Grid Technologies
Gridbus

Ruxandra Radulescu, AAC

1. Introduction

A "Grid" is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements. It should be noted that Grids aim at exploiting synergies that result from cooperation - ability to share and aggregate distributed computational capabilities and deliver them as service.

The next generation of scientific experiments and studies, popularly called as e-Science, is carried out by large collaborations of researchers distributed around the world engaged in analysis of huge collections of data generated by scientific instruments. Grid computing has emerged as an enabler for e-Science as it permits the creation of virtual organizations that bring together communities with common objectives. Within a community, data collections are stored or replicated on distributed resources to enhance storage capability or efficiency of access. In such an environment, scientists need to have the ability to carry out their studies by transparently accessing distributed data and computational resources. This is where the concept of resource brokers comes into picture.

The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, formerly GRIDS Lab, is a software research and development group within the Dept. of Computer Science and Software Engineering at the University of Melbourne, Australia. The CLOUDS Lab is actively engaged in the design and development of next-generation computing systems and applications that aggregate or lease services of distributed resources depending on their availability, capability, performance, cost, and users' quality-of-service requirements. The lab is working towards realising this vision through its two flagship projects: Gridbus and Cloudbus

The Gridbus Project which covers various research sub-projects that look into the management of distributed resources and scheduling of applications on global Grids. The Gridbus Project is unique in that it explores the practical application of well-known economic theories to solve resource management problems in Grids. In addition to fundamental R&D, the Gridbus Project has also partnered with various scientific, engineering, and business communities in applying Grid technologies to solve various challenging problems in e-Science and e-Business domains. The Gridbus Project/GRIDS Lab has advanced the discipline of Grid computing in the following ways:

- Carried out fundamental research in distributed resource management and application scheduling on global Grids.
- Pioneered the principles of Grid economy as well as techniques and mechanisms that enable the delivery of Grid services as utility-like services.
- Proposed several adaptive scheduling algorithms for deploying applications on global Grids based on users’ quality of service (QoS) requirements.
- Co-developed fundamental Grid technologies that enable the creation of scalable Grid environments as well as support the rapid development of Grid-enabled applications.
- Applied Grid technologies to several applications in collaboration with domain scientists, and deployed them both on national and international Grid infrastructure.

The research probes include:

- Service-Oriented Grid Architecture
- Grid Economy and Resource Management.
- Grid Service Broker
- NET based Grid Framework (Alchemi)
- Grid Workflows and Scheduling
- Service Level Agreements (SLA)-based Resource Allocation Systems (Libra).
- Grid Simulation Toolkit (GridSim).
- Resource Usage Accounting (GridBank).
- Grid Application Development Environment

- SensorGrid and Open SensorWeb Architecture.
- InterGrid for peering and internetworking between islands of Grids
- Application Targets: Drug Discovery (WEHI), Neuroscience (HFI & Osaka Uni), Kidney Modelling, Natural Language Processing, High-Energy Physics, Catchment Hydrology (eWater CRC), and Financial Investment Risk & Portfolio Analysis (Spain). The software developed as part of the

Gridbus Project has been released as open source which enables practitioners around the world to benefit from the products of the Grid research carried out at the University of Melbourne.

2. Gridbus System Vision and Architecture

Scientific discoveries and business decisions today are increasingly driven by analysis of data. Some of the target data-intensive applications that motivates our work include high-energy physics, molecular docking for drug discovery, and neuroscience. Drug designers conduct computationally intensive molecular docking technique to screen/analyse large-scale, distributed chemical databases to identify macromolecules that potentially serve as drug candidates. Businesses use various data mining techniques in decision support systems that analyse customer transaction records. In such data-intensive environments, there is a huge load on precious resources such as network bandwidth, computational and storage resources. Grid economy can be used to regulate the usage of these resources by using differential pricing strategies that provide users with incentives to trade-off lower costs for more relaxed timeframes and to use resources at off-peak hours. The Gridbus Project is investigating solutions for enabling such value-based interactions within a dataintensive computing environment. Figure 1 depicts a distributed data-oriented application scenario within which the Gridbus Project components have been deployed in conjecture with other middleware and hardware technologies.

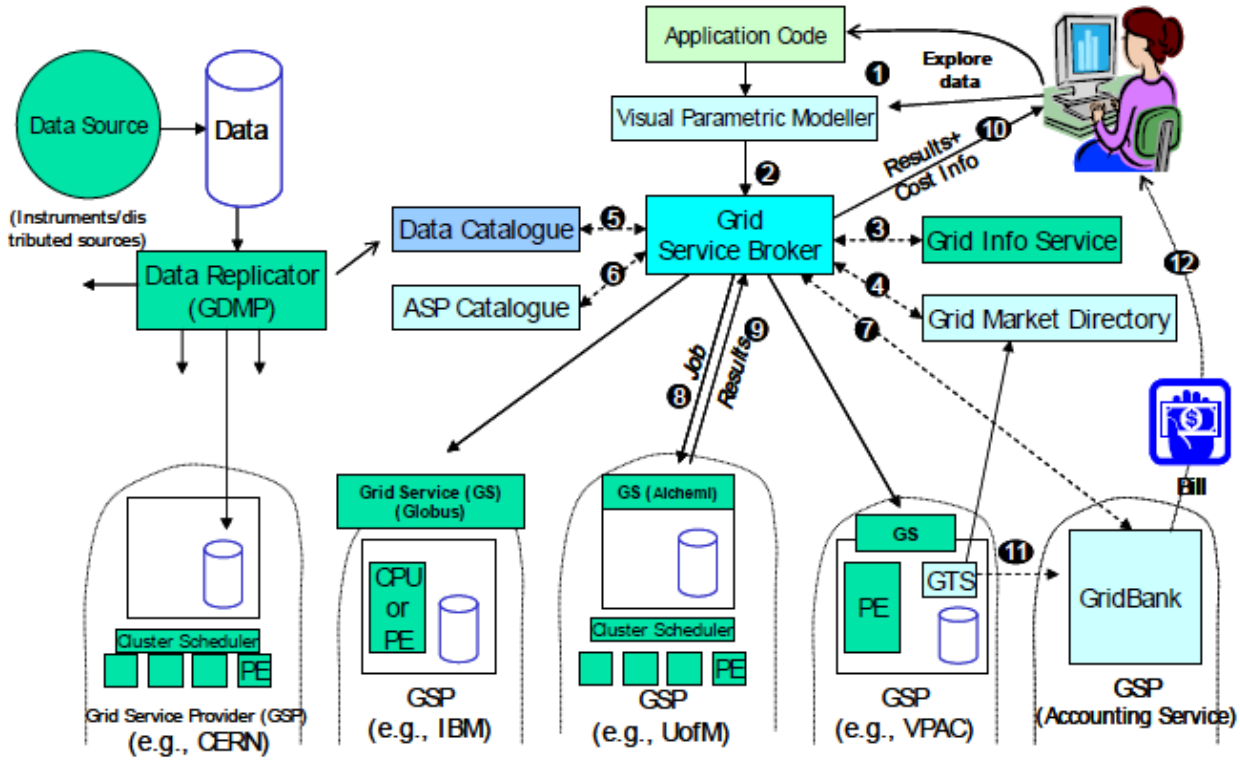

Fig 1. Gridbus architecture

The steps involved in analyzing distributed data are as follows. The application code is the legacy application has to be executed on a grid. The users compose their application as a distributed application

(e.g., parameter sweep) using visual application development tools (Step 1). The parameter-sweep model of creating several independent jobs is well suited for grid computing environments wherein challenges such as load volatility, high network latencies and high probability of failure of individual nodes make it difficult to adopt a programming approach which favors tightly coupled systems. Accordingly, this has been termed as a “killer application” for the Grid. Visual tools allow rapid composition of applications for grids while taking away the associated complexity. The user’s analysis and quality-of-service requirements are submitted to the Grid resource broker (Step 3).

The Grid resource broker performs resource discovery based on user-defined characteristics, including price, using the Grid information service and the Grid Market Directory (Steps 3&4). The broker identifies the list of data sources or replicas and selects the optimal ones (Step 5). The broker also identifies the list of computational resources that provides the required application services using the Application Service Provider (ASP) catalogue (Step 6). The broker ensures that the user has the necessary credit or unauthorized share to utilize resources (Step 7). The broker scheduler maps and deploys data analysis jobs on resources that meet user quality-of-service requirements (Step 8). The broker agent on a resource executes the job and returns results (Step 9). The broker collects the results and passes them to the user (Step 10).The metering system charges the user by passing the resource usage information to the accounting system (Step 11). The accounting system reports resource share allocation or credit utilization to the user (Step 12). In the following sections, we briefly discuss some of the Gridbus technologies shown in Figure 1.

3. Gridbus technologies

The Gridbus Project is engaged in the design and development of grid middleware technologies to support eScience and eBusiness applications. These include visual Grid application development tools for rapid creation of distributed applications, competitive economy-based Grid scheduler, cooperative economy based cluster scheduler, Web-services based Grid market directory (GMD), Grid accounting services, Gridscape for creation of dynamic and interactive testbed portals, G-monitor portal for web-based management of Grid applications execution, and the widely used GridSim toolkit for performance evaluation. Recently, the Gridbus Project has developed Windows/.NET-based desktop clustering software and Grid job web services to support the integration of both Windows and Unix-class resources for Grid computing. A layered architecture for realization of low-level and high-level Grid technologies is shown in the figure below. Some of the Gridbus technologies discussed below have been developed by making use of Web Services technologies and services provided by low-level Grid middleware, particularly Globus Toolkit and Alchemi. A summary and status of various Gridbus technologies is listed below.

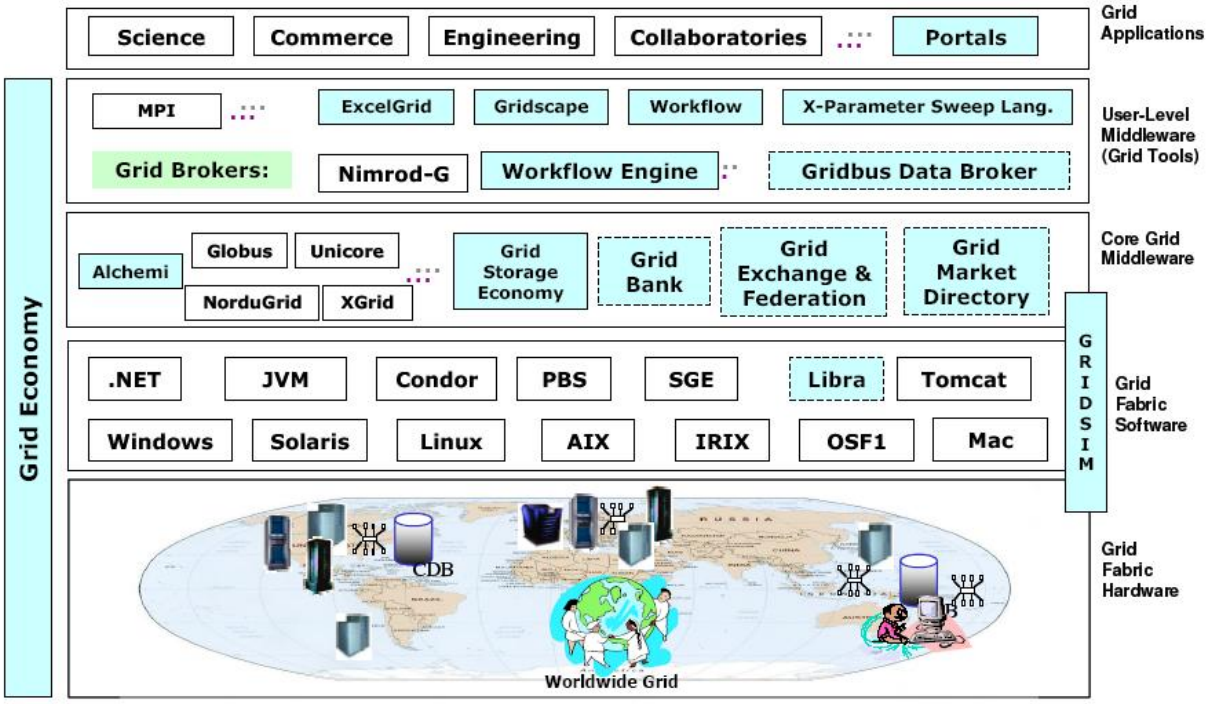

Figure 2 – Gridbus components

3.1 Visual Parameteric Modeler for Rapid Composition of Parameter - Sweep Applications for Processing on Global Grids

There exist a number of models for the construction of parallel and distributed applications. Parameter sweep is one of the simplest and most practical of the models that can yield powerful results. Parameter sweep applications consist of programs that are run independently on different nodes with different input parameters or data sets. There are numerous application areas where parametric studies find a use. Some application scenarios include:

- molecular biologist (drug designer) looking for compounds, in a large chemical data sets , that best dock with a particular protein ;
- geologist looking at the change in the density and depth of ore-body and the overlying rock's density to optimize cost and production;
- aerospace engineer understanding the role of geometry parameters in the aerodynamic design and optimization process;
- high energy physicist investigating on the origin of mass by analyzing petabytes of data generated by high-energy accelerators such as the LHC (Large Hadron Collider) ;
- Neuroscientist performing brain activity analysis by conducting pair-wise cross co-relation analysis of MEG (Magneto-Encephalography) sensors data.

The practical implications of performing parametric studies make it difficult for an application scientist, who has little or no knowledge of distributed computing, to use it effectively. The vision of the Grid is precisely to bridge this gap by providing a seamless access to compute and other scientific resources without the need of users concerning about the lower-level details of the computing infrastructure or the resource management issues

The Java based IDE called Visual Parametric Modeler (VPM) was developed as part of the Gridbus project, for rapid creation of parameter sweep applications. VPM provides a simple visual interface for the manipulation of scripts or input files of existing applications. Users can assign parameters to certain

values by highlighting them. They can select from a number of different data types and domains to describe their parameters. VPM also incorporates a task editor for creating the tasks carried out by different jobs during different stages of a distributed execution. The parameters and tasks together provide the basis of each run. VPM allows the rapid creation and manipulation of the parameters. While being flexible, it is also simple enough for a non-expert to create a parameter script, known as a plan file. The parameter sweep applications composed using VPM can be deployed on global Grids using the Nimrod-G resource broker that supports scheduling based on the user's quality of service (QoS) requirements—such as the deadline, budget, and optimization preference—and the access price of resources.

Using VPM, the users can select all application input data/configuration files and parameterize easily. The users can drag and select the value in the input file that they wish to assign a parameter to, or they can create parameters independent of an input file. This gives the user a great deal of flexibility and control. By giving the user fields to input their parameter configuration and then generating the plan specification automatically we can prevent errors. Even if the users create parameter script in their favorite editor, VPM allows them to import and make use of its capabilities. Once the plan specification is created, the users proceed to execution phase during which they have an option of changing values assigned parameters. Like enFuzion, the VPM will automatically create application jobs each with different parameter values will be created.

The visual parametric modeler architecture and parameter sweep application creation flow model is shown in Figure 3. VPM supports the creation of a new parameter sweep applications from scratch or the utilization of the existing parameterized application plans with further update. In the first case, the users can add all those files to be parameterized and use VPM to parameterize data items of interest. In the second case, the users can import the existing parametric plans and pass through the VPM scanner and parser that identify parameters and make them available for further update. The users can use the VPM task editor to create a task to be associated with jobs. Based on parameter types and their values a number of jobs, each representing a different parameter scenario, are generated automatically.

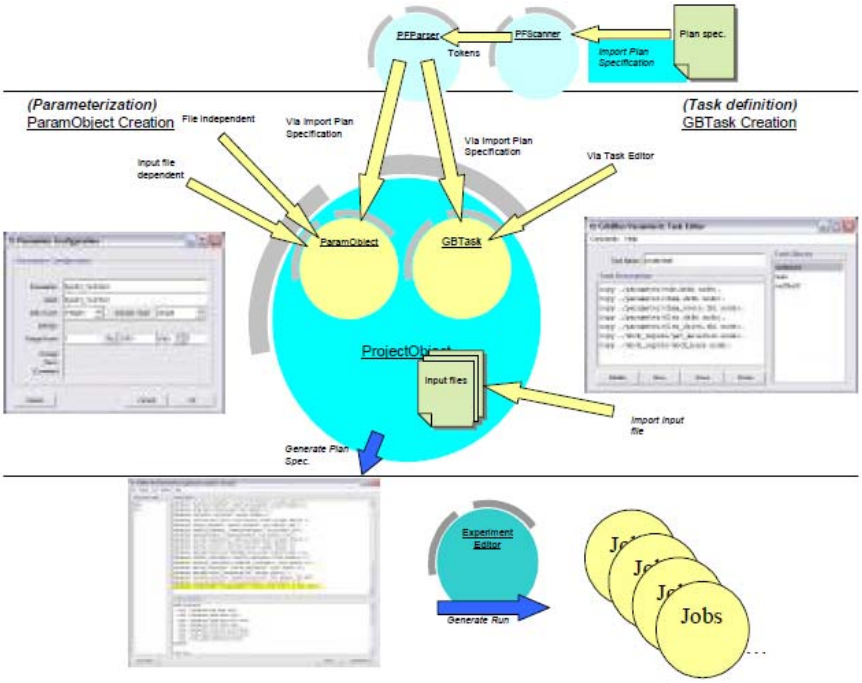

Figure 3

VPM consists of three major visual components: Project, Input Files and Tasks. These components are represented as Project Window, Input File Window and Task Editor, respectively. The design of VPM, shown in Figure 3, allows a single project to have several input data files and tasks. These visual components provide the user access to the objects that encapsulate the plan's information-model, namely to ParamObject and GBTask. ParamObject is created and manipulated from the project window or input file window, while the GBTask is created and manipulated using the TaskEditor. A plan consists of parameters and task. In VPM, parameters are internally represented as ParamObjects and tasks as GBTasks. ParamObjects are created by any of the following three methods.

- File dependent parameterization
- File independent parameterization
- Via imported plan specification

3.2 G-Monitor: Gridbus web portal for monitoring and steering application execution on global grids

The Grid consumers need an interface to the Grid environment where they will be able to monitor, control, and steer the execution of their applications (see Figure 4). This interface must provide the user with the means to access the functionality of the Grid Resource Broker (such as Nimrod-G) easily and intuitively. To provide the Gridbus toolkit with this interface the G-Monitor software package has been proposed.

Although Nimrod-G provides the user with an interface with similar functionality, it is unfortunately a heavyweight client with limited functionality and scalability problems. The Nimrod-G monitor requires the user to send their X display to the machine they wish to use it on, making it very bandwidth intensive and consequently unsuitable for wide area networks. Scalability limitations within the Nimrod-G monitor arise when the user is running a large scale experiment which involves many jobs and resources.

Taking into account the problems faced by the Nimrod-G client we have developed a Web-based G-Monitor portal.

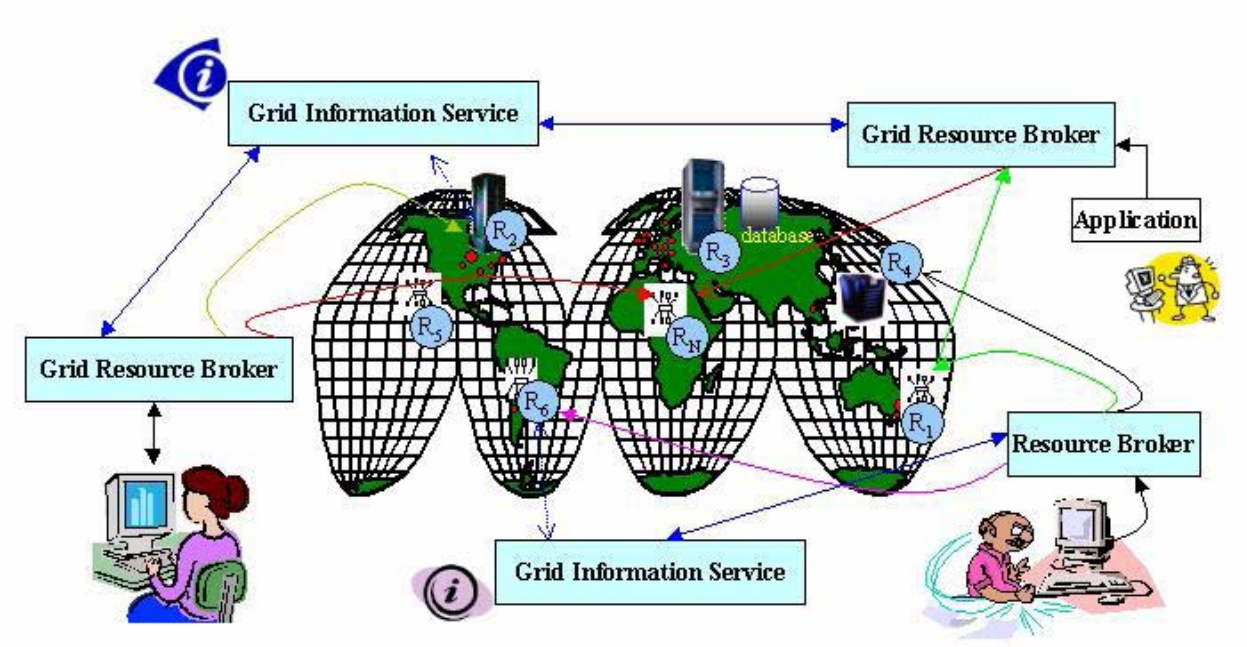

Figure 4

G-Monitor interacts with Grid Resource Broker (GRB), Nimrod-G in the current implementation, to provide the user with a GUI to the underlying Grid framework. It provides a ubiquitous interface that is easy to use, enabling the end-user to monitor and control jobs running within the Grid environment. G-Monitor is flexible enough to be run from anywhere without the need for custom client software or network overhead. G-Monitor is also scalable and therefore is able to handle thousands of nodes and jobs running in a Grid environment. G-Monitor provides a web interface for the Nimrod-G brokering system. It allows the user to remotely monitor and control a grid system. It enables the user to:

- Retrieve and set QoS (quality of service) parameters, such as Deadline, Budget, Optimization, start, stop shutdown
- Monitor/Control Jobs Information, such as Job name, status, remarks, grid node, Execution Time
- Monitor Resource status, such as Server name, Host name, Service cost, status, remarks
- Monitor Experiment status, such as Deadline, Budget, Job status, host status

The architecture of G-Monitor and its interaction with other components in a typical Grid Environment is shown in Figure 5. G-Monitor resides on a web server, which sits between the Grid Resource Broker (GRB) and the Web Browser. The Grid Consumer uses the Web Browser to access G-Monitor's functionality. G-Monitor serves the users requests by retrieving and setting information on the GRB. The GRB manages all the Grid nodes, keeping a detailed database of information about the status of the Grid nodes whilst offering scheduling and farming out facilities.

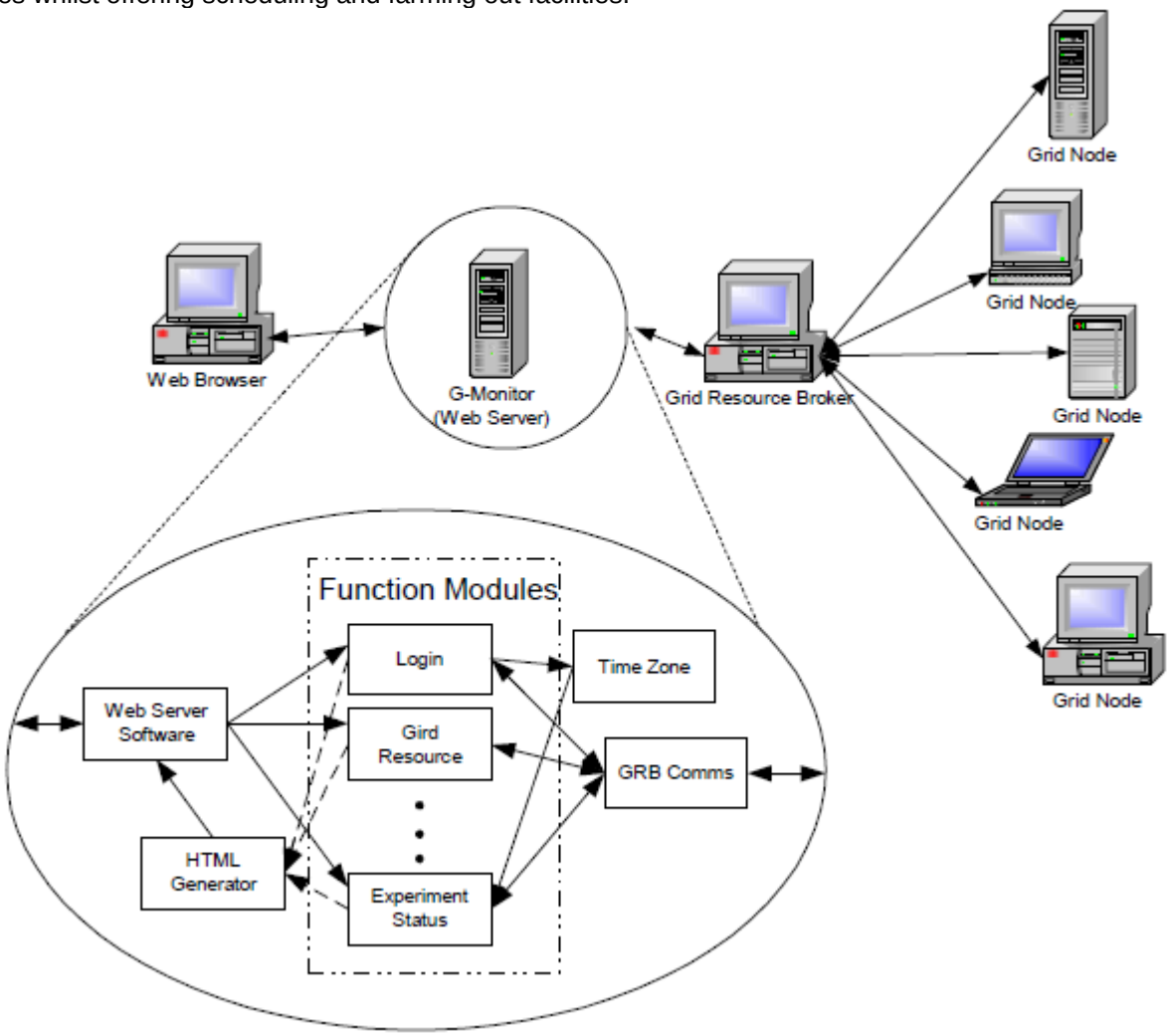

Figure 5

The five key components of G-monitor are:

- Web Server Software: Receives http requests made by the web browser, determines which of the “Function Modules” to execute and serves the HTML generated by the “HTML Generator” back to the web browser.
- HTML Generator: Receives calls from the “Function Modules” to generate HTML containing specified data. The HTML generated is then passed to the Web Server software.
- Function Modules: A set of modules which provide a one-to-one mapping with the web pages offered by G-Monitor.
- Time Zone: This module is called by the login script. Therefore, as the user is logging in, this module will register which timezone the user is coming from. The timezone module is then called by various “Function Modules” which require time conversion.
- GRB Communications: Serves requests from the “Function Modules” by establishing a TCP socket connection with the GRB to retrieve and serve requested information.

To get a better understanding of the interaction between various components we shall walk through a work flow example. The user wishes to retrieve information on the current status of their experiment and uses the web browser to click on a link to take them to the page containing this information. The following events then occur:

1. The Web browser sends the user request off to the Web Server.
2. The Web Server software receives this request and determines which of the G-Monitor “Function Module” should be executed.
3. The “Function Module” analyses the user requests and calls upon the “GRB Comms” module.
4. The “GRB Comms” module then initiates a TCP socket to the GRB server and uses the GRB protocol (e.g., Nimrod-G job management protocol) to retrieve information about the current status of the user's experiment.
5. Upon receiving the information from the GRB, the “GRB Comms” module passes this information back to the calling “Function Module”.
6. The executing “Function Modules” then makes a call to the “HTML generator” which wraps the data in HTML, and serves it back to the Web Server software.
7. The Web server software then serves the HTML back to the web browser.

3.3 Gridbus Broker

The Gridbus broker is designed to support both computational and data grid applications. For example, it has been used to support composition and deployment of neuroscience (compute intensive) applications and High Energy Physics (Belle) Data Grid applications on Global Grids. The architecture of the broker has emphasis on simplicity, extensibility and platform independence. It is implemented in Java and provides transparent access to grid nodes running various middleware. The main design principles of the broker include:

Assume ‘Nothing’ about the environment - No assumptions are made anywhere in the Broker code as to what to expect from the Grid resource except for one - that the resource provides at least one way of submitting a job and if running a flavor of Unix will provide at least a POSIX shell. Also, no assumption is made about service availability through hout an execution. The implications of this principle have a huge impact throughout the broker such as:

- The broker has no close integration with any of the middleware it supports. It uses the minimum set of services that are required to run a job on a resource supported by the middleware. The advantages of this are: In a Grid with multiple services configured differently, the broker tries to make use of every service possible by not imposing a particular configuration requirement. For example, in the case of Globus 2.4, all is required is that the GRAM service be set up properly on

the grid node. The broker can run jobs on resources with different middleware at the same time. The broker needs not to be refactored if there is a new version of the middleware.

- The broker is able to handle gracefully jobs and services failing throughout an execution. The job wrapper and job monitor code is written to handle every failure status possible. The scheduler does not fail if a service drops out suddenly.
- The failure of the broker itself is taken care of by the recovery module if persistence has been configured.

Client-centric design - The scheduler has just one target: that is to satisfy the users' requirements especially if the deadline and budget are supplied. Even in the absence of these, the scheduler strives to get the jobs done in the quickest way possible. Thus, services are evaluated by the scheduler depending on how fast or slow they are executing the jobs submitted by the broker. In keeping with Principle 1, the broker also does not depend on any metrics supplied by the service - it does its own monitoring.

Extensibility is the key -In Grid environments, transient behavior is not only a feature of the resources but also of the middleware itself. Rapid developments in this still-evolving field have meant that middleware goes through many versions and unfortunately, interface changes are a norm rather than the exception. Also, changing requirements of Grid users require that the broker itself be flexible enough for adding new features or extending old ones. Thus, every possible care has been taken to keep the design modular and clean. The advantages due to this principle:

- Extending broker to support new middleware is a zip – Requires implementation of only three interfaces. (For more details refer to Programming section)
- Getting broker to recognize the new information sources is also easy
- The differences in middleware are invisible to the upper layers such as the scheduler and vice versa. Thus any changes made in one part of the code remain limited to that section and are immediately applicable. For example, after adding a new middleware, the scheduler is immediately able to use any resource using that middleware.
- XPML is extensible. Adding any new constructs is easy, using the same reflection framework (see Programming Section). You could also do away with XPML altogether and implement your own favorites interface to describe applications.

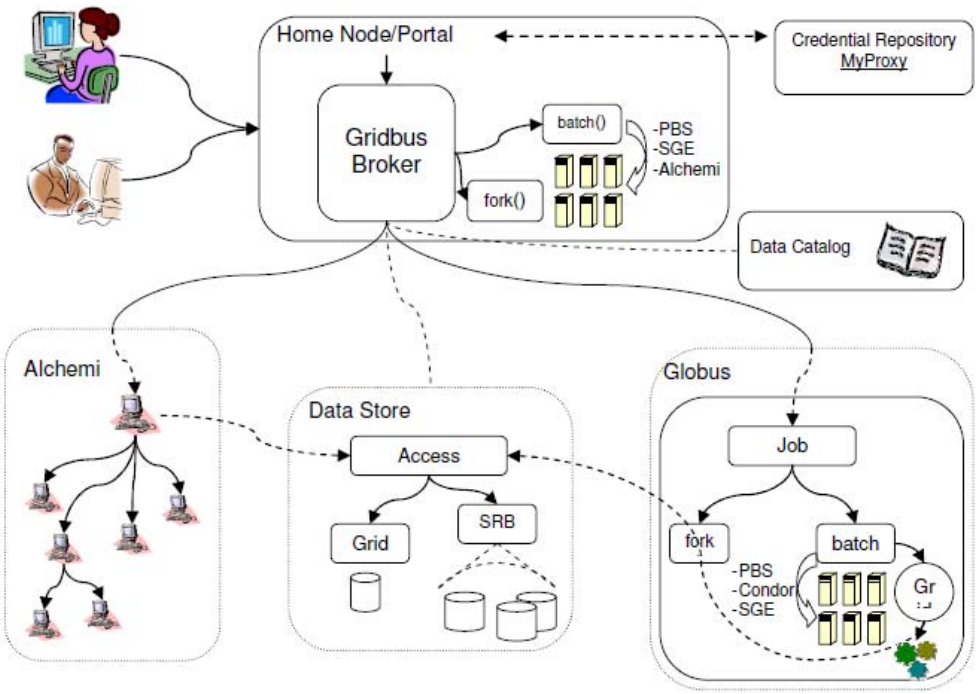

Figure 6

The Gridbus broker uses a default application-description interpreter for a language called XPML (eXtensible Parametric Modeling Language), which is designed to describe dynamic parameter sweep applications on distributed systems in a declarative way. As such the broker can easily execute parameter-sweep applications on the grid. A parameter sweep application is one in which there is a program which operates on multiple sets of data, and each instance of the running program is independent of the other instances. Such applications are inherently parallel, and can be readily adapted to distributed system. In addition, the broker can interpret GGF-defined JSDL documents (only a subset of JSDL is supported). New application descriptions can be easily plugged-in.

The Gridbus broker follows a service-oriented architecture and is designed on object-oriented principles with a focus on the idea of promoting simplicity, modularity, reusability, extensibility and flexibility. The architecture of the broker is shown in Figure 7.

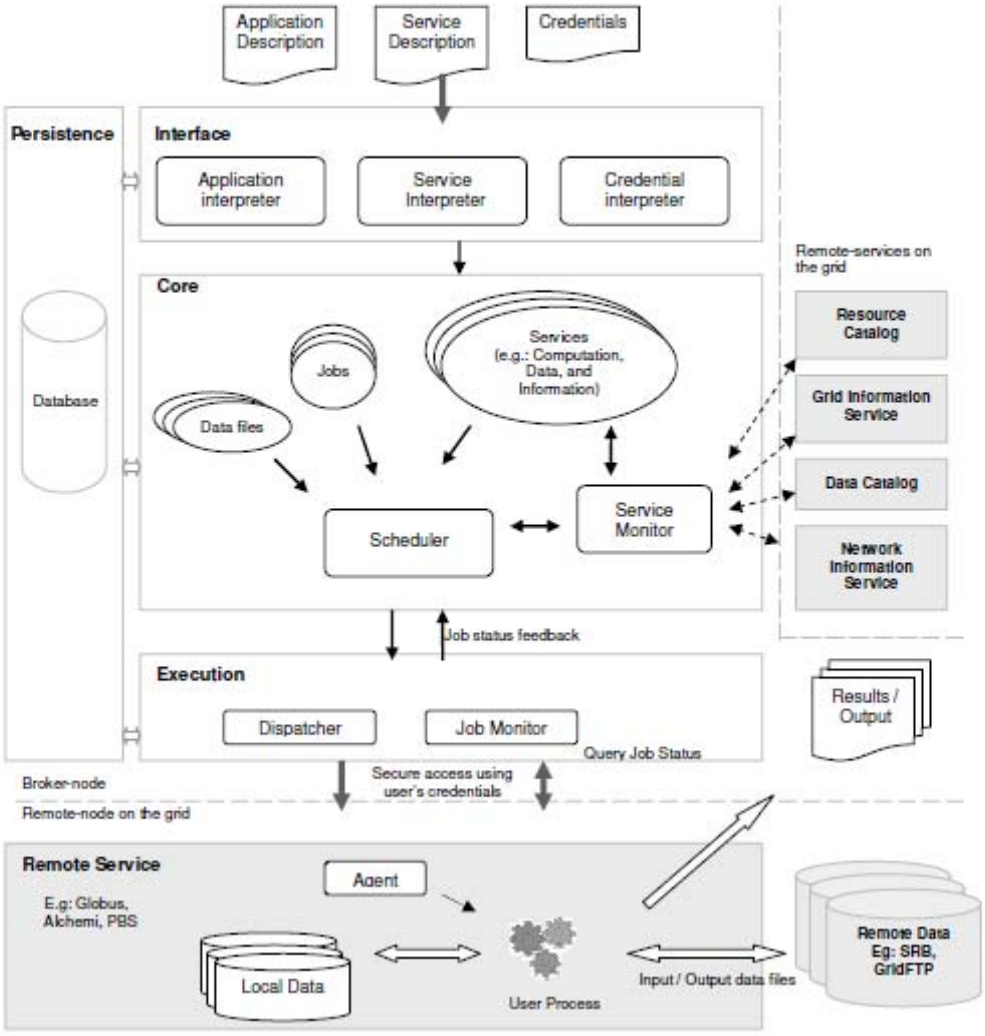

Figure 7

The broker can be thought of as a system composed of three main sub-systems:

- The interface sub-system
- The core-sub-system
- The execution sub-system.

The input to the broker is an application-description, which consists of tasks and the associated parameters with their values, and a resource/service description which could be in the form of a file specifying the hosts available or information service which the broker queries, and a set of credentials to use to authenticate to remote services. The interface sub-system has modules that convert these inputs into entities, called jobs and services with which the broker works internally in the core subsystem.

A job is an abstraction for a unit of work assigned to a node. It consists of a task, and variables. A variable holds the designated parameter value for a job which is obtained from the process of interpreting the application-description. A service represents a service on the grid, which could be a compute, storage, information or application service. The task requirements and the service information drive the discovery of services such as computational nodes, application and data services. The service discovery module connects to the servers to find out if they are available, and if they are suitable for the current application.

The broker uses credentials of the user supplied via the credential description, whenever it needs to authenticate with a remote service. Once the jobs are prepared and the services are discovered, the scheduler is started. The scheduler maps jobs to suitable servers based on its algorithm. The dispatcher submits mapped jobs to the servers using the actuator component, in the execution sub-system, which wraps the job in an agent that is sent to the remote node. The actuator is a middleware specific component which communicates with the remote grid service using protocols understood by it. The job-monitor periodically monitors the jobs using the services of the execution sub-system and updates the broker's internal state, which is persisted to non-volatile storage when needed. As they jobs get completed, the agents take care of clean up and gathering the output of the jobs. The scheduler stops after all the jobs have been completed. The scheduling policy determines whether failed jobs are restarted or ignored.

3.4 The Grid Market Directory (GMD)

The Grid Market Directory (GMD) developed by the Gridbus Project serves as a registry for publication and discovery of Grid service providers and their services. It enables Grid service providers to publish their services and related costs to the public, so that consumers can browse through web browser or query by using SOAP to find a suitable service meeting their requirements. GMD users use the web services and their own API for creating the Grid services. Grid services support discovery, register and usage. Web services use SOAP (Simple Object Access Protocol), WSDL and UDDI. We use SOAP to send and receive standard messages. WSDL tells how to interact with web services and for client implementation. UDDI gives what are all the web services available and how to register the services. To provide all these services in local language by a common man in the world, localization of GMD is developed. This localization would really be very useful for a common man who is not aware of International language. Also the localization done is generic, in the way that it can be localized to any of the native languages required.

The key components of the GMD are GMD Portal Manager (GPM) that facilitates service publication, management and browsing. GPM allows service providers and consumers to use a web browser as a simple graphical client to access the GMD. GMD-Query Web Service (GQWS) enables applications to query the GMD to find a suitable service that meets the job execution requirements. Both the components receive client requests through a HTTP server. Additionally, GMD repository, a database is configured for recording the information of Grid services and service providers.

GMD Portal Manager - The architecture of the GMD Portal Manager (GPM) is shown in Fig. 1. The GPM provides three different access interfaces: service browsing, provider administration and service management.

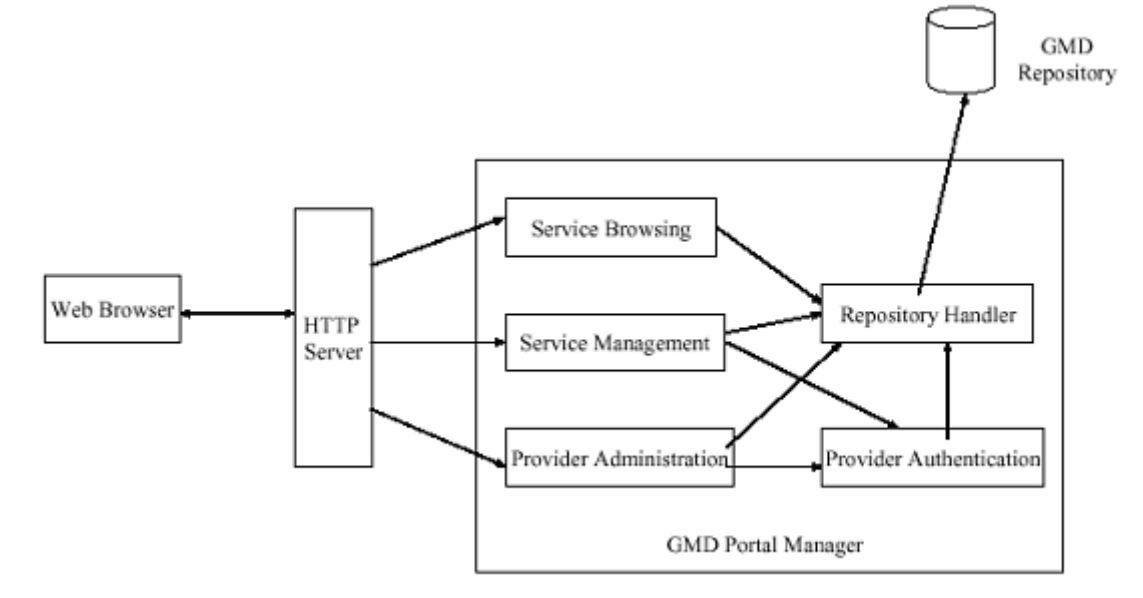

Figure 8

Service browsing - The GPM allows users to browse all registered services or only services offered by a specific provider. Additionally, services in the GMD are categorized by service type, such as Earthquake Engineering, Molecular Docking and CPU Service, so that users can browse them for a particular application area. For instance, the high-energy physics community can browse services related to its area along with their access costs.

Provider administration - The provider administration module is responsible for account management including registration and removal. The account information of the provider is acquired at the time of registration. This includes the provider's name, login name, password, contact address and some additional information.

Service management page is dynamically generated for each registered provider, through which it can add, update and remove services. Basic service attributes include service name, service type, hardware price (cost per CPU-sec), software price (cost per application operation), node host name and location of application deployment (path). In addition, security issues are also addressed in the GPM. A login authentication mechanism for identifying registered providers is employed in the service management and provider administration. In the service management interface, service modification operations are also authenticated before being committed to the repository

The GMD provides web services that applications can use to query the registry. The GQWS is built using SOAP. The main benefit of using SOAP is that it is based on standard XML so that Web services can even be invoked from different applications irrespective of the language used in their implementation.

The GQWS consists of two modules: Query Processor and Repository Handler. The GQWS communicates with its client by messages in XML format. The query message is encapsulated within a SOAP message, which is transferred by HTTP between the web server and the GMD client. The SOAP engine acquires the query message and forwards it to the GQWS. The Query Processor handles query message parsing and takes appropriate actions based on the content of the message, while the Repository Handler is responsible for retrieving data from the database. The response message is finally constructed by the Query Processor and sent back to the client.

The ultimate objective of the project is to enable the access of the grid by a common man not having proper knowledge of the International languages. This support is to be provided for rural area people who are lagging their knowledge of International languages like English. This will surely overcome the usage

of the grid services to the real world at the present scenario. The translation of all Grid market directory web pages to local languages by the translator designed enables them to provide all the web services in their well-known native languages. As they are very familiar with their native languages, the burden for them to get accustomed to these technologies become very easy. All web services translated their own native language support. This project will surely be a boon to all the rural people in a country like India.

The architecture as proposed in Fig. 9 clearly states the issues involved at the client side for the localization of the GMD user Interface. The user input is transferred to a local language Translator which in turn maps the corresponding font required for the conversion of the International language to local language. The localized input is then displayed on the Browser and hence the user interface at the client side input is customized for local language support. The HTTP request sent from the browser is then internally translated to International language by another translator which is not visible at the user level as the grid architecture internally understands only the international language for its operation. After the completion of the grid service by the Grid architecture the HTTP response is sent back to the client, which is once again being translated by the local language translator for providing a local language output interface of the service done at the browser of the client side. Hence both the input and the output user level interfaces for Grid services performed are converted for having local language support at the browser on the client machine.

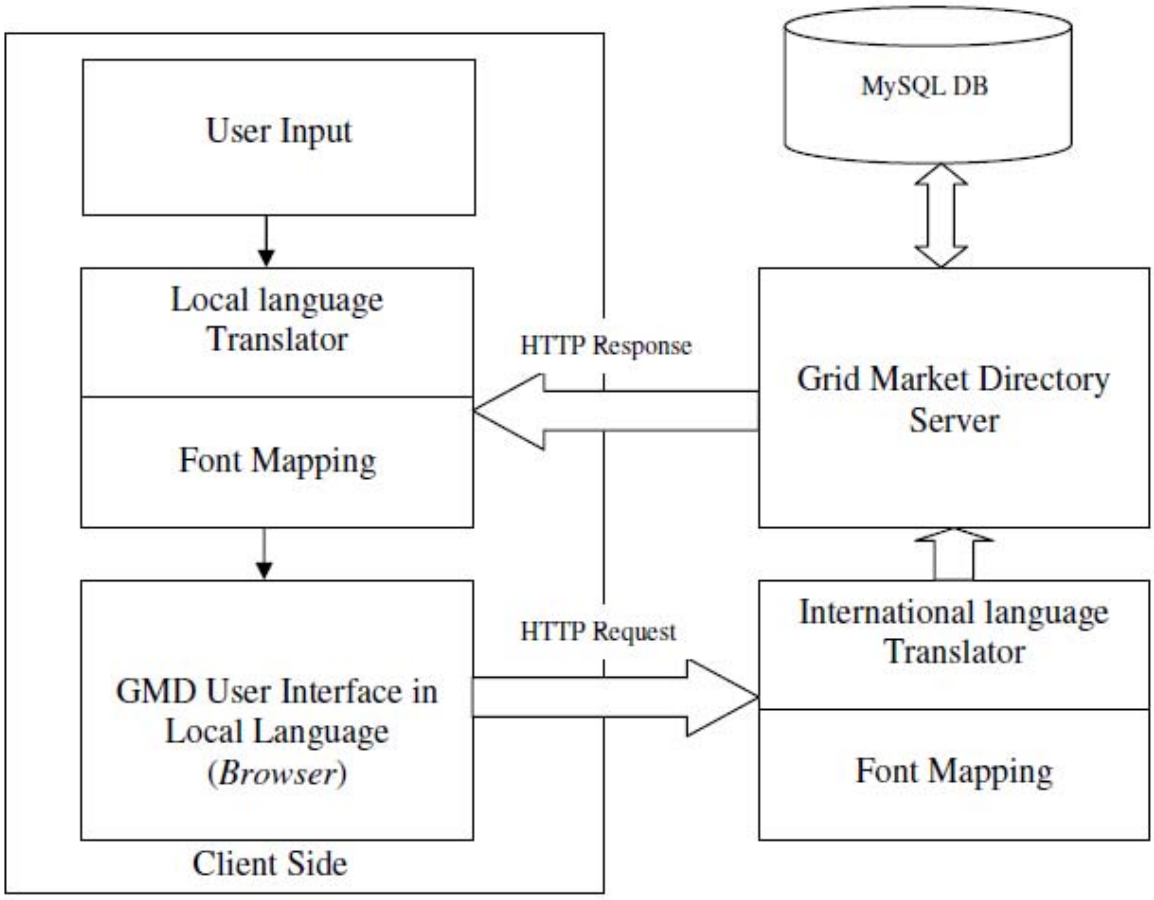

Figure 9

3.5 Grid Bank

GridBank (GB) is a secure Grid-wide accounting and (micro) payment handling system. It maintains the users' (consumers and providers) accounts and resource usage records in a database. GridBank supports protocols that enable its interaction with the resource brokers of Grid Service Consumers (GSCs) and the resource traders of Grid Service Providers (GSPs). It has been primarily designed to provide services for enabling a Grid computing economy; however, we envision its usage in e-commerce applications as well. The GridBank services can be used in both co-operative and competitive distributed computing environments.

3.6 Gridscape II

Gridscape II, presented in this paper, aims to minimize this problem by assisting in the creation of Web based portals as well as easing tasks such as administering these portals, and thus helping users to find the information that is important to them. Web portals have become a popular way of providing a single interface to multiple different systems or fragments of information. The main design premises of a Grid monitoring portal such as Gridscape II are that it should:

- Manage diverse forms of resource information from various types of information sources;
- Allow new information sources to be easily introduced;
- Allow for simple portal management and administration;
- Provide a clear and intuitive presentation of resource information in an interactive and dynamic portal;
- Have a flexible design and implementation such that core components can be reused in building new components, presentation of information can be easily changed and a high level of portability and accessibility (from the web browser perspective) can be provided.

The Gridscape II architecture consists of four main components:

1. Gridscape II Core
2. Gridscape II Resource Monitor
3. Gridscape II Portal
4. Interactive Client-Side Map (Google Maps Interface

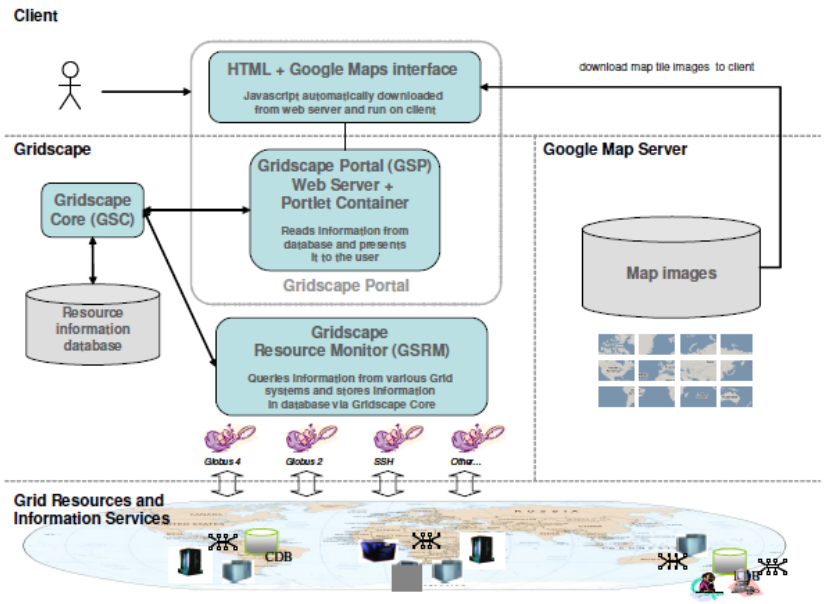

Figure 10

The architecture of Gridscape II and its interaction with various Grid resources are illustrated in Figure 10. The figure identifies the three components which form Gridscape II: the Gridscape II Portal which provides the user interface, the Gridscape II Resource Monitor which gathers information about Grid resources, and the Gridscape II Core which is the data model for the system. The other important component is the Google Maps JavaScript interface, which is a part of the portal.

The Gridscape II Core (GSC) is the data model and represents the set of core components which are common to the rest of the system. This includes entities representing resources, location and resource information. The core also provides data access abstraction for accessing and managing Grid resource information. The overall structure of Gridscape II was aimed towards being integrated with a persistent backend so that the updating of resource information and the presentation of that information could be performed by entirely independent entities or separate applications, and so that state information is not kept in the memory of the presentation layer.

The Gridscape II Resource Monitor (GSRM) has the flexibility to be able to query any information source, and in a Grid context this includes Grid middleware such as Globus 2 and Globus 4, Alchemi, Unicore, XGrid, SGE and PBS. Therefore, the GSRM is required to support a number of different protocols in order to gather information from these diverse information sources. The role of the GSRM is to gather information from these various sources and to publish that information for use by other components. The collective set of information becomes the current state of the Grid as seen by Gridscape II, and becomes available as a simple set of information to be utilized by a lightweight presentation layer. The importance of having this independent entity gathering resource information is that it alleviates the burden from the user interface and makes it easy to extend or create new user interfaces that are only required to present information. It was decided to avoid creating an information reporting agent that is installed on each resource, as this makes initial setup and adding new resources more difficult and often site administrators may not want to install such agents. As a result, Gridscape II can be used in a wide variety of applications and integrate with various information sources. Custom information gatherers can be created and plugged into the GSRM to enable the gathering of information from any type of information provider.

The Gridscape II Portal (GSP) provides the user interface for the Gridscape II system. Using portlet technology, Grid resource information is presented to the user via their web browser. The portlet implementation also means that the interface can be easily plugged into other portals. To improve the interactivity of the otherwise static HTML interface, the Google Maps API is used to provide a highly interactive view of the globally distributed Grid resources and their positioning around the world. It is also important that the information is made available in a Web environment for increased accessibility and availability, as well as providing an intuitive user interface to enable easy navigation.

Google Maps provides an interactive world map on top of which other information, such as the location and properties of Grid resources around the world, is overlaid. The lightweight Javascript-based Google Maps client is downloaded onto the user's machine and runs within the web browser. From the web browser, asynchronous HTTP requests are made to retrieve map tile images from the map server that need to be presented as part of the portal. The asynchronous requests allow the user to zoom and pan around the map, while updating the display seamlessly without requiring an entire page refresh.

3.7 Alchemi

Software to enable grid computing has been primarily written for Unix-class operating systems, thus severely limiting the ability to effectively utilize the computing resources of the vast majority of desktop computers i.e. those running variants of the Microsoft Windows operating system. Addressing Windows based grid computing is particularly important from the software industry's viewpoint where interest in grids is emerging rapidly. Microsoft's .NET Framework has become near-ubiquitous for implementing commercial distributed systems for Windows-based platforms, positioning it as the ideal platform for grid computing in this context.

Alchemi is a .NET-based grid computing framework that provides the runtime machinery and programming environment required to construct desktop grids and develop grid applications. It allows

flexible application composition by supporting an object-oriented grid application programming model in addition to a grid job model. Cross-platform support is provided via a web services interface and a flexible execution model that supports dedicated and non-dedicated (voluntary) execution by grid nodes.

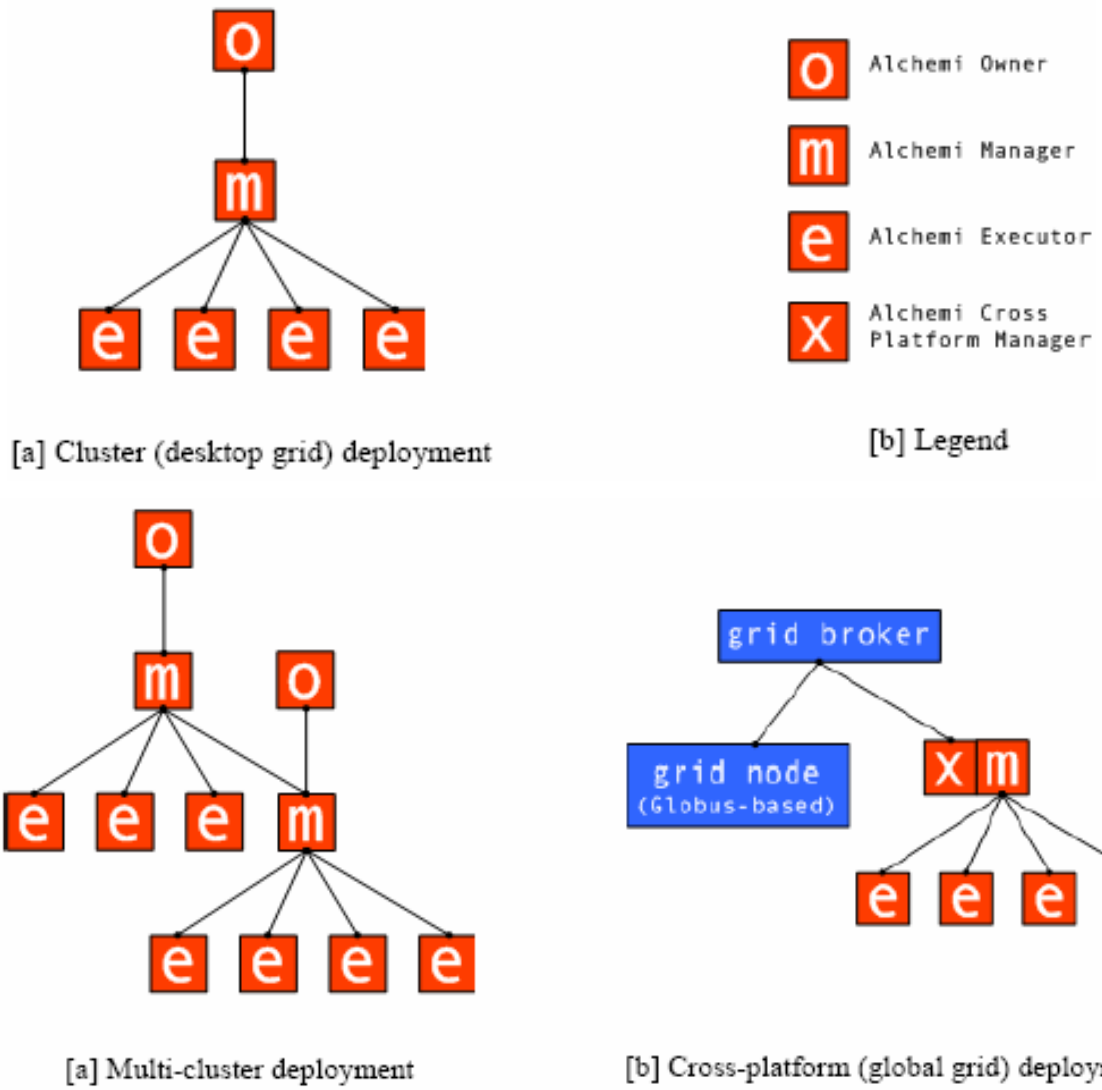

Figure 11

3.8 GridSim

The GridSim toolkit allows modeling and simulation of entities in parallel and distributed computing (PDC) systems-users, applications, resources, and resource brokers (schedulers) for design and evaluation of scheduling algorithms. It provides a comprehensive facility for creating different classes of heterogeneous resources that can be aggregated using resource brokers for solving compute and data intensive applications. A resource can be a single processor or multi-processor with shared or distributed memory and managed by time or space shared schedulers. The processing nodes within a resource can be heterogeneous in terms of processing capability, configuration, and availability. The resource brokers use scheduling algorithms or policies for mapping jobs to resources to optimize system or user objectives depending on their goals.

Overview of GridSim functionalities:

- Incorporates failures of Grid resources during runtime.
- New allocation policy can be made and integrated into the GridSim Toolkit, by extending from AllocPolicy class.
- Has the infrastructure or framework to support advance reservation of a grid system.
- Incorporates a functionality that reads workload traces taken from supercomputers for simulating a realistic grid environment.
- Incorporates an auction model into GridSim.
- Incorporates a data grid extension into GridSim.
- Incorporates a network extension into GridSim. Now, resources and other entities can be linked in a network topology.
- Incorporates a background network traffic functionality based on a probabilistic distribution. This is useful for simulating over a public network where the network is congested.
- Incorporates multiple regional GridInformationService (GIS) entities connected in a network topology. Hence, you can simulate an experiment with multiple Virtual Organizations (VOs).
- Adds ant build file to compile GridSim source files.

3.9 Gridbus Deployment in the Global Data-Intensive Grid Collaboration

The Gridbus Project has led the establishment a world-wide collaboration, called the Global Data-Intensive Grid Collaboration, with of aim of creating a virtual organization to demonstrate a large number of distributed data-intensive computing applications by harnessing geographically distributed resources. The collaboration was nominated as one of finalists for HPC Challenge event organized as a part of the IEEE/ACM Supercomputing Conference (SC 2003) held at Phoenix, Arizona, USA from Nov. 15-21, 2003. The collaboration has put together a World-Wide Grid (WWG) that contains over 200 Grid nodes (PCs, workstations, clusters, supercomputers, databases, and applications) contributed by organizations and volunteers based in Australia, Asia, Europe, North America, and South America. A snapshot of status of WWG resources as visualized using Gridscape is shown in Figure 12. The tested supported included resources running Unix-variant or Windows operating systems.

Some of the vital statistics of the tested are as follows:

- Grid Nodes: 218 distributed across 62 organizations from 21 countries around the world.
- Laptops, desktop PCs, WS, SMPs, Clusters, supercomputers
- CPU Architecture:
- Intel x86, IA64, AMD, PowerPC, Alpha, MIPS
- Operating Systems:
- Windows or Unix-variants – Linux, Solaris, AIX, OSF, Irix, HP-UX
- Intranode Network:
- Ethernet, Fast Ethernet, Gigabit, Myrinet, QsNet, PARAMNet
- Internet/Wide Area Networks
- GrangeNet, AARNet, ERNet, APAN, TransPAC, and so on.
- Grid Middleware:
- Alchemi for access to Windows nodes and Globus for Unix-variants.
- The Gridbus Service Broker for both Windows and Unix-variants resources.
- Nimrod-G for accessing Unix-variants resources for running GAMESS application.
- Other Gridbus technologies such as Gridscape, G-Monitors as described above.

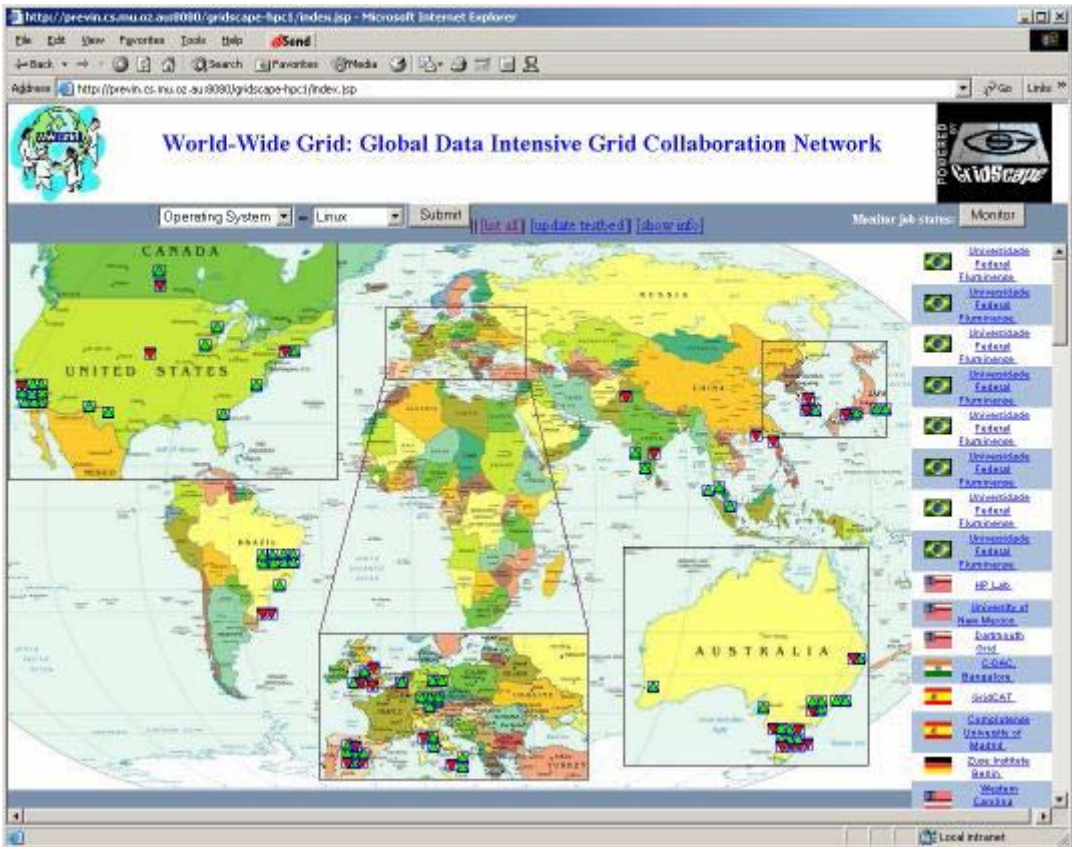

Figure 12

The contributors with resources running Unix-variant OSes have provided Grid access through Globus whereas those running Windows and .NET have provided Grid access through Alchemi middleware. The collaboration demonstrated on demand deployment of various data-intensive computing applications from natural language processing and particle physics to portfolio analysis on the WWG using the Grid Service Broker (GSB). The broker was able to simultaneously utilize both Alchemi and Globus-based resources and deploy appropriate application codes at runtime. Nimrod-G broker developed by Monash University was used in demonstrating a quantum chemistry application (GAMESS).

4. Bibliography

http://www.cloudbus.org/gridsim/

http://www.cloudbus.org/reports/GRIDS-Lab-AnnualReport2007.pdf

http://ww2.cs.mu.oz.au/~raj/grids/papers/vpm.pdf

http://www.cloudbus.org/broker/3.0/manual.v3.0.pdf

http://www.cloudbus.org/reports/GRIDS-Lab-AnnualReport2007.pdf

http://arxiv.org/ftp/cs/papers/0404/0404027.pdf

http://ww2.cs.mu.oz.au/~raj/grids/papers/vpm.pdf

http://arxiv.org/ftp/cs/papers/0302/0302007.pdf

http://www.cloudbus.org/reports/gmd-tamil.pdf

http://arxiv.org/ftp/cs/papers/0302/0302006.pdf