage	NormaLNetworkUsage	StrongNetworkUsage
Software	WeakSoftwareUsage	NormaLSoftwareUsage	StrongSoftwareUsage
Services	WeakServicesUsage	NormaLServicesUsage	StrongServicesUsage

Table 1. - Roles

Fig. 2. Service access and resource requirements and service description

The service can not grant weaker access than it needs to resources. Users with higher access rights could use services that require weaker access. If the service or resource manager requires fine-grained control or selection of resources it could use the service description provided with it. The user or administrator could specify the preferred way for a service for resources discovery and allocation – by Usage Roles, by Service Description or both.

User's roles determine what resources and with what access type he could use resources/services. The system has these types of users:

1. System Grid Administrator – has administrative rights to the whole grid.
2. System Cluster Administrator – has administrative rights only to the cluster in which he resides.
3. User with Cluster access – has access only to the local cluster where he resides.
4. User with Cluster and Grid access – has access to his local cluster and the global grid.
5. External Grid users – users who are not members of any cluster.

Users with access to Grid have additional Grid Roles stored in Grid Information Service and accessible through Grid Security Service (Fig.3.).

3. Architecture Components

The Grid/Cluster portals are the corresponding entry points to our grid model. They represent the user with a front-end where he could log into the system if he is successfully authenticated by the proper Security Service – Grid Security Service if he is logging on the grid portal or Cluster Security Service if he

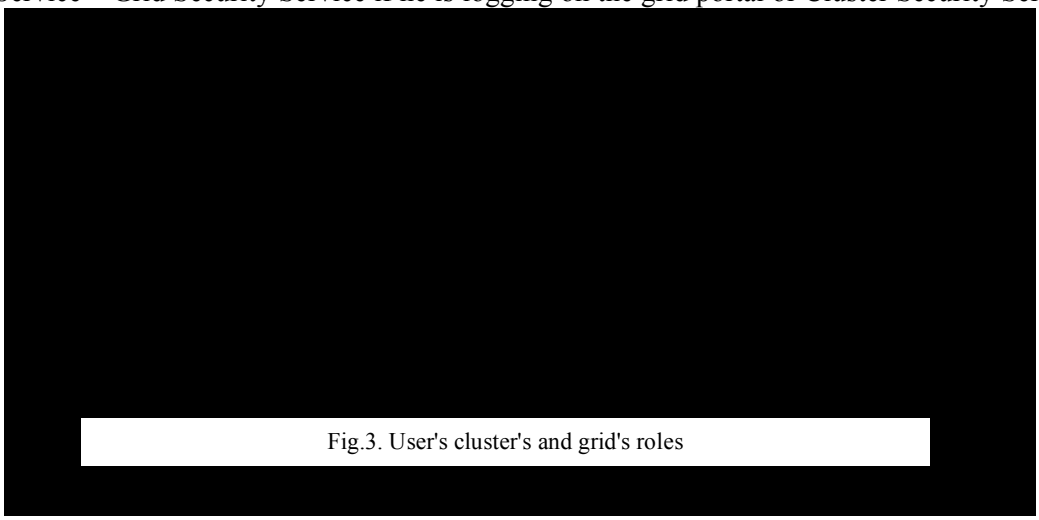


Fig.3. User's cluster's and grid's roles

is logging on the cluster portal (Fig. 4.). The portal lists the active services to which the user has access rights and he could browse them or he could browse the repository with inactive services that the grid/cluster provides access to.

Security Service (SS) authenticates users and provides access information for users and services. Other system services could query Security Service for users' rights confirmation and validation. Data used by SS is stored in Information Service and security sensitive data is accessible only from SS. This service is duplicated on cluster and grid level and each controls its domain. Both interact when they need to provide security functionalities between clusters. We have decided to implement in our grid a role based security model, as in [11, 12, 13]. A Role Based Security Model (RBAC) is also used in the PERMIS System for user authorization [14]. In this model, a number of roles are defined in the grid and rights are associated with roles, not with particular users. This approach has the advantage that there is no need to assign rights to every user separately – the user is assigned one or more roles and he / she automatically receives the rights with them. We consider this a scalable solution. Moreover, the node manager of every cluster node may impose further restrictions upon the access to the resources, which it controls. In this way an additional flexibility of defining access policy is gained. For security at the level of code execution sandboxing will be used, similar to the approach taken in the AliCE grid system [15, 16]. Thus data and code security are guaranteed by the implementation technology that will be used, namely Java.

Resource Management Service (RMS) Is responsible for:

- Resource Discovery – issues queries to Information Service for the necessary resources – matching the user roles, service roles (verify access to resource information) and if noted it could use service description for more refined resource filtering. RMS selects proper resources for job execution based on service configuration and resource usage statistic. If there are no available resources the task could be planned for later execution if the user agrees to do so (reservation management).
- RMS provides functions to monitor jobs' status for which the Monitoring Service (MS) keeps track of.
- Task Scheduler Service is a sub-module of Resource Manager. Tasks could be started immediately, reallocated for execution on specific node or planned for later or exact time execution. Our architecture enables users to create new services from existing services or meta services linking them – input of one service with the output of another. The system allows only matching input-output combinations in means of data type used. The user is presented with a proper front-end to build and order the execution of services.
- Accounting information is accessible through RMS and is updated by MS.

Data about services, service descriptions, user information, node status and information is kept in the **Information Service** (IS) and used from the other system services like Security and Resource Management. Accounting information and usage statistics for services and nodes' resources are stored in Information Service too.

Node Service (NS) is a piece of software that makes a node part of a cluster and thus of the whole grid infrastructure. It keeps track of node's resources and capabilities, interacts with the Monitoring Service (MS) to update information, statistics and status for that node. MS stores the proper data into IS. NS starts the real execution of tasks. RMS allocates tasks to particular nodes based on resource selection algorithm. NS provides functions for other system services to interact with it in order to monitor job execution, suspend or stop execution and find out tasks that have failed and need to be restarted or reallocated.

Monitoring Service (MS) updates status for each service into the Information Service. Monitoring service polls nodes comprising the cluster for their status and they could inform it too if there is a change in the status. The same applies for tasks in execution.

When a user selects a service, a GUI is shown based on the service description (structured XML). It lists the features of the service what it performs, what are the input parameters, output parameters, resource requirements, cost of usage and its status. Then if the user chooses to create a task for execution he enters the proper input data and he could specify if the task should be executed immediately, as soon as possible or postpone the execution for later period when the needed resources are available. The user could further limit his requirements for desired resources and cost and that will be taken into account by Resource Manager Service. RMS accepts the request from the user and issues a query to Information Service for the necessary resources. Based on the choices of the user to use available or wait for available resources a task is created and ready for execution. The user could monitor the status of his running or pending tasks from the portal.

4. Grid Access Use Cases

In this section we present two use-cases which demonstrate how our system functions: the submission of a user task and the user access to one or more persistent services.

First we consider the case of job submission (we may call this using a non-persistent service), and after that – the case of using a service (persistent service – it is offered by the grid system). As can be seen, the work done by the system in both cases is similar.

Suppose a user wishes to submit a job to the cluster portal. Following are the steps that are usually performed in this case.

1. First of all, he or she must log on the grid. This is done at the grid or cluster portal. After successful logon, the user has a token with his or her identity and other information, such as the user's roles. Now the user may submit the job.
2. After the job is submitted, the portal contacts the Cluster RMS (CRMS), which is responsible for scheduling the job for execution on particular resources.
3. The CRMS may send request to the Cluster Security Service (CSS) for confirmation that the user really has the roles listed in the token. If the roles are confirmed, the CRMS proceeds with the scheduling. This additional step may enhance security in case of a forged token.
4. The CRMS uses the Cluster Information Service (CIS) in order to find the possible resources. The CIS will search its database for resources, which meet the requirements of the task. Since in the description of every resource the roles, which have access to it will be listed, the CIS will be able to filter the resources and return to the Broker only those, to which the user has access.
5. After the results reach the CRMS, it decides which of them to use for the job (if there are more than needed) and partitions the job among them in case the job may be parallelized.
6. The CRMS contacts the chosen resources' Node Managers. They should confirm that the resources are free at the moment and may be used. The Node Managers also check whether there are any additional restrictions on the user's roles, which do not allow the execution of the task on the node. If there are no such local restrictions the job is started.

In the above scenario, a user cannot request directly a resource – every request is issued by the CRMS.

The situation is similar if the user contacts the Grid Portal and not a Cluster Portal. In this case after the job is submitted, the request is forwarded to the GRMS (here, too, the GRMS may request confirmation of user roles from the GSS), which contacts the Grid IS (GIS) for resources for execution of the job. The GIS communicates with CIS of the clusters in order to find available resources. After resources are found, the task is forwarded to the proper CRMS.

Both when a task is submitted to the Grid Portal and a Cluster Portal, it is possible if there are no available resources at the moment the task to be scheduled for a future moment.

In addition to submitting jobs to the grid, the user will be able to use other resources and services. All resources will be presented as services, so using resources will be in fact using services. These services may be called persistent services – they are offered by the grid system and are not submitted by the users. Now suppose that the user wants to use a service. After the user is successfully authenticated, the user sees all services to which he or she has access. Now the user finds the service that he or she wishes to use. The cluster portal forwards the user request for the service to the CRMS and from now on the steps are the same as in the previous use-case. If the user contacts the grid portal, then the request is processes by the GRMS, in a similar way to the previous use-case.

5. GRID Application Scenario: Molecular Dynamics Simulation

Scientific User Scenario. The problem is to establish a generic molecular model that describes the substrate specificity of enzymes and predicts short- and long-range effects of mutations on structure, dynamics, and biochemical properties of the protein. A molecular system includes the enzyme, the substrate and the surrounding solvent. Multiple simulations of each enzyme-substrate combination need to be performed with ten different initial velocity distributions. To generate the model, a total of up to of 3000 (30 variants x 10 substrates x 10 velocity distributions) MD simulations must be set up, performed and analyzed

Each simulation will typically represent 2 ns of the model and produce a trajectory output file with a size of several gigabytes, so that data storage, management and analysis become a serious challenge. Each simulation can typically require 50 processor days for each simulation. These tasks can no longer be performed interactively and therefore have to be automated.

The scientific user requires an application which is user-friendly (requires no specific programming or GRID knowledge) and can deliver and manage the required computing resources within a realistic time-scale. Such an application requires a workflow system with tools to design complex parameter studies, combined with control of job execution in a distributed computer network. Furthermore, the workflow system should help users to run experiments which will find their right direction according to a given criteria automatically.

This case of Molecular Dynamics Simulation has been mapped to SEGL (Science Experimental Grid Laboratory) – a Problem Solving Environment which has been used to solve a wide range of application scenarios in different fields such as statistical crash simulation of cars, airfoil design and power plant simulation. These scenarios have been tackled using extensive computing resources and parallelization. This section describes in more detail the application scenario of Molecular Dynamics simulation.

SEGL is a GRID aware application enabling the automated creation, start and monitoring of complex experiments and supports its effective execution on the GRID. The user of SEGL does not need to have the knowledge of specific programming language and knowledge about of GRID structure.

SEGL allows the description of complex experiments using a simple graphical language.

Figure 5. shows the system architecture of the SEGL. It consists of three main components: the User Workstation (Client), the ExpApplicationServer (Server) and the ExpDBServer (OODB). The system operates according to a Client-Server-Model in which the ExpApplicationServer interacts with remote target computers using a Grid Middleware Service such as UNICORE and SSH. Integration with Globus is planned for the future. The implementation is based on the Java 2 Platform Enterprise Edition (J2EE) specification and JBOSS Application Server. The database used is an Object Oriented Database (OODB) with a library tailored to the application domain of the experiment.

SEGL consists of two main parts: Experiment Designer (ExpDesigner), for the design of the experiment, and the runtime system (ExpEngine).

From the user's perspective, complex experiment scenarios are realised in Experiment Designer using a simple graphical system of icons and nested windows to represent workflow. The technical mapping from this user perspective to the underlying infrastructure is carried out via the use of three levels: control flow, data flow and data repository.

The control flow level is used for the description of the logical schema of the experiment. On this level the user makes a logical connection between blocks: direction, condition, and sequence of the execution of blocks. Each block can be represented as a simple parameter study. The data flow level is used for the local description of interblock computation processes.

On the experiment data repository level, a common description of the metadata repository is created. The repository is an aggregation of data from the blocks at the data flow level.

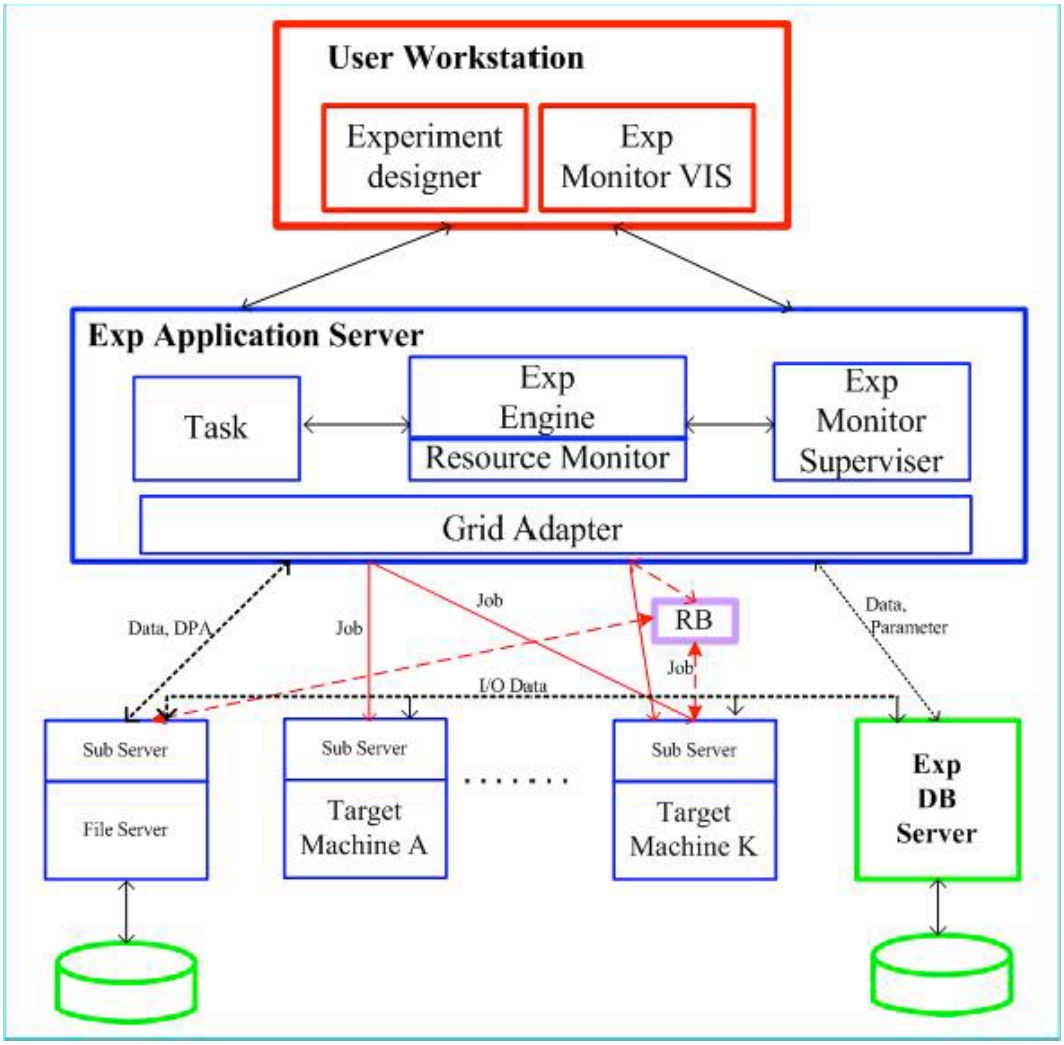


Fig. 5. System Architecture of SEGL

The Runtime System of SEGL (ExpEngine) chooses the necessary computer resources, organizes and controls the sequence of execution according to the task flow and the condition of the experiment program.

The user sets up initial conditions and parameters using the graphical icon language of SEGL. Then SEGL monitors and steers the experiment, informing the user of the current status. SEGL employs a dynamic parameterization capability which requires an iterative, self-steering approach. The communication between the Experiment Designer and the runtime system supports the monitoring of execution of the experiment.

6. Mapping of Scientific User Scenario to Lightweight Grid (LWG) Infrastructure

Similar scientific scenarios may be performed by our lightweight grid infrastructure. The program that performs the molecular modeling will be represented as a set of non-persistent application services in our grid system. In this case a Master-Workers application distribution model will be implemented. The master

application service will support two interfaces: the user interface and the interface to the rest of the services. All of these services have to be submitted to a portal (either the grid portal or one of the cluster portals) for co-scheduling and execution. Actually, for our architecture the master and worker services are just services for execution.

Depending on the developer's choice, the overall application may be grid-aware or grid unaware. The difference between these two approaches lies only in the implementation of the master application service – we have GAMs or GUMs respectively.

A grid-aware master (or GAM) supports one more interface – it interfaces the portal. GAM is responsible for the problem decomposition and its granularity, so that it can decompose the problem domain in different number of subdomains. The decomposition task can be done by GAM using only the domain attributes (e.g. domain size and domain structure, represented by the type of parameter studies. However, the application developer may choose to design GAM that negotiates with the portal the actual parameters of the grid environment (cluster-wide or grid-wide ones) prior to making the decomposition decisions. Then, using its grid or cluster portal interface, it submits the corresponding number of tasks as application services providing them with the appropriate metadata. The collection of the results (or the report of their location) is obviously the responsibility of the master service – either GAM or GUM.

Choosing the grid-unaware master (GUM) approach releases the application developer from the necessity to integrate a grid interface in the master code. As a matter of fact, the distributed Master-Workers application is ready to be submitted to the portal "as is", without further redesigning – just providing the metadata for the two types of services – the master and workers. The price of such a convenience is paid as usual in potential loss of performance.

The master service (no matter whether GAM or GUM is used) offers a simple and intuitive Graphical User Interface, which describes the service's features – what it performs, what input parameters are needed, its resource requirements. The user may choose from the GUI with what combination of parameters (enzyme-substrate combinations and velocity) the experiment should be carried out. It also should be possible to choose a sequence of parameter combinations for consecutive simulations, which are to be performed one after another. The user may further customize the requirements for resources – e.g., choose a greater number of CPUs for execution, or particular nodes, on which the service should be executed. The user may choose whether the service should start execution right now, or be scheduled for a later moment.

The services (both master and workers) are described with metadata tags. This metadata is represented by XML descriptors, which list the requirements of the service – number of CPU needed for execution, amount of memory, disk space, input and output data structures or requirements (e.g. graphical device output, printers, etc.).

In our middleware the workflow requirements are represented by metaservices and service wrappers, one of the functionalities of which is to support the transfer of a service's output [e.g. worker] to the input of one or more other services.

Monitoring of the services being executed (both master and workers) is performed by the Monitoring Service, which may query the nodes, where the job is running, for their status. This type of monitoring has system functions – for example, in case of failure appropriate measures to be taken. Further, the worker services should be able to report to the master service the work already done – e.g. number of combinations modeled, or percent of the job finished. The master service presents the user with this feedback through the GUI. The execution of a service at a node is monitored by the Node Service.

Our infrastructure is a multilevel grid, which consists of multiple clusters. That is why the simulation application may be executed either on a single cluster (i.e., on the cluster level of the grid), or in multiple clusters (i.e., on the grid level). Considering the system requirements of the simulation, a single cluster may be inadequate for its execution (unless the cluster is very large). In general case the scientific applications have to be executed at the grid level.

Following is the sequence of steps, performed for execution of the application (the case of GAM is considered, with grid-level execution):

First the GRMS loads the master service (1). After the user has entered all parameters in the GUI, the master service performs the necessary decomposition of the domain among the worker services (2) and sends these services to the GRMS (3), which should schedule them for execution. The GRMS contacts the

GIS to find available resources for the execution of the tasks (4). The GIS send requests to all CIS (5) for resources, available in the clusters (a worker service may be running on every node in every cluster, provided that it has the adequate disk space). After the resources are found, every CIS gives the GIS a list with the available resources in the cluster (6). The GIS sends this list to the GRMS (7) and it decides which resources to use. User restrictions such as choice of particular nodes may be used for this decision. Further, the GRMS contacts the CRMS of every cluster, where resources are to be allocated, and every CRMS in turn contacts every node in its cluster at which there is a resource, which will be used for the execution of the task. The GRMS sends the worker services, accompanied with the required metadata, to the appropriate CRMS (8), which forwards them to the Node Managers of the nodes (9). The node manager of every node is responsible for running the task (10). When it is finished, the Node Service (NS) should inform the CRMS. Also, the worker service, upon finishing, informs the master that it has finished.

Figure 6. illustrates this sequence of steps. The numbers in the above paragraph correspond to the numbers of the consecutive steps in the diagram.

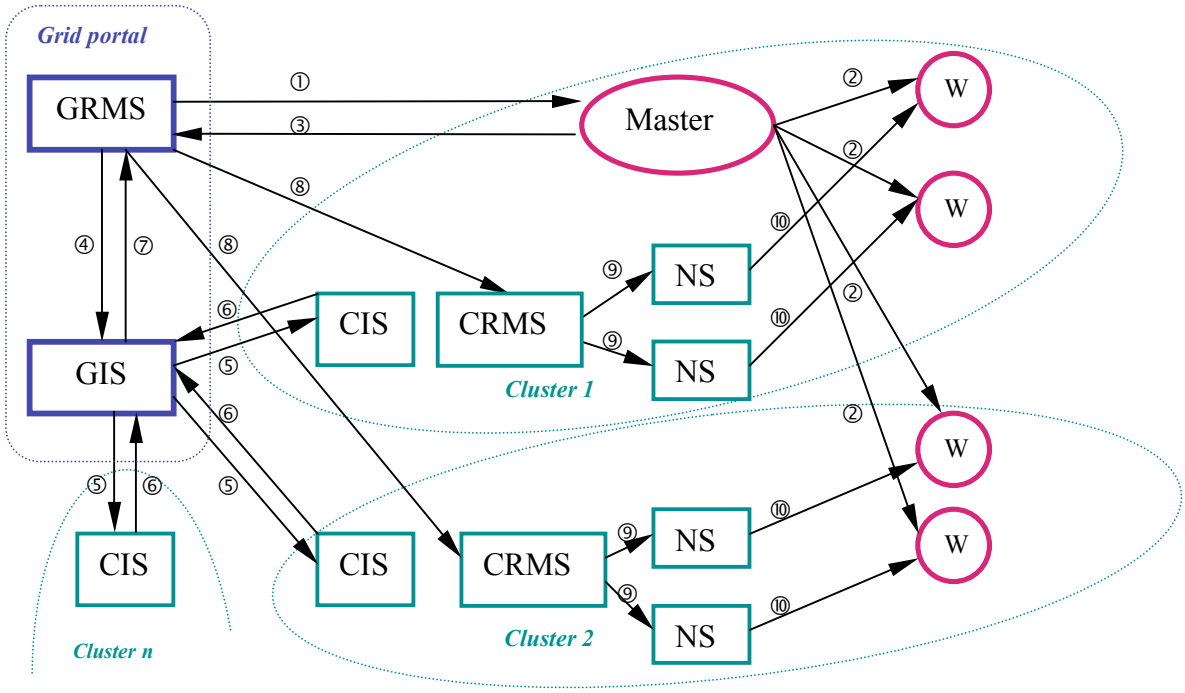


Fig. 6. Application case mapped to the clustered grid.

If the execution is in a single cluster, the process of allocation of resources and control of execution will be the same, the only difference will be that the services are submitted at a cluster portal. The GRMS and the GIS do not participate in the brokering and allocation, only CRMS and CIS of the cluster are used.

7. Conclusions and Future Work

In this paper we have presented the system architecture of a lightweight multilevel grid system. Our main purpose was to design a scalable and simple for management framework that enables quick and easy establishment of a grid infrastructure from diverse organizations. The Grid and Cluster Portals give local and external users the opportunity to easily access resources represented by different services, use persistent services shared in the grid or browse service repository for suitable to their needs pre-submitted tasks. Our model provides the ability to create services from existing and metaservices. In addition to that users could submit their own services.

The multilevel architecture enables sharing resources between different parties spread in diverse geographical locations. Each participant creates his own cluster with its own portal and they could choose to be a part of a global grid system. Underlying system components are implemented as services and have their representation in the cluster and grid layers thus simplifying the architecture and propagating the system structure through levels. This design eases the resource management activities by providing means to automate the resource discovery and allocation by incorporating role-base security model for controlling the access to services and the use of resources from the services. The whole frame work will use Java and Java communication and network technologies like Jini and JXTA. We consider this to be a scalable and platform independent solution.

Our future work is to simulate the work of the whole system and experiment with different algorithms for resource selection and allocation. We will start building a testbed and provide some example services that will give us the chance to examine the properties of our architecture in greater details and provide us with information for future improvements.

References:

[1] Rajkumar Buyya, Steve Chapin, David DiNucci, "Architectural Models for Resource Management in the Grid"

[2] Mumtaz Siddiqui, Thomas Fahringer, "GridARM: Askalon's Grid Resource management System"

[3] U. Schwiegelshohn, R. Yahyapour, „Resource Management for Future Generation Grids", CoreGRID Technical Report, Number TR-0005, May 19, 2005

[4] P. Wieder, W. Ziegler, „Bringing Knowledge to Middleware – Grid Scheduling Ontology", CoreGRID Technical Report Number TR-0008, May 19, 2005

[5] C.Ernemann, V.Hamscher, R.Yahyapour, „Benefits of Global Grid Computing for Job Scheduling", 5th IEEE/ACM International Workshop on Grid Computing, in Conjunction with SuperComputing 2004, GRID 2004 Pittsburgh; pages 374-379, IEEE Computer Society, November, 2004

[6] Thilo Kielmann, Andre Merzky, Henri Bal, Grid Application Programming Environments, CoreGRID Technical Report, Number TR-0003, June 21, 2005

[7] P. Kacsuk , N. Podhorszki, Scalable Desktop Grid System, CoreGRID Technical Report, Number TR-0006, May 24, 2005

[8] Marios D. Dikaiakos, Rizos Sakellariou, Yannis Ioannidis, Information Services for Large-Scale Grids A Case for a Grid Search Engine, CoreGRID Technical Report Number TR-0009, May 24, 2005

[9] Bogdański M., Kosiedowski M., Mazurek C., Wolniewicz M, PROGRESS USE Framework: GRID Service and Access Management within User Service Environment. Presented to the Global Grid Forum, Grid Computing Environments Research Group, September 2002 http://progress.psnc.pl/English/

[10] Michał Kosiedowski, Cezary Mazurek, Maciej Stroiński, PROGRESS – Access Environment to Computational Services Performed by Cluster of Sun Systems, Presented at the 2nd Cracow Grid Workshop, December 2002, Krakow, Poland, http://progress.psnc.pl/English/

[11] DataGrid Security Design, DataGrid Security Co-ordination Group, http://edms.cern.ch/document/344562

[12] The Security Architecture of the Jgrid System, Mark Magyarodi, Department of Informaiton Systems, University of Veszprem, accessible from http://pds.irt.vein.hu/jgrid/ documentation/JGrid_security.pdf

[13] GridLab Security Architecture (http://www.gridlab.org/ WorkPackages/wp-6/index.html)

[14] Permis project (http://www.permis.org)

[15] AliCE Grid Computing Project (http://www.comp.nus. edu.sg/~teoym/atsuma.htm)

[16] ALiCE: A Scalable Runtime Infrastructure for High Performance Grid Computing, Y.M. Teo and X.B. Wang, Proceedings of IFIP International Conference on Network and Parallel Computing, pp. xx, Springer-Verlag Lecture Notes in Computer Science, Wuhan, China, October 2004