DESIGN AND DEVELOPMENT OF GRID BASED RESOURCE BROKER ALGORITH USING GLOBUS

A Dissertation

Submitted in partial fulfillment

of the requirement for the award of the Degree of

MASTER OF ENGINEERING
in
COMPUTER TECHNOLOGY & APPLICATIONS

By

JAI PRAKASH
College Roll No. (13/CTA/03)
Delhi University Roll No. 3013

Under the guidance of

Dr. Goldie Gabrani

Department of Computer Engineering
Delhi College of Engineering,
University of Delhi

2003-2005

CERTIFICATE

This is to certify that the work that is being presented in this dissertation entitled ***"Design and Development of Grid based Resource Broker Algorithm using Globus"*** submitted by **Jai Prakash** in the partial fulfillment of the requirement for the award of degree of **Master of Engineering** in Computer Technology and Application, Delhi College of Engineering is an account of his work carried out under my guidance and supervision.

The work embodies in this dissertation has not been submitted for the award of any other degree to the best of my knowledge.

Professor D. Roy Choudhury
Head of Department
Department of Computer Engineering
Delhi College of Engineering
Delhi

Dr. Goldie Gabrani
Asst. Professor
Department of Computer Engineering
Delhi College of Engineering
Delhi

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extent my heartiest felt gratitude to everybody who helped me throughout the course of this project.

It is distinct pleasure to express my deep sense of gratitude and indebtedness to my learned supervisor **Dr. Goldie Gabrani**, Assistant Professor in the Department of Computer Engineering, Delhi College of Engineering, for her invaluable guidance, encouragement and patient review. His continuous inspiration only has made me complete this major project.

I would also like to take this opportunity to present my sincere regards to my teachers viz. Professor D. Roy Choudhury, Dr. Goldie Gabrani, Dr S. K. Saxena, Mr. Rajeev Kumar and Mrs. Rajni Jindal for their support and encouragement.

I would like to express my heartiest felt regards to Professor Asok De, Head of Computer Center for providing Linux lab for my work..

I am thankful to my friends and classmates for their unconditional support and motivation during this project.

Jai Prakash
M.E. (Computer Technology & Applications)
College Roll No. 13/CTA/03
Delhi University Roll No. 3013

ABSTRACT

A grid is a high performance computational environment that is composed of heterogeneous resources spanning wide area networks and multiple administrative domains. This is a new computing paradigm that aims to make high performance computational resources available to anyone accessing the grid. Problems inherent in such an environment include the ability to discover suitable resources and how to schedule separate tasks of an application on those resources in such a way as to maximize the applications performance. A component of the grid infrastructure of which an instance is associated with each application is called the resource broker. While there has been a lot of development into the mechanisms vital to grid computing there is a growing need to develop methods for resource brokering, an area of grid computing which has been generally overlooked.

The aim of this thesis is to setup of grid and to design and development of a resource broker using Globus 2.4 toolkit and its C API. Resource brokers task is to implement the execution of the application on the grid transparently from the users view. This project was an investigation of brokering algorithms in a grid environment. Brokering is the act of discovering, scheduling and monitoring resources to perform the tasks that constitute an application running in the grid environment.

CONTENTS

Certificate …… i

Acknowledgement …… ii

Abstract…… iii

Contents…… 1

1. Introduction…… 4

1.1 What is Grid…… 4

1.1.1 Types of Grid ……... 7

1.1.2 Grid features and its application……... 8

1.2 Grid System Architecture …… 10

1.3 Statement of Problem ……... 13

1.4 Organization of Dissertation……. 13

2. Grid Panorama using Globus……... 14

2.1 Globus Basic Architecture…… 16

2.2 Security Service……. 17

2.3 Resource Management ……. 20

2.4 Information Service…….. 21

2.5 Data Management……. 23

3. Resource Broker: Middleware between client and grid resources… 25

3.1 Globus Resource Management Architecture …… 26

3.2 Resource broker tasks ……..... …….. 29

3.2.1 Finding the resource……. 29

3.2.2 Resource selection……... 30

3.2.3 Job scheduling…….. …….’ 30

3.2.4 Job monitoring and migration…… . ……. 31

3.3 Motivation behind the work……. 32
3.4 Application Enablement considerations ……..…….. …. . 32
3.5 Proposed Algorithm…… . 33
3.6 Brokering Techniques……. 34
3.7 Dynamic Brokering…….. 35

4. Setup of Grid Environment ……. 37
4.1 System Requirement ……. 37
4.2 Installation Consideration…….. 38
4.2.1 Choosing a Host…… 38
4.2.2 Choosing File System……..…… 39
4.2.3 Contributing Resources……. 39
4.2.4 Information Services……. 40
4.2.5 Security…… 41
4.2.5.1 X.509 Certificate Process…….. 41
4.2.5.2 Environment Variables……. 42
4.3 Required Software……. 43
4.4 Lab Environment …….. 44
4.4.1 Naming and Addressing …….. 45
4.5 Setting up Linux requirement ……... 46
4.5.1 Linux Setup……..…… 46
4.5.2 Configure Network Time Protocol (NTP)…… 47
4.6 Installation and Configuration Globus……... 47
4.6.1 Setup of own Certificate Authority……... 47
4.6.2 Host Certificate…….…… 49
4.6.3 User Certificate …… 49
4.6.4 Grid mapfile entry…… 50
4.6.5 Testing…….. 50
4.7 Setting up Gatekeeper…… 51
4.8 Setting up MDS……. 52
4.8.1 MDS on client…….. 53

4.8.2 Secure MDS…….. 53
4.8.2.1 Request a LDAP Certificate……. 53
4.8.2.2 Signing a LDAP Certificate……. 53
4.9 Verification…… 54
4.9.1 Server Interface……. 54
4.9.2 Client Interface……. 54
4.10 Setting up Grid Service…… .. 56
4.10.1 Deploying Application…… 57
4.10.2 Making Application Data Available…… .. 58

5. Design and Development ……. 60
5.1 Integrating with the LDAP protocol…….. 60
5.2 Design issues……. 61
5.2.1 Globus libc APIs……. 61
5.2.2 Makefile…… 62
5.2.3 Globus module…… 63
5.2.4 Callbacks…… 64
5.2.5 GRAM server……. 66
5.3 Development …… 66
5.3.1 Implementation……. 73
5.3.1.1 Querying the MDS and establishing the connection……. 73
5.3.1.2 LDAP query……... 75
5.3.1.3 Submitting the LDAP query…….. 75
5.3.1.4 Retrieving results from the Globus MDS…….. 75

6. Conclusion and Future work …… 77

References ……..... 79

Appendix ……..…… 85

Chapter 1

INTRODUCTION

Distributed computing [1] is a subject that has been studied a lot in the past, but recently a new field, Grid computing, has drawn attention back to the area. Grid computing differs from traditional distributed computing in its goals: sharing resources on heterogeneous platforms on a geographically wide area is now coming possible through advances in network technology.

A grid [2, 46] enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems data sources and specialized devices owned by different organizations administrated with different policies. Grids are typically used for solving large-scale resource and computing intensive problems in science, engineering, and commerce.

Grid computing [3, 45] is a sub set of distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of the connected heterogeneous systems sharing various combinations of resources. The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing. Grid computing involves coordination and networking of resources across dynamic and geographically dispersed organizations in a transparent way for users. Grid technologies emphasize effective operation in large scale, wide area environments, including access to remote computation, information services, high speed data transfers and gateways to local authentication schemes.

1.1 What is Grid?

Distributed computing has fascinated researchers all over the world for past two decades. Traditionally the focus has been on developing a unified homogeneous distributed system.

Recently, there has been tremendous growth in wide area distributed computing leading to grid computing.

The term Grid is chosen as an analogy to electric power grid that provides consistent, pervasive, dependable, transparent access to electricity, irrespective of type and location of source. The primary focus in Grid computing is to harness the power of geographically distant supercomputers, and convert them into a big computing resource.

The Grid enables sharing, selection and aggregation of various resources including raw CPU cycles, storage systems, data sources and special services like application servers, etc. These resources may be geographically dispersed, operated by different organizations with different policies running on completely different operating systems. Figure 1.1 is the simplest grid consists of just a few machines, all of the same hardware architecture and same operating system, connected to local network.

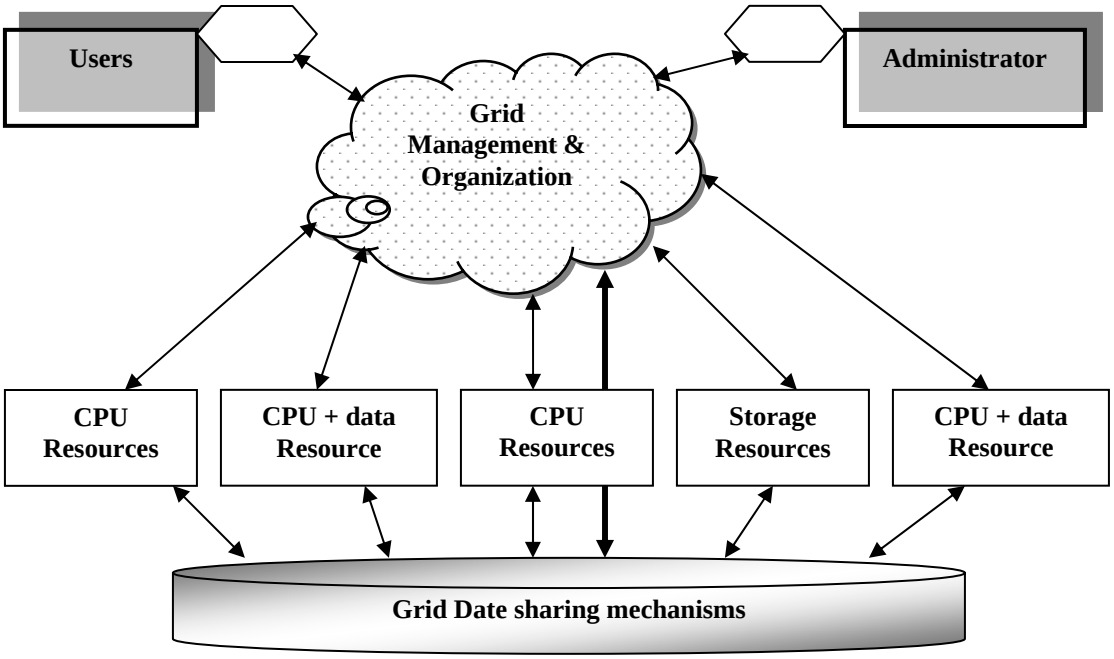

Figure 1.1: The Simple Grid

This kind of grid uses homogeneous systems so there are fewer considerations and may be used just for experimenting with grid software. The machines are usually in one department of an organization, and their use as a grid may not require any special policies or security concerns. Because the machines have the same architecture and operating system, choosing application software for these machines is usually simple. Some people would call this a "cluster implementation rather than a "grid."

Due to the recent explosion of Grid Technologies, there has been some confusion over the exact nature of a Grid. Dr. Foster provides a three point checklist [4] for evaluating grid systems.

A grid is a system that

- Coordinates resources that are not subject to centralized control - A Grid integrates and coordinates resources and users that live within different control domains for example, the user's desktop vs. central computing; different administrative unit of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings. Otherwise, we are dealing with a local management system.

- Uses standard, open, general-purpose protocols and interfaces - A Grid is built from multi-purpose protocols and interfaces that address such fundamental issue as authentication, authorization, resource discovery, and resource access. It is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application-specific system.

- Delivers nontrivial qualities of service - A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co- allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

Grids are composed of VO (Virtual Organizations), a conglomeration of network resources consisting of servers, desktop PCs, mainframes, clusters, etc. These VOs are managed by different organizations and may have different policies and security mechanisms. Grid enables sharing of resources between these heterogeneous VOs with a common set of open protocols. There are enormous opportunities for application writers to exploit the large amount of computational and storage resource provided by the grid. There are enormous opportunities for application writers to exploit the large amount of computational and storage resource provided by the grid.

1.1.1 Types of Grid

A grid should provide full-scale integration of heterogeneous computing resources of any type: processing units, storage units, communication units, and so on. However, as the technology hasn't yet reached its maturity, real-world grid implementations are more specialized and generally focus on the integration of certain types of resources. Different types of grids describe as follows [5]:

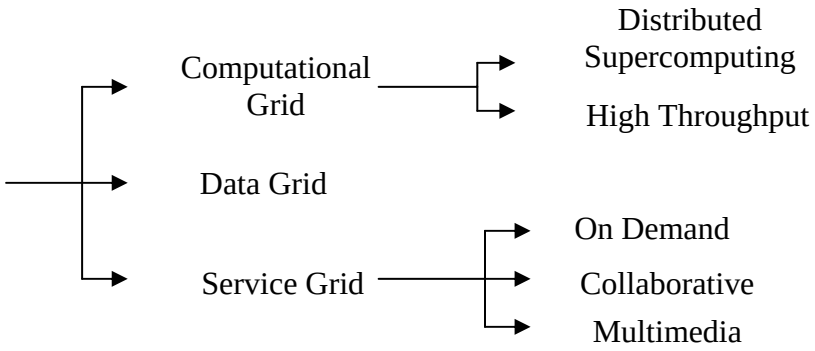

Figure 1.2: Grid Systems Taxonomy

1. Computational grid: A computational grid is a grid that has the processing power as the main computing resource shared among its nodes. This is the most common type of grid and it has been used to perform high-performance computing to tackle processing-demanding tasks.

2. Data grid: A data grid has the data storage capacity as its main shared resource. Such a grid can be regarded as a massive data storage system built up from portions of a large number of storage devices.

3. Service grid This is known as either a service grid or a delivery grid. Such a grid has as its main purpose to provide fault-tolerant and high-performance communication services. In this sense, each grid node works as a data router between two communication points, providing data-caching and other facilities to speed up the communications between such points. Examples of such grids are On Demand computing, Multimedia computing or Collaborative computing.

1.1.2 Grid Features and its application

A computational grid provides high-performance computing; a data rid provides large storage capacity; and a network grid provides high throughput communication that may be useful for a variety of applications, such as virtual conferences. So main reasons for using grid computing as follows:

- Improve efficiency/reduce costs
- Exploit under-utilized resources
- Enable collaborations
- Virtual resources and virtual organizations (VO)
- Increase capacity and productivity
- Parallel processing capacity
- Support heterogeneous systems
- Provide reliability/availability
- Access to additional resources
- Resource balancing
- Reduce time to results

Applications

Grid implementations only make sense in environments where a meaningful number of computing resources can be integrated to form a higher-performance system, which tends to be rather restrictive. Grid technology in different industries, but it is not restricted to only those areas. Grid computing capability is growing everyday, and we believe that someday, all

systems and applications will run in some kind of Grid like environment. The following list is far from complete, but represents potential for the present and future:

Finance

- Derivative analysis
- Statistics
- Portfolio risk
- Insurance policy cost
- Real-time stock market analysis

Life sciences

- Drug screening
- Protein folding
- Protein sequencing

Medicine

- Record management
- Automated analysis and diagnosis
- Research into the nature of disease

Government/Academia

- Dispersed research center collaboration
- Weather forecasting
- General high-performance computing

Energy

- Seismic analysis
- Oil field simulation

Manufacturing

- Product design
- Simulation
- Modeling
- Finite element analysis (anything involving flow (air, water, fuel, and so on)

Telecommunications/Media

- Video rendering
- Network gaming
- Content distribution
- Dynamic bandwidth for new classes of applications

Electronics

- Chip layout optimization
- Board layout optimization
- Circuit simulation

1.2 Grid System Architecture

In this section we present Grid architecture as it is described in the Globus project [6]. Figure 1.2, Grid Architecture represents a conceptualization of the main principles and requirements in Grid environments. The motivation for building this architecture is the need for a new model describing sharing of heterogeneous resources. This architecture identifies the basic components of Grid systems, defines the purpose of such components, and finally indicates how these components interact with each other.

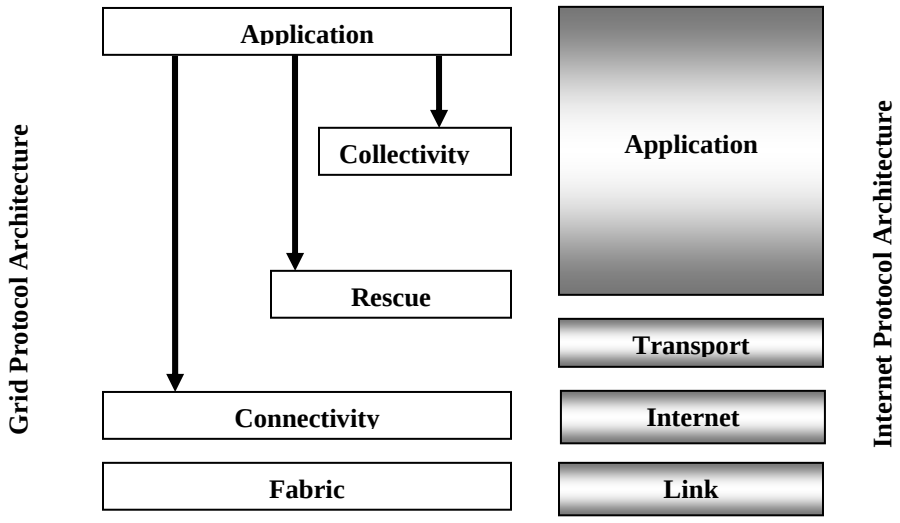

Figure 1.3: High-level Grid Architecture and Functional Blocks

The architecture of the Grid is described in terms of layers, each providing a specific function. In general, higher layers are focused on the user (user-centric), whereas lower layers are more focused on computers and networks (hardware-centric). The different layers and functionalities are described as follows:

- **Fabric layer:** Interfaces to local control, of physical and logical resources. The fabric layer is composed by computational resources, storage systems, catalogs, distributed file systems, network resources, and sensors to be share.

- **Connectivity layer:** Defines core communication and authentication protocols supporting Grid-specific network transactions.

The authentication protocols are governed by the following principles:

Single sign on applies to enabling the user to have multiple access to the resources from the Fabric layer during the same login, once the authenticity has been established. That is, once sign on is performed, the user is authenticated for the entire Grid.

Delegation designates the ability to provide a program with the appropriate rights such that it could behave on user's behalf and further access those resources to which the user has permissions.

Integration with various local security solutions addresses the issue of allowing communication with the local security solutions by providing mapping to the local environment. For instance, Grid security should be able to cooperate with Kerberos and Unix security which could be implemented by the providers of sites or resources.

User-based trust relationships are concerned with directing the security constrains from the user to the intended resources and not further among their providers.

- **Resource layer**: Allows the sharing of a single resource. This layer includes protocols for control and management of individual resources.

Two primary classes of resource layer protocols can be distinguished:

Information protocols are used to obtain information about the structure and state of a resource, for example, its configuration, current load, and usage policy.

Management protocols are used to negotiate access to a shared resource, specifying, for example, resource requirements (including advanced reservation and quality of service) and the operation(s) to be performed, such as process creation or data access. Management protocols are responsible for instantiating sharing relationships, ensuring that the requested protocol operations are consistent with the policy under which the resource is to be shared. Issues that must be considered include accounting and payment. A protocol may also support monitoring the status of an operation and controlling the operation.

- **Collective layer**: Allows resources to be viewed as collections. This layer includes all the services that allow us to manage several resources.

Examples of services are:

Directory services enabling the discovery of resources. A directory service supports queries for resources by name or by attributes such as type, availability, or load. Monitoring and diagnostics services enabling fault detection, such as overload, failure, intrusion.
Grid-enabled programming systems thus extending their functionality by augmentation. Software discovery services envisaging the discovery and selection of the best software and platform for solving a selected problem.

- **Application layer:** Uses the appropriate components of each layer to support the application. Applications Grid access to the infrastructure. According to the requirements of the application, it can be necessary to happen through all the layers or to connect themselves directly to the infrastructure.

1.3 Statement of the Problem

The aim set for the work, which is being presented in this dissertation, can be stated as follows:

1. To establish the grid computing environment for the different services of grid.
2. To design and development of resource broker using the Globus toolkit‘s Monitoring and Discovery system (MDS) and Globus toolkit C API.

1.4 Organization of the Dissertation

In chapter two, Globus environment and its features are explained .In this chapter various components of Globus toolkit and their utilities have been explained for creating globus environment .It also deals with security system used in grid environment . Chapter three describes functions of resource broker , a proposed algorithm to develop resource broker and brokering techniques used by some schedulers .Chapter four describes the steps for setting up grid environment in the lab . It also explains various software components needed to setup grid system. It entirely explains client-server architecture in grid system. Chapter five deals with design and development of resource broker using Globus toolkit 2.4 C API programming interface. A detailed description of routines used in the implementation of broker is also provided here. Sixth chapter concludes the dissertation with some suggestions for future work.

Chapter 2

Grid Panorama using Globus

A middleware is software that organizes and integrates the disparate computational facilities belonging to a Grid. Its main role is to automate all the machine to machine (M2M) negotiations required to interlace the computing and storage resources and the network into a single.

All major Grid projects [47] are being built on protocols and services provided by the Globus Toolkit, a software "work-in-progress" which is being developed by the Globus Alliance. It provides a set of software tools to implement the basic services and capabilities required to construct a computational Grid, such as security, resource location, resource management, and communications. This is an open-source initiative to produce a standard Grid architecture for distributed resources. Initially, Globus was intended to provide a secure means to submit jobs to a third-party scheduling and clustering system. It is based on the premise that grid computing can be seen as three pyramids built on top of a security infrastructure [7]:

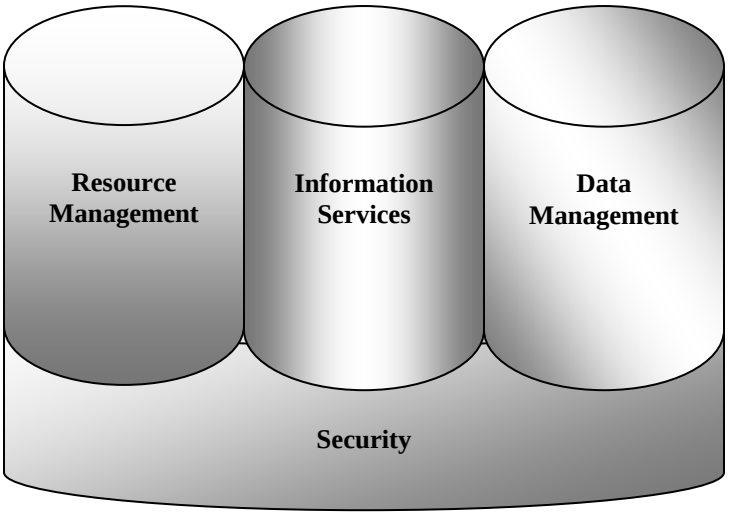

Figure 2.1: The Three pyramids

Examples of applications work being done by groups around the world include:

- **Smart instruments**: Advanced scientific instruments, such as, electron microscopes, particle accelerators, and wind tunnels, coupled with remote supercomputers, users, and databases, to enable interactive rather than batch use, online comparisons with previous runs, and collaborative data analysis.

- **Computationally enhanced desktops**: Software packages, such as, chemical modeling and symbolic algebra that transfer computationally intensive operations to more capable remote resources.

- **Collaborative engineering**: High-bandwidth access to shared virtual spaces that support interactive manipulation of shared data sets and steering of sophisticated simulations for collaborative design of complex systems.

- **Distributed computing**: Virtual supercomputers constructed from many individual supercomputers to solve problems too large for any single computer to accommodate.

- **Parameter studies**: Rapid, large-scale parametric studies, in which a single program is run many times in order to explore a multidimensional parameter space.

There are other software packages that provide similar services, but the Globus Toolkit differs from these in three significant ways:

- Its bag of services approach, which allows application software to use components of the Globus Toolkit without having to adopt the whole Globus Toolkit or a particular programming model or language.

- Its provision of specialized mechanisms that usually coexist with but also sometimes replace mechanisms provided by commodity computing.

- Its support for an information-based approach to meeting application performance requirements.

Using the basic services provided by the Globus Toolkit, researchers may build a range of higher-level capabilities. For example, the Globus Toolkit provides a complete implementation of the Message Passing Interface (MPI) that can run across heterogeneous collections of computers.

Due to its bag of services approach the Globus Toolkit can be used in different ways. The Globus Toolkit can be used at a site that wishes to participate in a computational grid to contribute resources to a grid's pool of resources. (The use of a site's grid resources is closely controlled by the site's access policies.) The Globus Toolkit can also be used to provide access to other grid resources without contributing any of a site's own resources that is, assuming that the appropriate access policies have been negotiated with the owners of the other resources. The Globus Toolkit also provides other services, like single sign-on authentication, without the need for contributing computational resources.

2.1 Globus basic service architecture

Globus Toolkit 2.4 [8] provides some basic services that should be found in a grid. Each service needs a standard protocol or component because a grid deals with diverse types of resources.

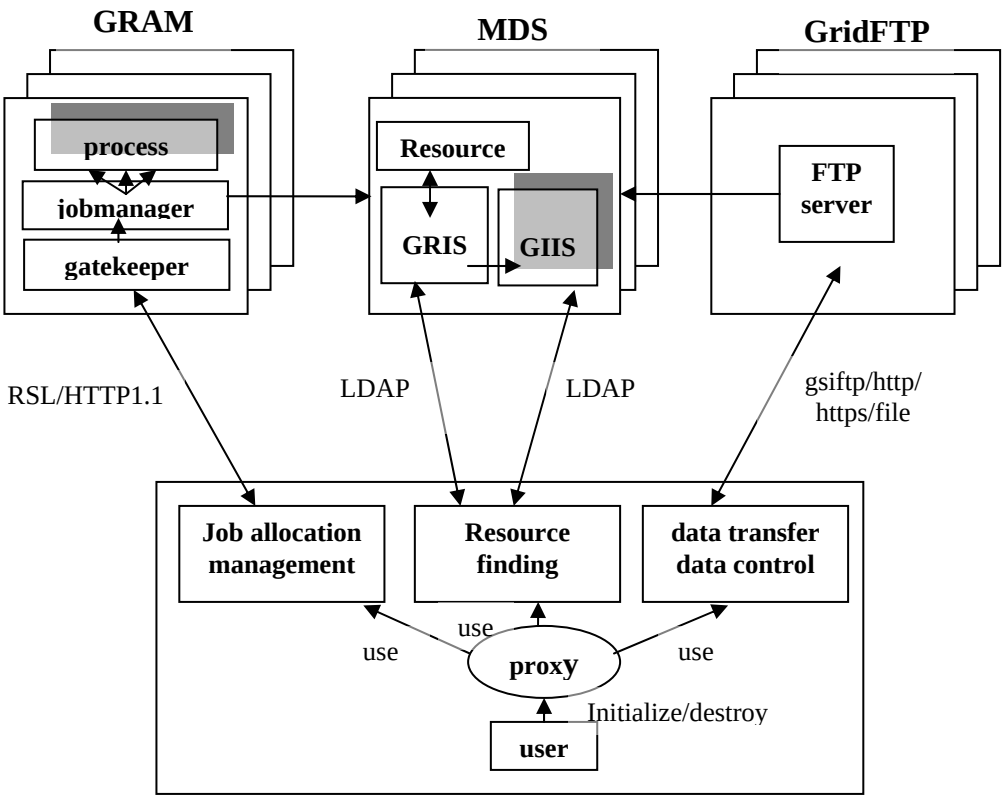

Figure 2.2: The system overview of Globus Toolkit

2.2 Security Service

Grid Computing, being distributed and heterogeneous in its nature, has high demands on security. Security service is implemented by the Globus Security Infrastructure (GSI) [9, 10]. To implement that, GSI utilizes Secure Socket Layer (SSL) [11] protocol, public key encryption [12], and X.509 [13] certificates.

The primary motivations behind the GSI are:

- The need for secure communication between elements of a computational Grid.
- The need to support security across organizational boundaries, thus avoiding a centrally managed security system.
- The need to support "single sign-on" for users of the Grid, including computations that involve multiple resources and/or sites.

GSI gives access to the grid using a Certificate Authority and a set of keys for public key cryptography. The routine to establish the GSI communication starts with copying the CA's public key to the GSI client which generates a private key and a certificate request. The certificate is sent to the CA which signs it and sends it back to the GSI client. Then secure communication is established as the client possesses a private key, the public key of the CA and his (her) digital certificate.

Once a new grid host has successfully gained access to the grid, it can start communicating with other hosts. When setting up a communication between two hosts, the first task is to be able to determine which host is certified and which is not. The GSI authentication process is summarized in the following illustration (Figure 2.3).

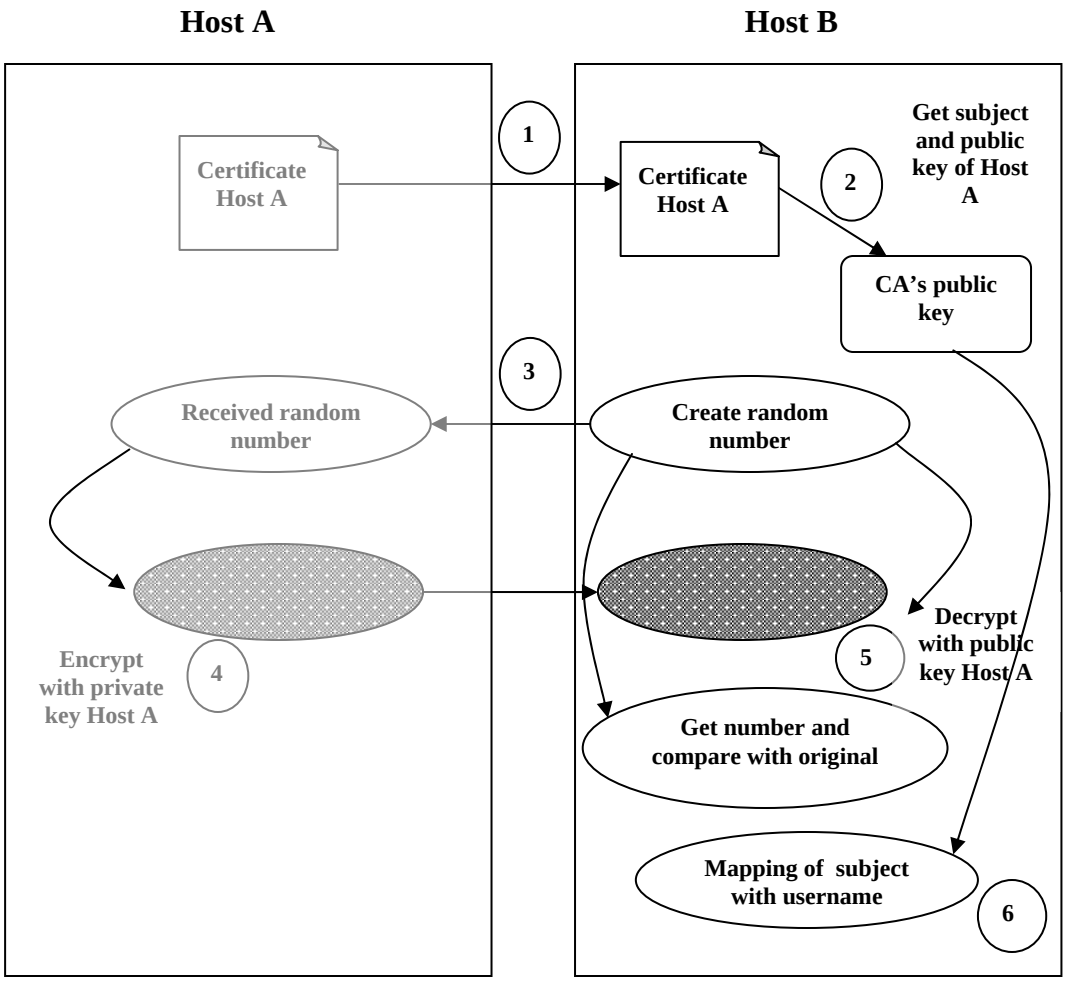

Figure 2.3: Authentication Procedure between two grid hosts

This short illustration shows that certificates and encryption keys (public or private) are the requested digital documents for grid authentication. The procedure consists in checking that the certificate and the keys of a given host are coherent.

Authorization mechanism: users that are willing to be authorized to use GSI-enable services need to belong to the GSI access control list. The GSI administrator verifies that the GSI Identity is owned by the username requesting the service.

2.3 Resource Management

The resource management [14] pyramid provides support for:

- Resource allocation
- Submitting jobs: Remotely running executable files and receiving results
- Managing job status and progress

Component of resource management are as follow:

- **GRAM**

Grid Resource Allocation Manager (GRAM) [15] Reports monitor and publishes information about the identity and state of local computations (registry). Moreover, it allows users to schedule and manage remote computations. Specifically, various classes and methods allow users to submit jobs, bind to already submitted jobs, and cancel jobs on remote computers. Other methods allow users to determine whether or not they can submit jobs to a specific resource (through a Globus gatekeeper) and to monitor the job status (pending, active, failed, done, and suspended).

A Grid may comprise more than one GRAM, each of them controlling a set of resources. By means of control or management we defer operations such as submission, monitoring, pausing or stopping. The job manager is created by the gatekeeper located on the remote computer and is responsible for starting and monitoring the job as well as for sending back to the client information regarding the changes in the job's status. A job manager exists for every client request and consists of a common component and a machine-specific component.

- **RSL**

The Resource Specification Language (RSL) [16], which is a structured language for specifying the resource requirements and parameters, is also parsed by GRAM. A gatekeeper is a process running as root on the server before any requests are sent from the client machine and its tasks are:

- Mutual authentication with the client
- Mapping the remote user to a local one
- Activating a job manager on the local host as a local user
- Pass the allocation arguments to the job manager

When the job is finished, the job manager sends the status information back to the client and terminates.

- **GASS**

Global Access to Secondary Storage (GASS) simplifies the porting and running of applications that use file I/O, eliminating the need to manually log onto sites and ftp files or to install a distributed file system. Globus provides an essential subset of GASS services to support the copying of files between computers (servers to client) on which the Grid Services are installed.

2.4 Information Services

The information is used for storing and retrieving information. Directory services are accessible via a network protocol. The Lightweight Directory Access Protocol, LDAP, is a directory service defining an information model and a protocol for querying and manipulating information in the directory. LDAP also include a hierarchical namespaces that defines the organization of the information. LDAP schemas specify what attributes the directory should contain. Each attribute has one type and zero or more values. For each such schema, there exists an information provider that generates the values for each attribute. These information providers are shell scripts. The values provided can either be static, e.g., the number of CPUs, or dynamic, e.g., the number of jobs in the queue of a batch system. Both the protocol and the information model defined in LDAP are extensible.

An information service for a grid environment must be able to handle a wide range of queries and many different types of resources. Furthermore, resource status and dynamic changes in VO membership must be handled as well as dynamic addition and deletion of information

sources. LDAP, and other existing directory services such as X.500 [17] and Universal Description, Discovery and Integration (UDDI) [18], do not fully meet these requirements. The design of Monitoring and Discovering Services (MDS), is an attempt to fulfill these requirements [19].

MDS consists of two parts, the Grid Resource Information Service, (GRIS), and the Grid Index Information Services (GIIS). MDS relies heavily on LDAP, both the GRIS and the GIIS are implemented as OpenLDAP [20] server back ends. Each grid resource runs a GRIS server that advertises static and dynamic information about the resource. Examples are CPU type, current load and available disk space. GIIS server registers themselves to one or more GIIS servers. GIIS servers can either dispatch incoming resource information requests to the appropriate GRIS server or cache information from each GRIS server for faster client access. Resource discovery is done by contacting a GIIS server and retrieving a list of available resources containing contacts to the GRIS server of each resource. All communication between a client and an MDS server is authenticated using the GSI infrastructure.

MDS supports a hierarchical structure for GIIS similar to the Domain Name Servers hierarchy. An example of a hierarchical structure is presented in Figure 2.4 where GRIS (on host B) registers GIIS (on host A) registers GIIS (on host B).

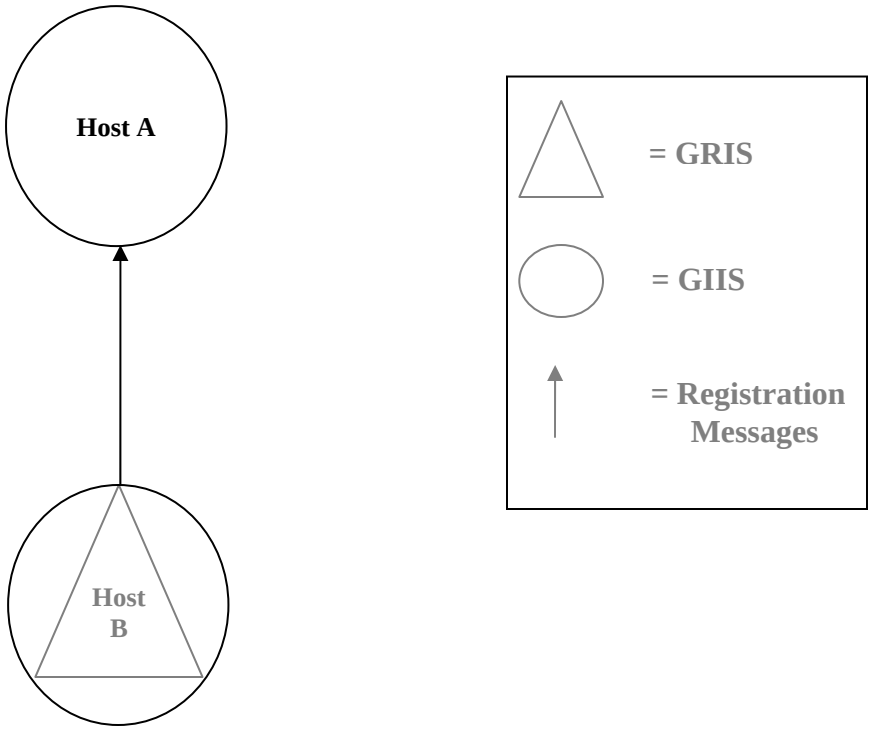

Figure 2.4: Overview of hierarchical GIIS structure

2.5 Data management

The data management [21] pyramid provides support to transfer files among machines in the grid and for the management of these transfers.

Data transfer: GridFTP [22] is a universal Grid data transfer and access protocol that provides a secure and reliable data transfer among grid nodes. It gives the members the possibility to act as a server or a client. This involves utilities such as GridFTP and globus-url-copy, which are used to move files between grid enabled. GridFTP is based on the FTP protocol [RFC 959] and provides a file transfer service with linked with grid security mechanisms. GridFTP Is the protocol proposed for all data transfers on the Grid. It extends the standard FTP protocol with facilities such as multistreamed transfer; auto tuning and globus based security. GridFTP must support Grid Security Infrastructure (GSI) and Kerberos authentication, with user controlled setting of various levels of data integrity and/or confidentiality.

Data access: the Global Access to Secondary Storage (GASS) subsystem provides the access to remote files. It allows programs to use the C standard I/O library to read, write files from remote computers. Copies of remote files opened for reading or writing are maintained in a local file cache with a database that keeps track of the local file name, access mode, URL and reference count.

Data replication: Globus Replica Management (GRM) [23] architecture is responsible for managing complete and partial copies of data sets. Data replication of great scientific interest as valuable data might be copied to several local storage to certify faster access.

Chapter 3

Resource Broker: Middleware between client and grid resources

The role of a broker in a grid environment can be very important. It is a component that will likely need to be implemented in most grid environments, though the implementation can vary from relatively simple to very complex .The basic role of a broker is to provide match-making services between a service requester and a service provider. In the grid environment, the service requesters will be the applications or the jobs submitted for execution, and the service providers will be the grid resources. With the advent of OGSA, the future service requester may be able to make requests of a grid service or a Web service via a generic service broker. A candidate for such a generic service broker may be IBM Web Sphere Business Connection, which is currently a Web services broker.

The Globus toolkit does not provide the broker function. It does, however, provide the grid information services function through the Monitoring and Discovery Service (MDS). The MDS may be queried to discover the properties of the machines, computers, and networks such as the number of processors available at this moment, what bandwidth is provided, and the type of storage available.

The term resource broker can only be applied to grid environments where the selection of resources to schedule jobs on is distributed. In a centralized system there exists only one entity (called a scheduler) that is responsible for the scheduling of all jobs, but in a decentralized system the scheduling actions of the resource management system are split over multiple entities. The distributed components that look for the appropriate resources are called resource brokers. An instance of a resource brokering component is associated with each application in the grid environment .The resource brokering component can also be associated with each resource in the system in systems where brokering is performed from the point of view of the resources but it is less common. Through out this thesis resource brokering is assumed to be performed for the application unless otherwise stated.

Grid computing is seen as the future of high performance computing but due to the distributed and heterogeneous nature of the resources many problems have yet to be overcome to see it fulfill its promise. This project is an investigation of brokering algorithms in a grid environment. Brokering is the act of discovering, scheduling and monitoring resources to perform the tasks that constitute an application running in the grid environment. While there has been a lot of development into the mechanisms vital to grid computing there is a growing need to develop methods for resource brokering, an area of grid computing which has been generally overlooked.

3.1 Globus Resource Management Architecture

Grid resource management as dealt by Globus is illustrated in Figure 3.1. Information about resource availability and characteristics is obtained from an information service.

- **Broker**

They are responsible for taking high-level RSL specifications and transforming them into more concrete specifications through a process called specialization. Multiple brokers may be involved in servicing a single request, with application-specific brokers translating application requirements into more concrete resource requirements, and different resource brokers being used to locate available resources that meet those requirements.

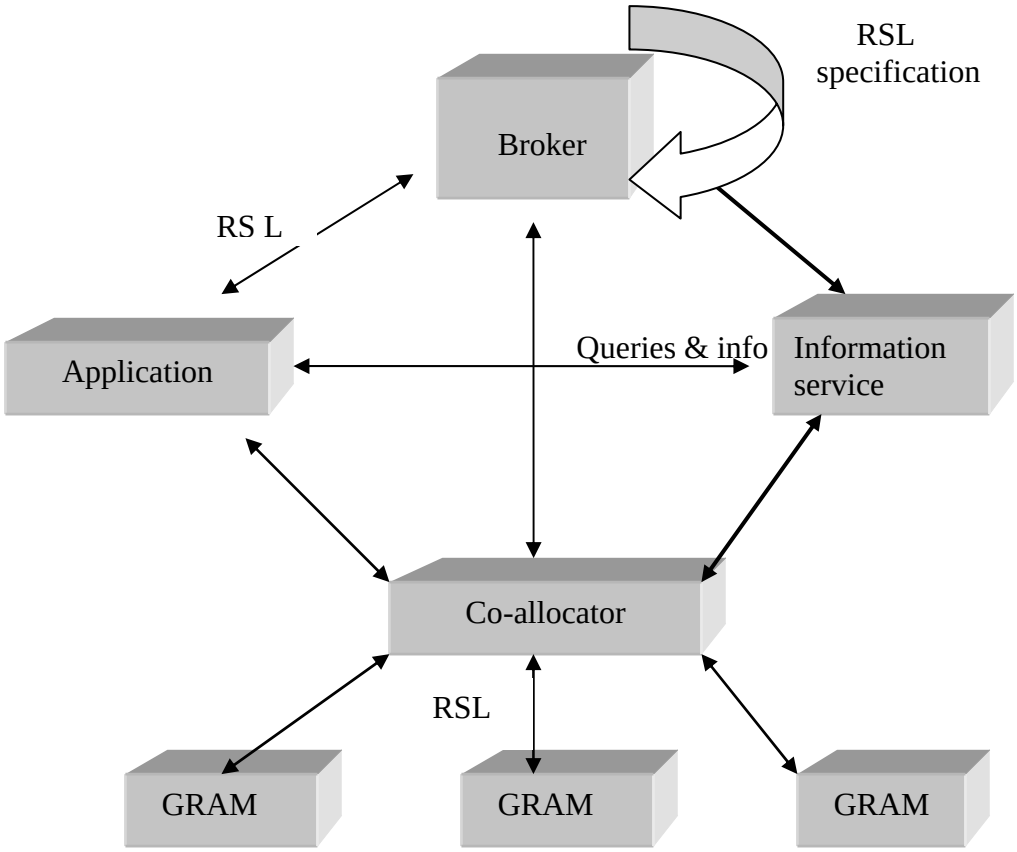

Figure 3.1: Globus Resource Management Architecture

- **Co-allocator**

Transformations effected by resource brokers generate a specification in which the locations of the required resources are completely specified. This ground request is passed to a co-allocator, which is responsible for coordinating the allocation and management of resources at multiple sites. Resource co-allocators break down a multirequest, that is, a request involving resources at multiple sites – into its constituent elements, and pass each component to the appropriate resource manager. DUROC – Dynamically-Updated Request Online Co-allocator – is a co-allocator that guarantees simultaneous allocation of a collection of resources while enabling the distributed collection of processes to be treated as a unit.

- **Grid Information Services**

This service is responsible for providing efficient and pervasive access to information on resource availability. This is used to locate specific resources with their details. The Metacomputing Directory Service (MDS) uses the data representation and API defined on LDAP to meet uniformity, extensibility and maintenance requirements. It provides access to static and dynamic information of resources. The MDS contains objects representing components infrastructure and computer resources and is responsible for defining a suitable data model for distributed computing applications.

- **Local Resource Management**

The Globus Resource Allocation Manager (GRAM) is the lowest level of Globus resource management architecture. GRAM allows you to run jobs remotely, providing an API for submitting, monitoring, and terminating your job. It provides the remote execution and status management of the execution. When a client is asked to run a job by a distant host, the request is sent to the client and handled by the gatekeeper daemon. It creates a job manager to start and monitor the job and once the job is finished, it has the responsibility to send the status information back to the client and terminate. It is responsible for parsing and processing Resource Specification Language (RSL) specifications, enabling remote management of jobs and updating MDS with information relative to resources availability.

- **Scheduling mechanisms**

Globus resource management architecture does not solve the scheduling problems of Grid computing. Computational Grids are emerging as a new computing paradigm for solving grand challenge applications in science, engineering, and economics [2]. Grid development involves the efficient management of heterogeneous, geographically distributed, and dynamically available resources. In this environment, the resource broker (or scheduler) becomes one of the most critical components of the grid middleware, since it has the responsibility of selecting resources and scheduling jobs in such a way that the user/application requirements are met, in terms of overall execution time (performance) and cost of the resources utilized.

In this chapter, we analyze in detail the main tasks that the resource broker has to tackle [25], like resource discovery and selection, job scheduling, job monitoring and migration, etc., and we examine different approaches to perform these tasks

3.2 Resource broker tasks

3.2.1 Finding the resource

The first task of the scheduler is resource discovery. The goal is to identify a list of authorized machines that are available to a given user. Most resource discovery algorithms interact with some kind of grid information service (GIS), like MDS (Monitoring and Discovery Service) in Globus [26]. This initial list of authorized resources can be filtered, in order to meet the minimum requirements of the application: hardware platform, operating system, minimum RAM or disk space requirements, etc

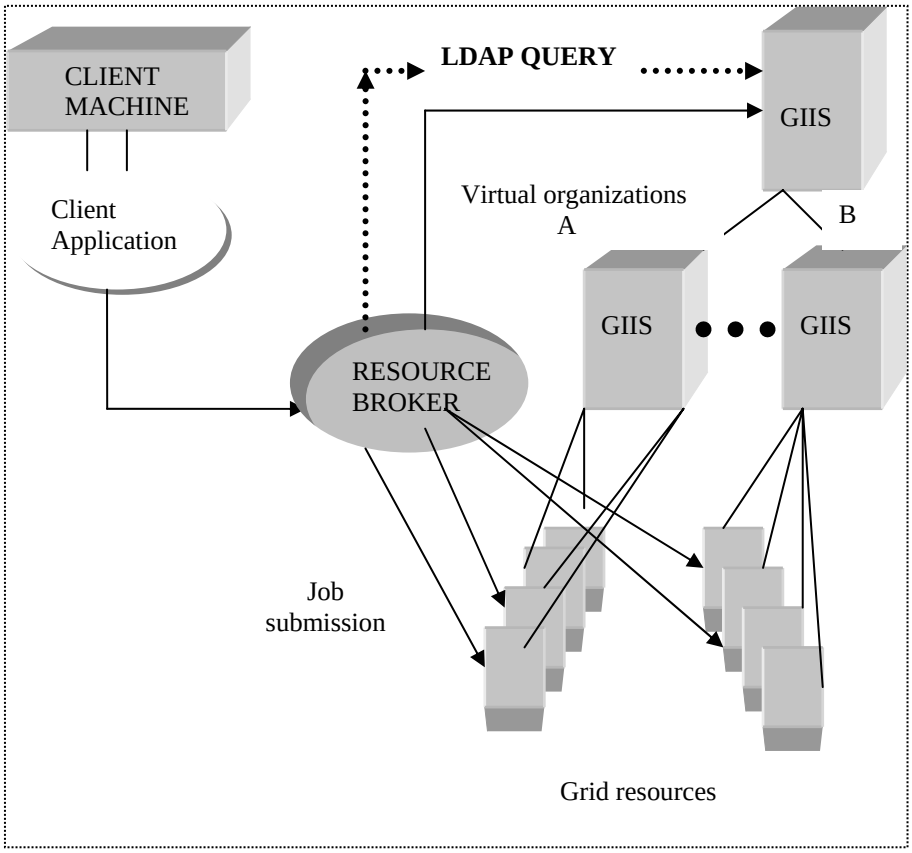

Figure 3.2: Resource Broker

3.2.2 Resource selection

Once the list of possible target machines is known, the second phase of resource brokering is the selection of those resources that are expected to meet the time or cost constraints imposed by the user. In order to fulfill the user time restrictions the resource broker has to gather dynamic information about resource accessibility, system workload, network performance, etc. This kind of information can be provided by different services, like for example NWS (Network Weather Service) [27]. However, in an economic environment, where the user can specify cost limitations, besides time constraints, the resource broker has to gather additional information about the price of the resources. Some environments like GRACE (Grid Architecture for Computational Economy) [28] provides a suite of trading protocols which enables resource consumers and providers to negotiate the cost of resources according to the expected starting time, the usage duration, the amount of memory or storage required, etc.

3.2.3 Job scheduling

The next stage for resource brokering is job scheduling, i.e., the mapping of pending jobs to specific physical resources, trying to minimize some cost function specified by the user. This is a NP-complete[35] (Buyya I, 2000). problem and different heuristics may be used to reach an optimal or near-optimal solution [29] We classify scheduling approaches in two major categories: performance-guided schedulers and economy-guided schedulers. Most of the grid systems in the literature fall in the first category, since they try to find a job-to-resource mapping that minimizes the overall execution time (i.e. optimizes performance). One of the simplest performance-guided scheduling algorithms is the greedy or opportunistic approach [30] [31], which iteratively allocates each job to the machine that is likely to produce the best performance, without considering the rest of pending jobs. This approach leads usually to sub-optimal solutions, since scheduling decisions are based only on local job information. Other more complex scheduling algorithms, which explore the solution space and try to overcome local optimal solutions, are based on genetic algorithms [28][32], simulated annealing [30][33], tabular search [30], or branch and bound methods [34].

On the other hand, economy-guided schedulers include cost of resources as optimizing criteria [35][36]. For example the Nimrod/G broker [35] allows users to specify a budget

constraint (cost of resources), a deadline constraint (execution time), or both, and it incorporates three adaptive scheduling algorithms for cost optimization, time optimization, or conservative time optimization, within time and budget constraints. In the most general case, scheduling algorithms have to adapt to the different optimization criteria that a user can specify for each particular job. Some of the most frequent optimization criteria important for a user are the following:

- optimizing performance without regarding to cost
- optimizing cost without regarding to performance
- optimizing performance, within a specific cost constraint
- optimizing cost, within a specific time constraint
- optimizing performance, within a specific cost and time constraint
- optimizing cost, within a specific cost and time constraint

3.2.4 Job monitoring and migration

A grid is inherently a dynamic system where environmental conditions are subjected to unpredictable changes: system or network failures, system performance degradation, addition of new machines, variations in the cost of resources, etc. In such a context, job migration is the only efficient way to guarantee that the submitted jobs are completed and that the user restrictions are met. The major tasks involved in migration are job monitoring, re-scheduling, and check pointing. Job monitor is responsible for detecting alert situations that could initiate a migration. This information is reported to the re-scheduler, which evaluates if it is worth migrating the job, and in that case, decides a new allocation for the job. Check pointing is the capability of capturing periodically a snapshot of the state of a running job, in such a way that the job can be restarted from that state in a later time in case of migration. Most of the systems dealing with job migration face up to the problem from the point of view of performance [37][38]. The main migration policies considered in these systems include, among others, performance slowdown, target system failure, job cancellation, detection of a better resource, etc. However, there are hardly a few works that manage job migration under economic conditions [36]. In this context, new job migration policies must be contemplated, like the discovery of a new cheaper resource, or variations in the resource prices during the

job execution. There are a broad variety of reasons that could lead resources to dynamically modify their prices, for example:

- Prices can change according to the demand. Resources with high demand can increase their fares, and vice versa.
- Prices can change according to the time or the day. For example the use of a resource can be cheaper during the night or during the weekend.
- A provider and a consumer can negotiate a given price for a maximum usage time or a maximum amount of resources consumed (CPU, memory, disk, I/O …). If the job in execution violates the contract by exceeding the time or the resource limit, the provider can increase the price.
- Resource provider can announce a special offer to attract new consumers, reducing the price of the resources during a specific time period

3.3 Motivation behind the work

The Globus Toolkit does not include a broker service .Therefore; a Grid application built with the Globus Toolkit must develop its own job broker or use third-party software. The broker service provides a link between the resources of the Grid topology and the workload generated by the application. Third-party software such as schedulers PBS, Platform Computing LSF, or Condor often provides broker functionality for a Globus grid. My broker is a software service that can allocate Globus resources and assign a job to one of these resources. The Monitoring and Discovery Service (MDS) provides access to the system configuration and status information about grid resources. A broker needs to use the MDS to find out which resources are available. Through the MDS, the broker also knows the system configuration of the resources and can intelligently assign a job to the appropriate resource.

3.4 Application Enablement considerations

When designing an application for execution in a grid environment, it is important to understand how resources will be discovered and allocated. It may be up to the application to identify its resource requirements to the broker so that the broker can ensure that the proper and appropriate resources are allocated to the application.

3.5 Proposed Algorithm

Input: User's RSL job request(s)[Fig. 3.2]

Action: Find, select and allocate the resource most appropriate for the job

Output: none.

1. Process RSL job request.
2. Contact one or more GIIS servers to obtain a list of available resources.
3. Query the GRIS of each resource for the hardware as well current state of that resource jobs and loads
4. For each job :
    (a) Select the resource to which job will be submitted.
    (b) Submit the job to the selected resource.

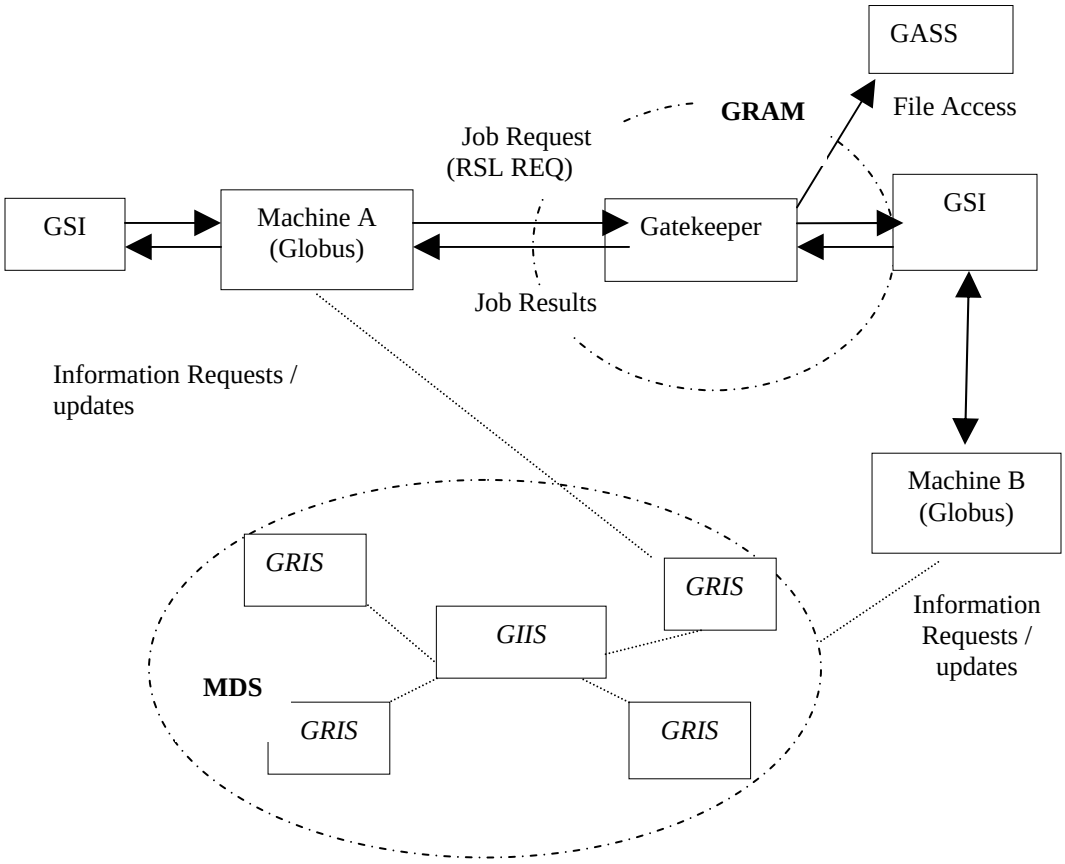

Figure 3.2: Overview of Globus Gatekeeper

3.6 Brokering Techniques

The resource brokering projects outlines above have shown various different techniques for performing the allocations of application tasks to resources. These techniques can be grouped into two types:

- Purely application performance maximization
- The notion of a computational economy.

The view from the application is that it wishes to have its tasks completed in the shortest period of time. If brokering is considered purely from this standpoint it leads to brokers searching for the most powerful resources and using them, second come will either have to look somewhere else or wait. Note that consideration has to be taken here whether the resource is willing to provide service to an arbitrary amount of tasks, in which performance will drop off as more tasks are serviced, or whether the resource undertakes to guarantee a certain rate of service that is agreed upon when the task is submitted.

The other technique that is seen as the next step in resource brokering as it not only considers the applications needs but also the need of the entire system is the use of a computational economy where abstract currency is exchanged in return for services. This also has the advantage that it can be applied when there is a real cost for using resources as is foreseen in the future when grid computing becomes more wide spread (there is of course already a cost for using resources). This can be further broken down into the idea of a commodities market where certain services have a market driven cost or an auctioning scheme. An auctioning scheme can be from the viewpoint of the resources wishing to buy tasks to perform or of the applications wishing to buy computation to perform tasks. The seller advertises and buyers bid for the item using private bids, in most schemes the highest bidder is awarded the item at the price of the second bid, this is equivalent to the highest bidder just outbidding the second. This auctioning scheme in a grid environment requires a central component equivalent to eBay TM which hosts the auctions. Sellers advertise their wares and buyers search for items they wish to buy, on finding something of value they place a bid before the auction finishes. When the auction concludes the bidder with the highest bid is awarded the item. If the act of

bidding is performed by the resource they will need to be continuously monitoring when they will have the ability to host more tasks (i.e. when will they finish) and comparing the time that this will occur to the closing times of auctions. If the bidding is performed by the applications a slightly different approach can be taken.

Applications can discover a set of resources that it thinks are appropriate and then ask only those to bid. This leads to a much quicker auction than advertising to a central host but removes the act of discovering resources from the auctioning process .The commodities market concept is where current research is heading as it has be studied in depth in the field of economics and is considered to be well known. By modeling a grid system where application tasks signify goods that are either bought or sold at a market rate an efficient scheme is hoped to be created for the use of grid resources. In the future when grid computing is adopted for more than just research there will be a real cost for using resources, this will lead to the grid economy using actual currency rather than an abstract one and this cost will of course be the overriding factor in decision making in all but a few cases. Could the equivalent of the pay phone be introduced to supercomputing in the future such that supercomputing becomes accessible to anyone that has a thin client with a slot to put coins.

3.7 Dynamic Brokering

Having scheduled tasks onto resources we then want to be able to monitor those tasks to make sure that they are progressing as they should be. This can be performed by actively polling the resource performing the task or by the resource notifying the application when there is a change in the state of the task or the performance of the resource. The action of monitoring tasks in the view of possibly changing resource allocations is called dynamic brokering. When the performance of a resource degrades to such an extent that the application would benefit from moving the task to a different resource there needs to be a mechanism for the application to be notified. This can be done by the application setting thresholds for the resource performance, when the performance crosses that threshold then the resource signifies this by creating a threshold event which is sent to the application so that the application can consider whether it wishes to take some action. This could be process migration but could also be restarting the task on a different resource that will hopefully

complete the task sooner. With the creation of threshold events as a mechanism for monitoring resource performance two situations need to be considered, these are when the resource has guaranteed to provide performance at a certain rate or when a resource simply takes new tasks based on the new task wanting to use it.

The second case would very rarely be acceptable, simply because a new tasks doesn't mind a lower rate of performance, all of the other tasks must cope with it as well. Thus, this situation is ignored as when a tasks is submitted to a resource it is assumed to have been given some guaranteed rate of performance.

With a guarantee for the rate of performance there is two cases, that the performance has dropped below this level or that the task was getting a higher rate which has dropped but is still above the guaranteed rate.

Chapter 4

Establishment of Grid Computing Environment

"Grid" computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

This chapter describes the implementation of grid computing environment which include basic requirements for setting up a grid computing environment, how to setup an initial grid and how to maintain and expand the grid.

Before enabling grid environment some of the planning considerations that should be taken into consideration [31]. This includes:

- Planning for installation
- Planning for security
- Planning for related software
- Planning a production environment
- Planning a development environment

4.1 System Requirements

The hardware and software required for the Globus Toolkit 2.4 are described as follows:-

- **Hardware**

In order to build, install, and run the Globus Toolkit on system, the following hardware are taken into consideration:-

CPU

The Globus software itself is not CPU intensive, but the computing power required to run the Globus Toolkit depends on what kind of host system used .

Physical Memory

The Globus Toolkit itself is not memory intensive; therefore, the hosts on which it will run need only have a nominal amount of memory for the sake of the Globus Toolkit code.

Disk Space

Disk space requirements for building, installing, and deploying the Globus Toolkit can vary depending on the number of architectures and the number of development libraries that are built. Thus only approximate disk space requirements can be given.

- **Software**

The software on which the Globus Toolkit depends and what additional software is recommended and why are as follows:

In order to install and run the Globus Toolkit, SSLeay and OpenLDAP are used.

SSLeay. Globus Toolkit can be compiled using SSLeay. The Globus Security Infrastructure (GSI) is implemented in terms of a Generic Security Service Application Program Interface (GSSAPI) that is built on top of the SSLeay package.

OpenLDAP. The Globus Toolkit uses modified libraries from the OpenLDAP distribution, though this will change in the future as the Globus Toolkit modifications are rolled back into OpenLDAP.

4.2 Installation Considerations

There are several issues need to be consider before beginning to install the Globus Toolkit. They are as follows:

4.2.1. Choosing a Host

Choosing the host on which the Globus Toolkit will be installed depends on how it will be used. If no resources will be contributed to the Globus grid resource pool, then the host on which the Globus Toolkit is installed is a matter of convenience. In that case, it is assumed that the Globus Toolkit will be used primarily for something like single sign-on

authentication or to provide access to other resources in the Globus grid. Therefore, no special requirements are needed of the host on which the Globus Toolkit will be installed. If the host on which the Globus Toolkit is installed is to be made available as part of the Globus grid resource pool, then consider issues such as computing power, disk space, and memory. However, running the Globus Toolkit on the same resource is acceptable because the Globus Toolkit is not CPU intensive and should not have an impact on any Globus grid jobs requesting use of this host.

4.2.2 Choosing File systems

There are three considerations for choosing the appropriate file systems for system on which to build, install, and deploy the Globus Toolkit:

1. The Globus Toolkit is better installed on a *shared* file system.
2. Each host running a Globus Toolkit gatekeeper must be able to accommodate deployment of the Globus Toolkit on its local file system.
3. The deployment location must have room for the log files.

4.2.3. Contributing resources

Contribution of resources to the Globus grid resource pool depends on the impact of Globus Toolkit job requests on these resources. There are some methods:

- **Local Site Policies**

Many sites have policies regarding the use of their resources. These policies will have an impact on the choice of resources to contribute and the extent to which they can be made available. Consider the limits on the amount of physical resource such as CPU, memory, and disk; limits on the number of jobs a user can submit or run; and the cost and level of service. Consider how and if these policies can be enforced on Globus Toolkit jobs.

- **Level of Service**

GRAM provides a convenient way of submitting and monitoring jobs remotely. The level of service provides for Globus Toolkit jobs is determined as it would be for any other application running.

- **Accounting**

Resource usage accounting is presumed to be handled locally by each site. The Globus Toolkit does not change any existing local accounting mechanisms. Globus Toolkit jobs run under the user account as specified in the grid-mapfile. Therefore, a user is required to already have a conventional Unix account on the host to which the job is submitted.

4.2.4 Information Services

Globus Toolkit includes two specialized types of information services:

GRIS

The GRIS is a distributed information service that can answer queries about a particular resource by directing the query to an information provider deployed as part of the Globus services on a grid resource. Examples of information provided by this service include host identity (e.g., OS and versions), as well as more dynamic information such as CPU availability. Each resource on which the Globus Toolkit is installed will run a GRIS. Therefore, each GRIS is responsible for providing information only about the resource on which it is running. By default, a GRIS is automatically configured and will be assigned to use port 2135.

GIIS

The GIIS represents a centralized MDS server that provides information about all of resources. The normal configuration for an organization would be to have a GRIS on each resource running Globus and one GIIS managed by the lead site for that organization.

4.2.5. Security

The Globus Toolkit uses an authentication system known as gssapi_ssleay, the Generic Security Service API based on Eric A. Young's implementation of Secure Sockets Layer (SSL). This system uses the RSA encryption algorithm[] for its encryption, therefore employing both public and private keys.

4.2.5.1 X.509 Certification Process

The gssapi_ssleay authentication relies on an X.509 certification process. Globus Toolkit users place their X.509 certificates in their home directories, thus identifying themselves to the system.

- **User Certificates and Keys.** The X.509 certificate includes information about the duration of the permissions, the RSA public key, and the signature of the Certificate Authority (CA). The certificates can be created only by the CA, who reviews the X.509 certificate request submitted by the user and accepts or denies it according to an established policy.

- **Gatekeeper Certificates and Keys.** The gatekeeper also must have a certificate and key. They are requested and created in a like manner by the system administrator using the Globus Toolkit certificate request generation script.

- **Proxies**: The gssapi_ssleay authentication requires the use of proxies, a convenient mechanism for reducing the number of times users must enter their pass-phrase. Proxy files must be kept secure, within system's local file system, rather than on the Network File System (NFS). They must allow only the user to have read-access to them, as they essentially allow job submission without pass-phrase protection, a feature that can potentially compromise the security of the system.

  A proxy consists of a new certificate and a private key. The key pair that is used for the proxy, i.e. the public key embedded in the certificate and the private key, may either be regenerated for each proxy or obtained by other means. The new certificate contains the

owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA. The certificate also includes a time notation after which the proxy should no longer be accepted by others. Proxies have limited lifetimes.

4.2.5.2 Environment Variables

The Globus Toolkit locates the certificate and key files in the user's ~/.globus directory by default or by referring to environment variables. The basic environment variables that can be set:

GLOBUS_LOCATION

The GSI libaries use GLOBUS_LOCATION as one place to look for the trusted certificates directory. The location $GLOBUS_LOCATION/share/certificates is used if X509_CERT_DIR is not set and /etc/grid-security and $HOME/.globus/certificates do not exist.

GRIDMAP

This environment varibale can be used to override the default location of the grid-mapfile, which is normally /etc/grid-security/grid-mapfile.

X509_CERT_DIR

This environment variable can be used to override the default location of the trusted certificates directory, which is normally /etc/grid-security/certificates.

X509_USER_DELEG_PROXY

This environment is set by the GSI libaries to point at the location of credentials that it recieves during delegation. Application servers usually then copy this value to 509_USER_PROXY and users generally never see it. Setting this value has no effect.

X509_RUN_AS_SERVER

If this environment variable is set(to any value) it causes the GSI libaries not to look for a proxy credential unless X509_USER_PROXY is explicity set. The intent is for this to be used with servers that should always use a given certificate and private key.

X509_USER_CERT

This environment variable can be used to override the default location of the certificate file. For users this is normally $HOME/.globus/usercert.pem. For servers this is normally /etc/grid-security/hostcert.pem.

X509_USERS_KEY

This environment variable can be used to override the default location of the private key file. For users this is normally $HOME/.globus/userkey.pem. For servers this is normally /etc/grid-security/hostkey.pem.

X509_USER_PROXY

This environment variable can be used to override the default location of the user proxy credential, which is /tmp/ *x509up_u<uid>*.

X509_CERT_FILE

File in which one or more trusted certificates are stored (not normally used)

4.3 Required Software

Globus Toolkit Version 2.4 is used for the setup of grid. Globus Toolkit supports Red Hat Linux 9.0 [32] on xseries.

The list of required files to be downloaded [33]:

- Globus Packaging Technology
  gpt-3.0.1-src.tar.gz
- Globus client

globus-all-client-2.4.3-i686-pc-linux-gnu-bin.tar.gz

- Server bundle
  globus-all-server-2.4.3-i686-pc-linux-gnu-bin.tar.gz
- Certificate Authority
  globus_simple_ca_bundle-latest.tar.gz
- Network Time Protocol
  ntp-4.1.1-1.i386.rpm

4.4 Lab environment

This section provides an overview of the configuration of the software and hardware used in our lab. It is a simple Grid environment. We used an Ethernet LAN with four server xSeries machines named alpha, beta, gamma, delta and one client zeta on the LAN. We made alpha server a certificate authority (CA Server). These machines were installed with the Red Hat 9.0 Linux distribution. Figure 4.1 illustrates this environment with the host names and the functionality of each machine.

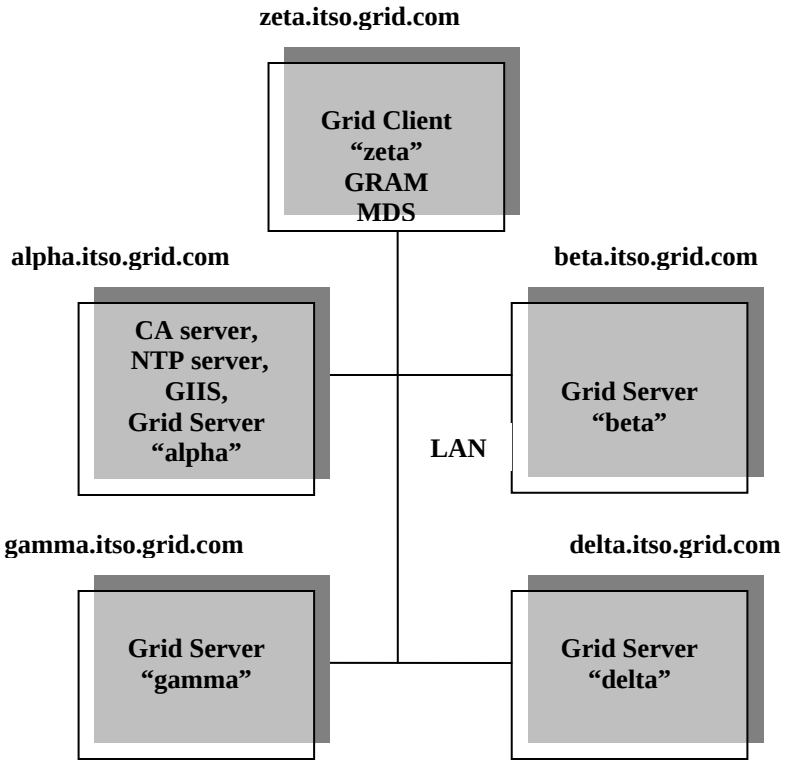

Figure 4.1: Hardware environment and software functions of each machine

4.4.1 Naming and addressing

Table 4.1 summarizes the names of the machines, their IP addresses, the Linux distribution used, and their primary functions.

Table 4.1: Host names and IP addressing

Host name	IP	Linux distribution	Function
alpha.itso.grid.com	192.168.10.201	Red Hat 9.0	CA server, NTP server, Grid Server
beta.itso.grid.com	192.168.10.224	Red Hat 9.0	Grid Server
gamma.itso.grid.com	192.168.10.204	Red Hat 9.0	Grid Server
delta.itso.grid.com	192.168.10.223	Red Hat 9.0	Grid Server
zeta.itso.grid.com	192.168.10.221	Red Hat 9.0	Grid Client

Table 4.2 describes the distinguished name used for the Certificate Authority in our environment:

Table 4.2: CA distinguished name and passphrase

Certificate Authority DN	Passphrase
cn=my test CA, ou=alpha.itso.grid.com, ou =grid	*********

The distinguished name (DN) and passphrase will be used by the Certificate Authority to sign certificate requests.

Table 4.3 describes suggested user and group IDs and passwords.

Table 4.3: User ID and group ID

User ID	Group ID	alpha password	beta password	gamma password	delta password	zeta password
root	root	***	***	***	***	***
globususer	globus	***	***	***	***	***
snobol	snobol	***	***	***	***	***
adminca	adminca	***	No-id	no-id	no-id	no-id

The root ID is used on all machines. A cell containing “no-id” means that the corresponding machine does not have that user ID installed on it. The globususer ID is used to run jobs on the grid for the user and to FTP files. The snobol ID is used to submit jobs to the grid. The adminca ID is used to receive certificate requests for the Certificate. The certificates will be signed using the root ID on machine alpha.

4.5 Setting up the Linux requirements

This section describes the steps that are required to install the Linux environment for using the Globus toolkit 2.4. The major steps to set up this environment are:

4.5.1 Linux Setup

The Linux operating system is used in many ways including support for networking, software development, servers and desktop platforms and is considered as a low cost alternative to other operating systems.

Install Linux on all machines that will be part of the grid. In our lab, we install Red Hat 9.0 on five machines choosing the default installation distributions with no firewall protection so

that network requests (such as RPC) would not be hindered when we needed to access the infrastructure server.

4.5.2 Configure Network Time Protocol (NTP)

For the grid to work properly, the system clocks must be synchronized using NTP server. GSI certificates use GMT and is very sensitive to the time The grid security process creates proxy certificates that are valid for specific times. If the system clocks are not synchronized, the proxy certificates may appear as if they have expired and users may not be able to use the grid. In our lab environment we used an NTP server on machine alpha.

4.6 Installation and Configuration Globus

1. Declare where to install Globus via the environment variable GLOBUS_LOCATION; define GPT_LOCATION to indicate where to install gpt.
2. Install gpt and the necessary client and/or server packages according to the Globus installation instructions.

4.6.1 Setup of Own certificate Authority

The Globus Project provides the Globus Simple CA, convenient way of setting up a certificate authority. Script was installed on alpha to set up a new SimpleCA. Run this script once per Grid.

$GLOBUS_LOCATION/setup/globus/setup-simple-ca

This command creates a new CA 1024-bit RSA private key (CA.key) and certificate (CA.cert) with lifetime 1825 days.

- ***Configure the subject name***

This script prompts information about the CA .The unique subject name for this CA is:
cn=my test CA, ou=alpha.itso.grid.com, ou=demotest, o=Grid

Table 4.4: CA Name components

cn	Represents "common name". Identifies this particular certificate as the CA certificate within the "GlobusTest/simpleCA-hostname" domain, which in this case is my test CA.
ou	Represents "organizational unit". Identifies this CA from other CAs created by SimpleCA by other people. The second "ou" is specific to your hostname (in this cases demotest).
O	Represents "organization". Identifies the Grid.

- ***Configure the CA's email and pass phrase***

The email used by CA, receive certificate requests. It should be real email address of the administrator not the address of users.

It requests a pass phrase to protect the key, The passphrase of the CA certificate will be used only when signing certificates (with grid-cert-sign).

- ***Confirm generated certificate***

A self-signed certificate has been generated for the Certificate Authority with the subject:

/O=Grid/OU=demotest/OU=alpha.itso.grid.com/CN=my test CA

The private key of the CA is stored in :

/home/globus/.globus/simpleCA//private/cakey.pem

The public CA certificate is stored in:

/home/globus/.globus/simpleCA//cacert.pem

The distribution package built for this CA is stored in:

/home/globus/.globus/simpleCA//globus_simple_ca_11116_setup-0.13.tar.gz

The number 1116 is known as CA hash. It will be an 8 hexadecimal digit string.

- ***Complete setup of GSI***

To finish the setup of GSI, run the script on each of machine as root:

$GLOBUS_LOCATION/setup/globus_simple_ca_<hash>_setup/setup-gsi –default

4.6.2 Host Certificate

In order to use resources in a grid, host must first request and install security certificates from a reputable certificate authority (CA).

Request and sign a host certificate and then copy it into the appropriate directory for secure services. The certificate must be for a machine which has a consistent name in DNS; you should not run it on a computer using DHCP where a different name could be assigned to computer.

- ***Request a host Certificate***

On each of the server machines (alpha, beta , gamma, delta) as root , the administrator issue the command.

grid-cert-request -host <hostname of requesting machines>

This creates the following files:

- /etc/grid-security/hostkey.pem
- /etc/grid-security/hostcert_request.pem
- /etc/grid-security/hostcert.pem(empty)

- ***Signing host Certificate***

CA machine (alpha) sign the request certificate as root, the administrator issue the command.

grid-ca-sign -in /root/hostcert_request.pem -out /root/hostcert.pem

The file hostcert.pem contains the certificate and should be sent back to the user and should be saved in the directory /etc/grid-security/certificates. The administrator should verify the identity of the user. The certificate should be owned by the user, and read-only for other users.

4.6.3 User Certificates

For each user using the grid, must request user certificates which will sign using the *globus* user.

- ***Request a user Certificate***

User snobol on machine zeta, run the command:

grid-cert-request

This creates the following files:

- /home/snobol/.globus/userkey.pem
- /home/snobol/.globus/usercert_request.pem
- /home/snobol/.globus/usercert.pem(empty)

- ***Signing user Certificate***

CA machine (alpha) sign the request certificate as root, the administrator issue the command.

grid-ca-sign -in /root/usercert_request.pem -out /root/usercert.pem

The file usercert.pem contains the certificate and should be sent back to the user and should be saved in the directory /etc/grid-security/certificates. The administrator should verify the identity of the user. The certificate should be owned by the user, and read-only for other users.

4.6.4 Adding a Grid mapfile entry

Certificates have been signed and installed, users must be added to the grid mapfile so that they can access resources on a grid host. The mapping consists of associating a grid user's DN with a local user on the host.

4.6.5 Testing

To verify the Simple CA is installed in /etc/grid-security/certificates and that certificate is in place with the correct permissions, run:

user$ grid-proxy-init -debug –verify

A Full Proxy has been created by grid-proxy-init which can be use to perform various grid operations.

4.7 Setting up the gatekeeper

On each server (alpha, beta , gamma, delta) , aappend to /etc/services the following lines.

```
gsigatekeeper   2119/tcp        # Globus Gatekeeper
gsiftp          2811/tcp        # GsiFTP
```

In the file /etc/xinetd.d/gsigatekeeper on each server, containing the lines:

```
service gsigatekeeper
{
 socket_type =  stream
 protocol     =   tcp
 wait          =   no
 user          =   root
 env           =   LD_LIBRARY_PATH=${GLOBUS_LOCATION}/lib
 server        =   ${GLOBUS_LOCATION}/sbin/globus-gatekeeper
 server_args  =    -conf \ ${GLOBUS_LOCATION}/etc/globus-gatekeeper.conf
 disable       =    no
}
```

In the file /etc/xinetd.d/gsiftp on each sever, containing the lines:

```
service gsiftp
{
 instances          = 1000
 socket_type        = stream
 wait               = no
 user               = root
 env                = LD_LIBRARY_PATH=${GLOBUS_LOCATION}/lib
 server             = ${GLOBUS_LOCATION}/sbin/in.ftpd
 server_args        = -l -a -G ${GLOBUS_LOCATION}
 log_on_success  += DURATION USERID
 log_on_failure   += USERID
 nice               = 10
 disable            = no
}
```

4.8 Setting up MDS

Monitoring and Discovery Service (MDS) is based on OpenLDAP, allowing to create own configuration of hierarchical GIIS. MDS have one Grid Information Index Service (GIIS) in the alpha machine, which collects the data reported by the Grid Resource Information Servers (GRIS) in all of the machines. The GRIS servers send information about their respective servers to the GIIS. The user will be able to query the GIIS from the zeta client machine.

To set up this structure, modification of several configuration files is be done. These files name the GIIS and GRIS, and show how these components should register with each other. Figure 9-2 shows the relationship among the MDS components:

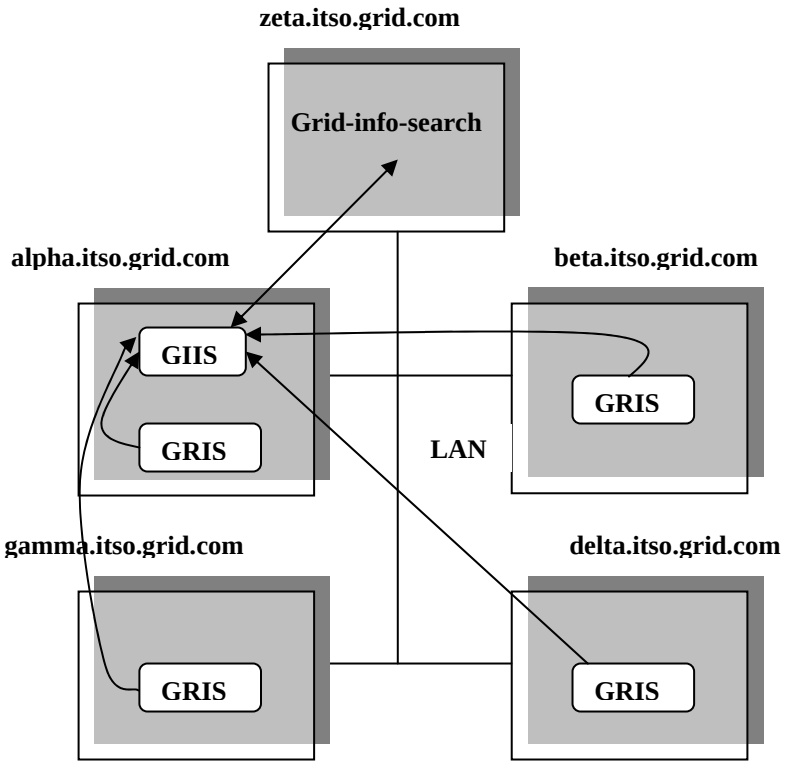

Figure 4.2: MDS Configuration

GIIS are set on the alpha machine and GRIS are set on all the other server (alpha, beta , gamma , delta).

4.8.1 MDS on client

Modiified the $GLOBUS_LOCATION/etc/grid-info.conf file lines as shown below so that the searches go to the GIIS on machine alpha.

GRID_INFO_HOST = “alpha.itso.grid.com”

GRID_INFO_ORGANIZATION_DN”=”Mds-Vo-name=alpha.itso.grid.com, o=Grid”

4.8.2 Secure MDS

This MDS permits anonymous access. The grid-info-search command should use the -x flag to indicate an anonymous search request. The MDS can be secured so that only certified users can access the GIIS and only certified server GRISs can register to send information to the GIIS. Each resource will have its own LDAP information service that can be connected to remotely for obtaining system and status information. The GIIS can only be configured to run on a host on which the Globus Toolkit will be installed.

4.8.2.1 Request a LDAP certificate

On each of the server machines (alpha, beta , gamma, delta) as root , the administrator issue the command.

grid-cert-request –service ldap -host <hostname of requesting machines>

This creates the following files:

- /etc/grid-security/ldap/ldapkey.pem
- /etc/grid-security/ldap/ldapcert_request.pem
- /etc/grid-security/ldap/ldapcert.pem(empty)

4.8.2.2 Signing LDAP certificate

CA machine (alpha) sign the request certificate as root, the administrator issue the command.

grid-ca-sign -in /root/ldapcert_request.pem -out /root/ldapcert.pem

The file ldapcert.pem contains the certificate and should be sent back to the user and should be saved in the directory /etc/grid-security/ldap. The administrator should verify the identity of the user. The certificate should be owned by the user, and read-only for other users.

4.9 Verification

4.9.1 Server interface

Installation on each machine can be check as root using the command:

$GPT_LOCATION/sbin/gpt-verify

GRAM can be check by listening on their port:

netstat –an | grep 2119

The command used to check the secure MDS are:

globus-mds start

globus-mds stop

4.9.2 Client interface

On client machine (zeta) , logged on as the user snobol and sets up the enviroment so that globus command can be issued by the user.This line is be added to one's login profile:

. $GLOBUS_LOCATION/etc/globus-user-env.sh

Proxy is created with the command:

grid-proxy-init

Client interface for GRAM , MDS and Grid FTP are discussed below

- **GRAM**

GRAM has the following client commands to submit and manage jobs on the grid environment.

- **globus-job-run**

This is an online interface for job submissions. It is the easiest command to use to submit a job and returns the output of its result.

The basic command syntax is:

globus-job-run <contact string> <command>

where <contact string> specifies a machine's host name, port, and service to which to send the request.

The syntax of a contact string is host:port/jobmanager-name. The default port is 2119 and the default job manager's name is "jobmanager".

- **globus-job-submit**

This is an interface for batch job submissions. It will immediately return with an URL (with the job contact string embedded) that can be use to query the status of job. The command is similar to *globus-job-run*, but the *globus-job-submit* command does not return the output of its result. To obtain the output, run the job management commands *globus-job-status*, *globus-job-get-output*, and *globus-job-clean/globus-job-canc1e* pointing to the URL generated as result of the *globus-job-submit* execution.

The basic command syntax is as follows:
globus-job-submit <contact string> command

- **globusrun**

This is a command that gives access to the RSL, the language which provides a common interchange to describe resources [31]. The *globus-job-run* and *globus-job-submit* commands are both shell script wrappers around *globusrun.*

The basic command syntax is:
globusrun <contact string> <RSL>

- **globus-job-status**

This is a job management command that returns a job status of one of the following:

- pending
- active
- done
- failed
- others

- **globus-job-get-output**

This is a job management command that collects the output when the job finishes.

- **globus-job-clean/globus-job-cancel**

This is a job management command that stops the job if it is running, and cleans up the cached copy of the output.

➢ **MDS (GRIS and GIIS)**

MDS has a client command to query for details about resources in the grid environment.

- **grid-info-search**

This command sends one or more queries to GRIS and GIIS and displays the result in the standard output. The queries are RFC1558 compliant with the LDAP search filter, since the command embeds the *ldapsearch* command.

The basic command syntax is:

grid-info-search [options] <filter> [attributes...]

➢ **GridFTP**

GridFTP provides a client command to copy files between local and remote locations.

- **globus-url-copy**

The basic command syntax is:

globus-url-copy [options] <source URL> <destination URL>

Where:

<source URL> is the URL to source file, or '-' for standard input.

<destination URL> is the URL of the destination file, or '-' for standard output.

4.10 Setting up grid applications

In grid platforms, setting up a grid application should be very straight forward and should not require additional remarks about non-trivial issues. Setting up grid applications that the grid administrator should be aware of are as follows:

4.10.1 Deploying an application

Grid applications are single instruction multiple data (SIMD) programs. Most of the computing demanding applications have this feature and that, in a loosely coupled distributed system, the data-parallelism tends to be more efficiently exploited. The deployment of an application has two distinct phases:

Code deployment This phase is performed when the application is first deployed to the grid or when the code is modified and has to be updated.

Data deployment This phase has to be performed every time a new execution is issued. Data deployments are more time-consuming, more frequent, and while they are being performed, the application stands idle waiting for the data to arrive. For this reason, there are a few things that are worth mentioning when discussing application deployments:

- Some grid platforms make it possible for multiple applications to be executed simultaneously; if this is the case, application deployments do not cause much impact, as the grid does not have to be idle while they are performed.

- Some few applications are capable of dealing with streaming data, and some grid platforms do support this sort of application (the application start processing the data as soon as it gets to the nodes). If a single-application grid is to be set up and its application works this way, adopting a streaming enabled grid platform is something to consider.

- Deployments should be ultimately performed by the system administrator, but the platform might make facilities available for the application developers to submit their application code and data so that every deployment is correctly logged and assigned to its developer.

Application deployments should not be a serious concern in terms of performance. For a well-behaved grid application, the processing time has to be much greater than the communication time.

4.10.2 Making application data available

Deploying the application data may be performed in several ways. If the application relies on centralized data-base servers, there must be a platform tool or even an application task for fetching the data at the server, partitioning it conveniently, and sending the pieces to the grid nodes. This automated process is usually the best option when the grid application is integrated with legacy systems that store their state on data bases, but other issues arise when deciding how to spread the application data across the grid.

- **Web publishing**

Probably the simplest way to make data available to grid applications is to publish it on the ordinary Web sites of FTP servers. There is a whole generation of systems and tools to aid developers to accomplish this task efficiently, but this philosophy has some major drawbacks. If publishing the data itself is easy, getting it to process may not be; the grid application programmer will have to deal with network programming to build its application, which is not desired; additionally, depending on how the application is designed, it can suffers from scalability, as every node may try to access the data at once. This happens because the responsibility for distributing the data across the grid relies on the application designer, and not on the grid platform.

- **Data-base server oriented**

It differs from the previous scenario in the sense that many legacy systems already have their data stored on data-base servers. The main drawbacks also apply to this case: The application developers will probably have to deal with database access issues when developing their applications.

- **Grid platform driven**

When the grid platform provides facilities for fetching the application data and distributing it across the platform automatically, the application development process can be drastically simplified. In this case, the platform must include tools for describing the data in its original source and specifying how it should be partitioned and distributed among the grid nodes. Scalability issues remain totally under control of the platform itself.

This is certainly the best option among all, but so far there isn’t any grid platform that provides full-fledged facilities for fetching and distributing application data across its nodes.

Chapter 5

Design and Development

The MDS information data is stored in an LDAP[45] directory. For this purpose, the Globus project uses the OpenLDAP server that is a free implementation of an LDAP directory. The Globus project modified the OpenLDAP libraries to integrate the GSI security in the connection established between a client and the OpenLDAP server. Consequently, my broker will:

- Use the credentials created with **grid-proxy-init** to securely access and retrieve information from the MDS server.
- Use the OpenLDAP C client API.
- Use an LDAP query to retrieve Grid resources information from the MDS.
- The LDAP query can be submitted to either a GIIS server or directly to a GRIS server.

However, the retrieved information will consider only the resources managed by the GIIS or the GRIS. Therefore, to have access to all resources managed by a grid, a request is sent to the GIIS server at the top of the LDAP tree because it represents the virtual organization. Globus GIIS usually listens on port 2135 and not the usual LDAP port.

5.1 Integrating with the LDAP protocol

The LDAP protocol provides a rich and powerful query protocol, even if it is not quite as rich as the SQL language. It can express a query such as "find all Linux nodes that have either two Pentium IV CPUs faster than 2GHz or four Pentium III CPUs faster than 1 GHz with at least 512 MB of memory and around 100 MB of scratch disk storage." The query works on the attributes defined in the Globus schema of the MDS. The sophisticated query operation includes:

i. Logic operators: AND (&), OR (|), and NOT (!). |
ii. Value operators: EQUAL TO (=), GREATER THAN OR EQUAL TO (>=), LESS THAN OR EQUAL TO (<=), and
iii. NOT EQUAL TO (!=) for approximate matching.

There are two commands provided by the Globus Toolkit to access the MDS (grid-info-search and ldapsearch). These commands show the result of an LDAP command without the need to recompile the application.
The string (or filter) that specifies the LDAP query itself is written as:
&(Mds-Os-name=Linux)(Mds-Cpu-model=Pentium*)(Mds-Cpu- SpeedMHz>=500).

5.2 Design issues

5.2.1 Globus libc APIs

The Globus Toolkit 2.4 is a cross-platform development framework and allows the application development of portable grid applications by using its API [42]. The globus-libc API provides a set of wrappers to several POSIX system calls. The grid developer must use these wrappers to ensure thread-safety and portability. The globus equivalents to the POSIX calls add the prefix globus_libc to the function while prototypes remain identical. For example, globus_libc_gethostname() should be used instead of gethostname(), or globus_libc_malloc() instead of malloc(). The globus-thread API provides system call wrappers for thread management. These include:

- thread life-cycle management
- mutex life-cycle, locking management
- condition variables, signal management

This API must be used to manage all asynchronous or non-blocking Globus calls and their associated callback functions. Usually a mutex and a condition variable are associated for each non-blocking Globus function and its callback function.

5.2.2 Makefile

globus-makefile-header is the tool provided by the Globus Toolkit 2.4 to generate platform- and installation-specific information. It has the same functionality as the well-known autoconf tools.

The input parameters to this tool are:

- The flavor required: gcc32, gcc32dbg for debugging purposes, gcc32pthr for multi-thread binary. The flavor encapsulates compile-time options for the modules built

- The list of modules that are used in your application and that need to be linked with your application are globus_io, globus_gss_assist, globus_ftp_client, globus_ftp_control, asyn_gram_job, globus_common, globus_gram_client, and globus_gass_server_ez.

- the --static flag can be used to get a proper list of dependencies when using static linking. Otherwise, the dependencies are printed in their shared library form.

The output will be a list of pairs (VARIABLE = value) that can be used in a Makefile as compiler and linker parameters.

These variables are built based on the local installation of the Globus Toolkit 2.4 and provide an easy way to know where the Globus header files or the Globus libraries are located.
Consequently, the procedure to compile a Globus application is the following:
Generate an output file (globus_header in the example) that will set up all the variables used later in the compile phase.

globus-makefile-header --flavor=gcc32 globus_io globus_gss_assist globus_ftp_client globus_ftp_control ASYN_GRAM_JOBglobus_common globus_gram_client globus_gass_server_ez globus_openldap > ***globus_header***

The application will be linked with Globus static libraries or Globus dynamic libraries, depending on the kind of Globus installation

Under Linux, if application uses dynamically linked Globus libraries, then be sure that:

1. Either the LD_LIBPATH_PATH is properly set to $GLOBUS_LOCATION/lib when application is run.
2. Or $GLOBUS_LOCATION/lib is present in /etc/ld.so.conf
3. Run the . $GLOBUS_LOCATION/etc/globus-user-env.sh command to set the any other Globus variables .

The list of main packages that are used in this implementation are:

globus_common used for all cross-platform C library wrappers
globus_openldap for querying the MDS server
globus_gass_server_ez to implement a simple GASS server
globus_gass_transfer for GASS transfer
globus_io for low-level I/O operation
globus_gss_* for GSI security management
globus_ftp_client, globus_ftp_control for gsiftp transfer
ASYN_GRAM_JOBfor job submission

5.2.3 Globus module

In Globus Toolkit V2.4, each Globus function belongs to an API provided by a specific Globus module. This module must be activated before any of the functions can be used. The globus_module API provides functions to activate and deactivate the modules:

- **globus_module_activate ()** calls the activation function for the specified module if that module is currently inactive.
- **globus_module_deactivate ()** calls the deactivation functions for the specified module if this is the last client using that module.

The function's return value is GLOBUS_SUCCESS if it is successful. GLOBUS_GRAM_CLIENT_MODULE is activated and deactivated in the broker code. That may be an issue if it is called from a program that thinks that this module is still active after its call.

5.2.4 Callbacks

The Globus toolkit provides two ways to perform an operation:

- Blocking calls that wait for the operation before returning.
- Non-blocking or asynchronous calls that immediately return and use a callback mechanism that lets the application know the operation has completed or its status has been last modified

Asynchronous calls have several advantages:

- They permit an application to execute a large number of Globus actions in a very short time because the functions return immediately.
- The application can remain passive and let the Globus framework manage these operations. For example, an application can submit hundreds of jobs in a few milliseconds and will be kept informed of job status by the Globus environment. It will not need to regularly poll job status.

When using Globus asynchronous calls, the application needs to provide an entry point that will allow the Globus framework to communicate with the application. This entry point is called a *callback*.

The Globus Toolkit provides portable and thread-safe functions across various operating systems to manipulate mutexes. These functions are provided by condition variables in the portability library **globus_common**. Each Globus call corresponds to a normal POSIX system call and is prefixed with globus_.

On the other hand, the asynchronous function usually takes at least two arguments:

- The first argument is a pointer to the callback function. The prototype of the callback varies according to the Globus function.
- The second argument is a pointer of any type that points to a structure that represents the data shared with the callback. This pointer is then passed back to the callback function by the Globus framework and permits the callback function to retrieve the data.

Figure 5.1 examine a simple, schematic example of an asynchronous Globus function called globus_operation().

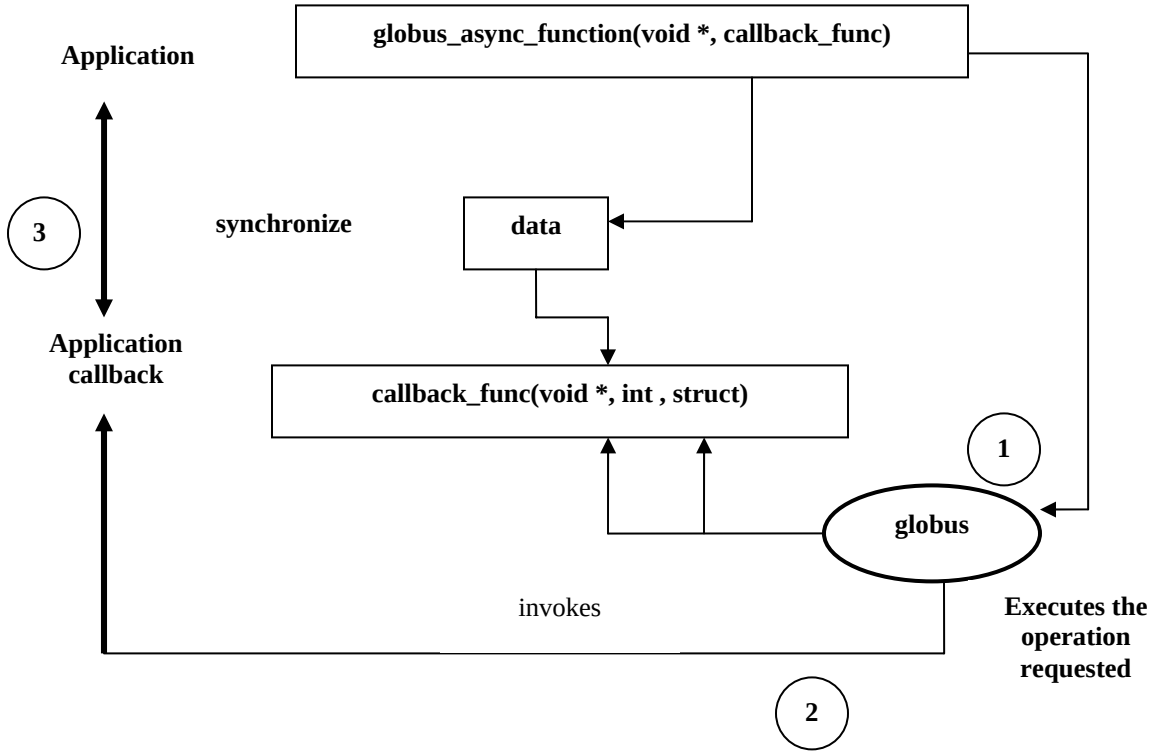

Figure 5.1

The prototype of such a Globus asynchronous function is:

globus_async_operation((globus_operation_function*) callback_func, (void *) data)

The corresponding callback is:

```
static void callback_func(
        void* user_callback_arg,
        int   state,
        int   errorcode)
{
    (DATA_TYPE) * data = (DATA_TYPE*) user_callback_arg;
    ....
    data->done= GLOBUS_TRUE;
}
```

State and error code are additional arguments received by the callback and provided by the Globus libraries during the callback invocation. These additional arguments are specific to this globus_operation() function. user_callback_arg is set to the value of data.

The argument data is set by the application during the globus_operation() call and not by the Globus Toolkit libraries.

The callback is not specific to a Globus function calls but to a Globus function. Consequently, it must be reentrant because it can be invoked by all the calls to the Globus function: it is declared as static and it will be used for all the calls to the same Globus function.

5.2.5 GRAM Server

The first step to is to create the GRAM server on the execution node that will monitor the status of the job and associate a callback with this job.

5.3 Development

A C++ object method pointer cannot be directly passed as a function pointer to the Globus asynchronous function because of the re entrance issue. The strategy is to use an intermediary C static function that will call an object method. The object will be specific to

the asynchronous function call but not to the callback itself. The void pointer shared between the callback and the asynchronous Globus function will be used to pass this object pointer. Then, not only will some data be exchanged between the application and callback but the callback can invoke the method of an object created in the application. Consequently, it actively influences the behavior of the application. This influence is managed by the mutex/condition variables mechanism previously described that avoid deadlocks and race conditions.

GRID_COM is a possible implementation of such a class. It embeds a done variable as an attribute. done is a boolean attribute of the class that describes the state of the operation: GLOBUS_TRUE when completed or GLOBUS_FALSE otherwise. It also provides the necessary methods that will be called both from the callback and from the application to perform the thread synchronization when they modify their shared data. GRID_COM represents the communication between the application and the callback function.

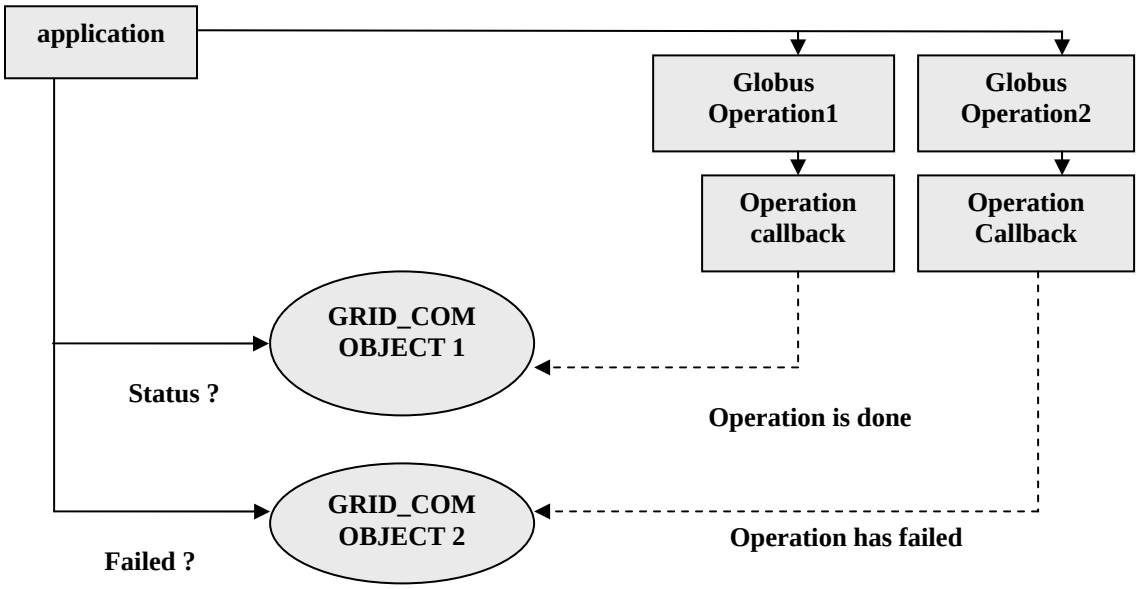

Figure 5.2

Figure 5.2 Callback mechanisms using the GRID_COM C++ class

The definition of the GRID_COM class is given in grid_com.h

```
class GRID_COM {
      globus_mutex_t mutex;
      globus_cond_t  cond;
      globus_bool_t  done;
    public:
      GRID_COM();
      ~GRID_COM();
      globus_bool_t  IsDone();
     virtual void   setDone();
     virtual void   Continue();
     virtual void   Wait();

};
```

GRID_COM methods that can be used by the application and the callback are:

- setDone() sets the status of the operation to done . The done attribute is actually set to false when the object is initialized. This method is normally called from the callback.
- Wait() waits for the completion of the operation. This method is usually called from the main program.
- isDone() permits retrieval of the state of the operation (done or not) and checks the value of the done attribute. This method is called in the Wait() method:
- Continue() resets the value of the attribute done to false. This method is called to invoke the same asynchronous function and use the same GRID_COM object to manage the synchronization between the callback and the application.

All the GRID_COM methods are declared virtual because its purpose is to be derived. Classes will implement specific processing for this method and will invoke GRID_COM methods. For example, ASYN_GRAM_JOB will use Continue() to release memory allocated by a previous submitted job.

When a program needs to be informed about data generated by a Globus asynchronous function, a GRID_COM object can be used jointly with a C static callback. The GRID_COM object, as well as the callback function pointer, are passed as argument to the Globus function. The GRID_COM object will be shared by the callback and the application. The method of the object is completely thread-safe and embeds the synchronization mechanism previously described.

- The Globus asynchronous function call in the application looks like this:

```
GRID_COM  GRID_COM_object_pointer;
globus_async_operation( (globus_operation_function*) callback_func, (void *)

    &GRID_COM_object_pointer )
```

- The corresponding callback looks like this

```
static void callback_func(
  void* user_callback_arg,
   int   state,
   int   errorcode)
{
 (GRID_COM) * Monitor = (GRID_COM*) user_callback_arg;
    ....
    Monitor->setDone();

}
```

To submit a job with the Globus Toolkit, an RSL string describing the job must be provided. The globus_gram_client API then provides an easy way to submit the job. This is illustrated in figure 5.3

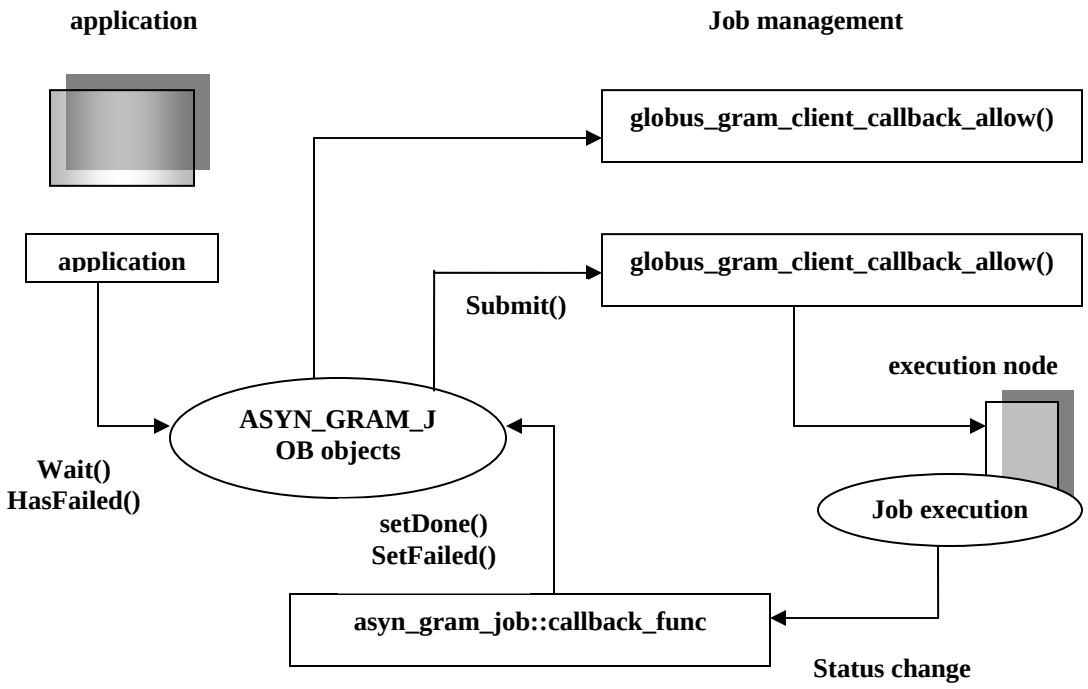

Figure 5.3: Submitting a job

The asynchronous job submission will be implemented via the ASYN_GRAM_JOBclass.ASYN_GRAM_JOBis derived from GRID_COM. One object is created for each submitted job:

```
class ASYN_GRAM_JOB: public GRID_COM {
      char*     job_contact;
       globus_mutex_t  job_contact_mutex;
       char*    callback_contact;
       bool      failed;   // used to check if a job has failed
   public:
     ASYN_GRAM_JOB();
     ~ASYN_GRAM_JOB();
     bool Submit(string,string);
     void Cancel();
     void SetJobContact(const char*);
     void Continue();
     void SetFailed();
     bool HasFailed();

};
```

The class attributes has boolean attributes, failed and done, used to manage the state of the job:

- failed is set to true (C++ boolean type) when the job failed on the execution host. Its value is set to false by default during the object instantiation.
- done (from GRID_COM) represents the state of the job submission: completed or not.

The following methods are used by the callback made by the Globus framework, to indicate job state change:

- SetFailed() is used to indicate that the job has failed. It sets the failed attribute to true.
- setDone() is used to indicate that the job is finished. It sets the done attribute to true.

The application needs to create only one object per job that it wants to submit; thereafter it can use the following object methods to submit and manage each job:

- Submit() is used to submit a job. It takes the hostname of the execution node as well as the RSL string describing the job to be submitted.
- Continue() is used to reset the object for a new submission. This method also releases all memory allocated for the previous job submission to store job_contact; now able to submit another job. If job_contact equals null, another thread deallocated the memory or the previous job failed to be submitted.
- HasFailed() is used to check the value of the attribute failed and to know if the job has failed or not.
- Cancel() is used to cancel the job.

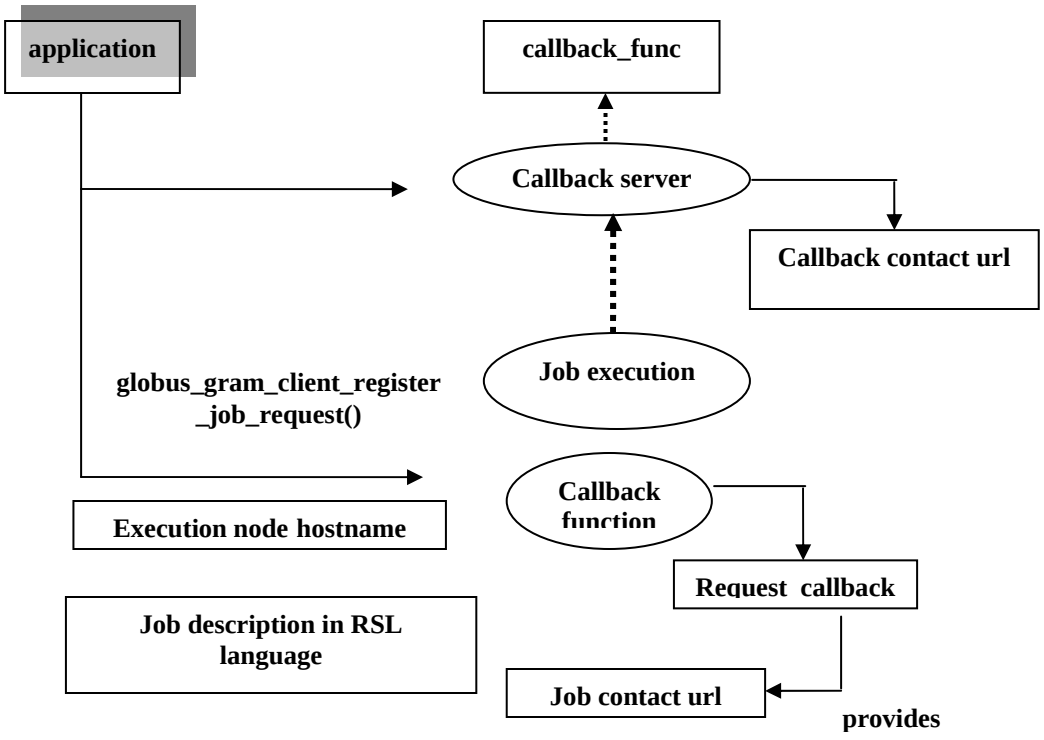

Figure 5.4: The Submit() method in detail

The first step is to start a callback server on the submission host that will listen to the job status changes. The second step is to start the job itself on the execution host. Two Globus calls then need to be invoked to perform the two actions.

In the ASYN_GRAM_JOB constructor, **globus_gram_client_callback_allow()** permits declaration of the callback associated with the job. This invokes the Globus framework for any job status change (for example, when it has finished). This Globus function also returns the url of the callback server that is used to receive jobs status change from GRAM servers. The contact information for this server is returned in a string allocated by the Globus framework (callback_contact). In the implementation, the ASYN_GRAM_JOB object itself (derived of GRID_COM) and the static function callback_func() defined in **asyn_gram_job.h** will be used to manage the asynchronous behavior of the function. We create one GRAM callback server per job on the submission node:

```
ASYN_GRAM_JOB::ASYN_GRAM_JOB() {
    cout << "Gram contact " << callback_contact << endl;
    globus_gram_client_callback_allow(
        ASYN_GRAM_JOB::callback_func,
        (void *) this,
        &callback_contact);

};
```

In the Submit() method, **globus_gram_client_register_job_request()** asks the Globus Toolkit to start the job. The execution hostname, as well as the RSL string, are provided to this function.

The callback associated with this function is made when the job is started on the execution host and returns the job contact URL that is used to remotely contact the job. It permits detection if an error occurred during the job submission, such as a wrong RSL string, for example. This callback is the function request_callback() given as parameter to **globus_gram_client_register_job_request()**.The RSL submission string is passed as argument, as well as the hostname of the execution node, to globus_gram_client_register_job_request(). We invoke the c_str() method to pass a C++ string type to a Globus function that expects for a char* parameter.

GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL specifies to the callback server that the application wants to monitor all states (done, failed, staging files).

callback_contact is the URL of the GRAM callback server that will manage all the jobs state modification and previously started in the GLOBUS

5.3.1 Implementation

5.3.1.1 Querying the MDS and establishing the connection

Using the OpenLDAP[44] API, a LDAP query is sent to the Globus MDS.. However, when using the Globus LDAP API, make sure to include the following two header files:

#include "lber.h"

#include "ldap.h"

To connect to the Globus MDS, the following informations are provided to the LDAP API by the application :

- A GIIS or a GRIS hostname.
- A port number where this GIIS or GRIS server is listening. The port numbers are different for a GIIS or for a GRIS.
- The distinguished name to bind the MDS LDAP directory .The three parameters are defined as constant

```
#define GRID_INFO_HOST "alpha.itso.grid.com"
#define GRID_INFO_PORT "2135"
#define GRID_INFO_BASEDN "mds-vo-name=alpha, o=grid"
```

The first step in sending an LDAP query to the Globus MDS is to connect to the LDAP server .With OpenLDAP library, the connection will use the Globus GSI secure authentication mechanism .Therefore , valid proxy credentials need to be generated before trying to connect to the MDS server. The connection is established in two steps after the successful calls to ldap_open() and ldap_simple_bind_s(). The _s suffix indicates the use of the secure mode, which means GSI with the Globus Toolkit.

```
LDAP * ldap_server;
/* list of attributes that we want included in the search result */
/* Open connection to LDAP server */
if ((ldap_server = ldap_open(server, port)) == GLOBUS_NULL)
{
ldap_perror(ldap_server, "ldap_open");
exit(1);
}
/* Bind to LDAP server */
if (ldap_simple_bind_s(ldap_server, "", "") != LDAP_SUCCESS)
```

```
{
ldap_perror(ldap_server, "ldap_simple_bind_s");
ldap_unbind(ldap_server);
exit(1);
}
```

5.3.1.2 LDAP query

When the connection is established, the LDAP query is issued in terms of desired characteristics.

string filter=(&(Mds-Os-name=Linux)(Mds-Host-hn=))*

Here string filter is used to filter out the hosts which are running Linux operating system.

5.3.1.3 Submitting the LDAP query

Query is submitted to LDAP server using the following

```
if (ldap_search_s(
ldap_server,
base_dn,
LDAP_SCOPE_SUBTREE,
const_cast<char*>(filter.c_str()), attrs, 0,
&reply) != LDAP_SUCCESS)
{
ldap_perror(ldap_server, "ldap_search");
ldap_unbind(ldap_server);
exit(1);
}
```

The result of the query is a set of entries that matches the query filter. Each entry is a set of pairs (one attribute and its value).

5.3.1.4 Retrieving results from the Globus MDS

An entry is referenced in the LDAP directory by a distinguished name (DN). The *ldap_first_entry()* and *ldap_next_entry()* permit scrolling the list of entries. The

ldap_first_attribute() and *ldap_next_attribute()* permit browsing the attribute list, and *ldap_get_values()* allows to get their value from MDS.

The result will display the distinguished names of the entries that match the query as well as the list of the attributes and their values for this entry. These attributes are defined in the Globus schema.

Mainfile: myBroker.cc

This is the file in which main function is written .Implementation of broker is via the GetLinuxNodes() function. **main()** function calls this function .This function takes an integer as the number of required nodes and returns a vector of strings containing the hosts names. GetLinuxNodes() function has been implemented using the designing part described in sections 5.2 and 5.3 of this chapter.

The broker checks the status of the nodes by using the Globus ping functions provided by the globus Gram module API. The algorithm takes into account the CPU speed , the number of processors, and the CPU workload ,etc, and returns the best available nodes.

Chapter 6

Conclusion and Future work

6.1 Conclusion

The Globus Toolkit provides a robust framework for building grid applications and consists of a set of services and an API to access these services. It is not an integrated and complete grid solution because services like job scheduling, brokering, and GUI administration tools are not included. However, the Globus Toolkit offers great flexibility that allows us to integrate a grid project into an existing IT infrastructure and into other grid technologies. In many respects, Globus is to grid applications as TCP/IP is to client-server applications. This dissertation shows how we can develop high-level service ,resource brokering , by using Globus MDS features and GRAM services to meet the needs of our grid applications.

The Globus toolkit is a set of tools useful for building a grid. Its strength is a good security model, with a provision for hierarchically collecting data about the grid, as well as the basic facilities for implementing a simple, yet world-spanning grid. Globus will grow over time through the work of many organizations that are extending its capabilities. A particular focus of the Globus effort is the development of a small grid based on toolkit providing essential services that can be used to implement a variety of higher-level programming models, tools, and applications. As explained earlier, Globus components have been deployed in large scale and used to implement a variety of applications. Grid Computing can now be seen as a field located at the meeting point of many highly technical and delicate fields such as Security, Distributed Systems and Programming.

6.2 Future work

The Globus Toolkit does not provide an allocation mechanism that can make advanced reservations, and it does not include an intelligent scheduler. Indeed, it relies on PBS, Condor, or LSF for these functions. This dissertation does not implement a queuing system and assumes that there is always a node available to run a job. Therefore, the broker application needs to know the number of available nodes and will avoid submitting more jobs than the number of available CPUs only because the job cannot be queued. Also, a running job cannot be stopped to run another job with a higher priority. These concerns have not been touched in this dissertation.

So the work of this dissertation could be enhanced to accommodate queuing as well as priority system .In real environment, the broker should take into account a variety of factors and information. Not all of the information has to come from MDS. For instance, some other factors that might affect the broker's choice of resources could be viz., service level agreements, time range of utilization, client location and many other factors.This introduces an area yet to be implemented in Globus toolkit.

The usage of the broker puts some demands on the user. Estimating the size of the job output may be sometimes be difficult .However, a user running the same application quickly learns how much output it generates and can in later submission include estimates of the job output size in the job descriptions. To fully utilize a broker, the user must know what benchmarks are relevant for the performance of the submitted application. Users developing their application usually have some sense of this.

References

[1] Andrew S. Tanenbaum , Maarten van Steen Distributed Systems: Principles and Paradigms, 3/e, 2004.

[2] Foster and C. Kesselman, editors. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco, California, 1999.

[3] C. Kesselman I. Foster and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal Supercomputer Applications*, 15(3), 2001.

[4] Ian Foster. What is the grid? A three point checklist. Grid Today, 1(6), July 2002.

[5] Rajkumar Buyya, Muthucumaru Maheswaran, et al. A taxonomy and survey of grid resource management systems for distributed computing. The Journal of Software Practice and Experience, v.32 n.2, pp.135–164, 2002

[6] The Globus Project: A Status Report. I. Foster, C. Kesselman. *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, pp. 4-18, 1998.

[7] C. Kesselman. I. Foster. Globus: A metacomputing infrastructure toolkit. In *International J. Supercomputer Applications*, page 115. 1997.

[8] http://www.globus.org/. The globus alliance.

[9] I. Foster, C'kesselman, and S.Trecke. A secure architecture for computational grids. In Proc. 5th ACM Conference on Computer and Communications Security Conference, pages 83-82, 1998.

[10] Security for Grid Services. V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, IEEE Press, to appear June 2003

[11] J. Hirsch. Introducing SSL and certificates using SSLeay. World Wide Web Journal, 2(3 Summer) , 1997.

[12] Douglas R. Stinson. Cryptography, theory and practice. CRC press, New York, 1995.

[13] X.509 Proxy Certificates for Dynamic Delegation. V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist. *3rd Annual PKI R&D Workshop*, 2004.

[14] Grid Resource Management. J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds). Kluwer Publishing, Fall 2003.

[15] Czajkowski K., et al, *Resource Management Architecture for Metacomputing Systems*, in The 4th workshop on Job Strategies for Parallel Processing. 1998. pp 62- 82

[16] The Globus resource specification language rsl v 1.0. Internet, May 2004. http://www-fp.globus.org/gram/rsl_specl/html.

[17] R. Butler Von Welch, D. Engbert, I. Foster, S. Tuecke, J. Volmer, and C. kesselman. A National Authentication Infrastructure. IEEE Computer Society Press, 2000.

[18] Inc. The Stencil Group. The evolution of UDDI. Internet, May 2004. http://www.stencilgroup.com/ideas_scope_200207uddiv3.pdf

[19] K. Czajkowski, S.Fitzgerald, I. Foster, and C.Kesselman. Grid information services for distributed resource sharing. In Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10). IEEE CS Press, Aug. 2001.

[20] OpenLDAP. http://www.openldap.org. Internet, May 2004

[21] Data Management and Transfer in High Performance Computational Grid Environments. B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. Parallel Computing Journal, Vol 28 (5), May 2002, pp. 749-771.

[22] The Globus alliance . Grid FTP universal data transfer for the grid.Internet, May 2004. http://www-fp.globus.org/datagrid/deliverable/C2WPdraft3,pdf

[23] Performance and Scalability of a Replica Location Service. A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, R. Schwartzkopf. *Proceedings of the International IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, June 2004.

[24] BTierney, W. Johnston, J. Lee, G. Hoo, and M. Thompson. End-to-end performance analysis of high speed distributed storage systems in wide area ATM networks.In *NASA/Goddard Conference on Mass Storage Systems and Technologies,* 1996, LBNL-39064.

[25] Schopf, J.: A General Architecture for Scheduling on the Grid. Submitted to special issue of JPDC on Grid Computing (2002).

[26] Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information services for Distributed Resource Sharing. 10th IEEE International Symposium on High-Performance Distributed Computing (2001)

[27] Wolski, R.: Dynamically Forecasting Network Performance Using the Network Weather Service. Cluster Computing (1998)

[28] Buyya, R., Abramson, D., Giddy, J.: An Economy Driven Resource Management Architecture for Global Computational Power Grids. International Conference on Parallel and Distributed Processing Techniques and Applications (2000)

[29] Abraham, A., Buyya, R., Nath, B.: Nature's Heuristics for Scheduling Jobs on Computational Grids. International Conference on Advanced Computing and Communications (2000)

[30] Raman, R., Livny, M., Solomon, M.: Matchmaking: Distributed resource management for high throughput computing. International Symposium on High Performance Distributed Computing (1998)

[31] Huedo, E., Montero, R.S., Llorente, I.M.: An Experimental Framework For Executing Applications in Dynamic Grid Environments. ICASE Technical Report (2002)

[32] Martino, V., Mililotti, M.: Scheduling in a Grid Computing Environment using Genetic Algorithms. International Parallel and Distributed Processing Symposium (2002)

[33] YarKhan, A., Dongarra, J.J.: Experiments with Scheduling Using Simulated Annealing in a Grid Environment. Workshop on Grid Computing, Baltimore (2002)

[34] Neary, M.O., Cappello, P.: Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing. Joint ACM Java Grande - ISCOPE Conference (2002).

[35] Abramson, D., Buyya, R., Giddy, J.: A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems Journal, Volume 18, Issue 8, Elsevier Science (2002) 1061-1074

[36] Sample, N., Keyani, P., Wiederhold, G.: Scheduling Under Uncertainty: Planning for the Ubiquitous Grid. International Conference on Coordination models and Languages (2002)

[37] Allen, G., Angulo, D., Foster, I., and others: The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. Journal of High-Performance Computing Applications, Volume 15, no. 4 (2001)

[38] Vadhiyar, S.S., Dongarra, J.J.: A Performance Oriented Migration Framework for The Grid.

[39] The Role of Planning in Grid Computing. J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, K. Vahi. *ICAPS 2003*, 2003.

[40] Red Hat Linux 9.0 specification www.redhat.com

[41] Globus Toolkit 2.4 Install www.globus.org/download/gt2.4/

[42] Globus Toolkit API Documentation • www.globus.org/developer/api-reference.html

[43] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In Proc. 6th IEEE Symp. on High Performance Distributed Computing, pages 365-375. IEEE Computer Society Press, 1997.

[44] T. Howes and M. Smith. The ldap application program interface. RFC 1823, 08/09 1995.

[45] Ahmar Abbas Grid Computing : A practical guide to technology & Application

[46] A book on Grid Computing by Jushy Joseph Craig Fellenstein.

[47] A book on Grid Computing : “ Making the Globus Infrastructure a reality “ by Fran Berman , Geoffrey C. Fox , Anthony. J. G. Hey.

Appendix A Commands

This appendix has descriptions of the commands available in the Globus Toolkit. The commands are grouped together by key areas of Globus Toolkit functionality.

Security Commands

Command	Description
grid-cert-request	Creates a new certificate request and private key.
grid-cert-info	Displays certificate information. Unless the optional -file argument is given, the default location of the file containing the certificate is assumed to be the $X509_USER_CERT. If X509_USER_CERT not set $HOME/ .globus/ usercert.pem
grid-cert-renew	Creates a new key and renewal request for a Globus certificate. The Globus Certificate Authority (CA) will notify a user when the user's certificate is about to expire. The notification message also contains a challenge (a text string) that grid-cert-renew will embed in the renewal request, for higher security.
grid-change-pass-phrase	Changes the "pass phrase" that protects your private key. If the -file argument is not given, the default location of the file containing the private key is assumed to be $X509_USER_KEY If X509_USER_KEY not set, $HOME/.globus/userkey.pem
grid-check-ca-policyfile	grid-check-ca-sig <signer> <subject> Obtains policy file (EACL) from the default location, parses it, builds the GAA internal policy structure and checks if the specified signer is allowed to sign certificates for specified subject.
grid-inquire-policyinfo	grid-inquire-policy-info <signer>. obtains policy file (EACL) from the default location, parses it, builds the GAA internal policy structure, finds ACL entry corresponding to the signer and returns a list of authorized rights and corresponding conditions, if any.
grid-proxy-init	Creates a proxy certificate that can be used for authentication without having to enter the protecting pass-phrase.

Command	Description
grid-proxy-info	Displays proxy certificate information. Common name of the subject (/C=US/O=Globus/ CN=user1) ; common name of the proxy generator; proxy information: (limited of full proxy); strength of proxy (number of bits used in the key); and remaining time. It can also work in another mode, verifying that it is a valid proxy, which means that the following requirements are met: the proxy exists, the proxy is not a limited proxy, the proxy has not expired and is active for another H hours, and the proxy is encrypted using at least B bits Unless the optional -file argument is given, the default location of the file containing the proxy certificate is assumed to be the $X509_USER_PROXY. If X509_USER_PROXY not set, tmp/x509up_uXXX, where XXX is the local UID#.
grid-proxy-destroy	Removes any user proxy certificates generated using grid-proxy-init. It can also be used as a general-purpose "safe remove" program, in that it opens the file and replaces the information contained therein with garbage before deleting it. If no arguments are given, the proxy file at the default location (or at the location pointed to by env .var X509_USER_PROXY, if set) will be destroyed.
grid-security-config	This script will ask some questions about site specific information. This information is used to correctly generate the Grid Security Infrastructure for your site.

Job Submission Commands

Command	Description
globusrun	Run a single executable on a remote site. The job startup is done using the GRAM or UROC Globus services. Also, the GASS service can be used to provide access to remote files and or redirecting standard output streams. In addition to starting jobs, `globusrun` can be used to list previously started jobs or do authentication tests to GRAM gatekeepers.
globus-setup-test	Verifies credentials setup. Also verifies whether the user has submission capabilities to a certain gatekeeper. If no arguments are given, all gatekeepers on the local host will be tested.
globus-job-cancel	globus-job-cancel Cancels a job previously started using globus-job-submit.
globus-job-run	Allows the user to run a job at one or several remote resources. It translates the program arguments to a RSL request and uses globusrun to submit the job.
globus-job-clean	Kills the job if it is still running and cleans the information concerning the job.
globus-job-get-output	For the job specified, gets the standard output or standard error resulting of the job execution.
globus-job-status	Display the status of the job. See also globus get-output to check the standard output or standard error of your job
globus-job-submit	For batch job submission (i.e., submitting a job to a queue via some local scheduling manager). Allows the user to submit a job to a remote resources, using the same 1command-line syntax as `globus-job-run` does. The program translates the program arguments into an RSL request and uses `globusrun` to submit the job in batch submission mode, that is, after the job is submitted, no connection exists between the local host and the remote host. `globus-job-submit` prints out a job id after successful submission of the job to a remote resource. Unless stdout/stderr are specified, the program output will be buffered at the remote site and can be retrieved at any time with `globus-job-get-output`. If the output is buffered, the user must make sure to run `globus-job-clean` when the program output is no longer needed.

Information Services Commands

Command	Description
grid-info-add	Modifies the GIS server based on the contents of input file. adds one or several objects to the MDS, according to the LDIF entries in the file pointed by the -file argument. If the -file argument is omitted, the program assumes this information is available on stdin. To be able to store information in the MDS, the password of the Directory Manager for the local organization must be provided. If the password argument is not given, the password will be asked for interactively by grid-info-add.
grid-info-create	Modifies the MDS server based on the contents of the input file.
grid-info-genfilter- template	To provide a bootstrapping of scripts that all follow the same convention. And to remove the burden from the developer by inserting the appropriate code to perform input validation and to guarantee output conformance.
grid-info-host	Creates a set of CLDIF entries for the host '${bindir}/globus-hostname' where options are: -usage (Displays this message); -version (Displays the current version number); -f[ile] hostfile (Creates only CLDIF entries for host listed in hostfile); -all (Creates CLDIF entries for all hosts at the site)…
grid-info-hostsearch	Queries a GRIS on a specified host and port. By default the local host GRIS and associated default GRIS port of 2135 are used.
grid-info-hostsearch	Queries a GRIS on a specified host and port. By default the local host GRIS and associated default GRIS port of 2135 are used.
grid-info-interfaces	For the current host, build the associated GlobusNetworkInterface and GlobusNetwork Interface Image templates.
grid-info-modify	Update an existing database entry. Metadata is inserted by grid-info-update.

Command	Description
grid-info-networks	For the current site, build the associated GlobusNetwork and GlobusNetworkImage templates. For the current site, build the associated network objects based on the built computational resource nodes for the site.
grid-info-prep	To convert CLDIF to LDIF with simple format and error checking. This script is directly tied to the format and structure of a "commented LDIF" template file.
grid-info-remove	See grid-info-add. deletes one or more objects to the MDS, according to the LDIF entries in the file pointed by the -file argument. If the -file argument is omitted, the program assumes this information is available on stdin. To be able to delete information from the MDS, the password of the Directory Manager for the local organization must be provided. If the -password argument is not given, the password will be asked for interactively by `grid-info-remove`.
grid-info-search	Searches the GIIS. Sends one or more queries to the GIIS and displays the result on stdout. The query QUERY is a RFC 1558 compliant LDAP search filter. By default, the object's name and all its attributes are displayed: this can be narrowed down by specifying what attributes to display after the query.
grid-info-site	Usage: $PROGRAM_NAME [hostname | -f[ile] hostfile]. Creates a set of CLDIF entries for the host \"hostname\, where options are: -usage (displays this message); -version (displays the current version number); -f[ile] hostfile (creates only CLDIF entries for host listed in hostfile).The hostfile contains a list (one per line) of host names and an optional location of the Globus bin directory. The default for this directory is $bindir, e.g., host-dns-name [bindir]

Commands	Descriptions
grid-info-update	See grid-info-add. modifies one or more objects to the MDS, according to the LDIF entries in the file pointed by the -file argument. If the -file argument is omitted, the program assumes this information is available on stdin. To be able to delete information from the MDS, the password of the Directory Manager for the local organization must be provided. If the -password argument is not given, the password will be asked for interactively by `grid-info-remove`.
grid-mapfile-addentry	can be used to add entries to the mapfile. For example the following command would add a user to the mapfile: gridmapfile- add-entry -dn "/C=US/O=Globus/O=State University/CN=Joe User" –ln juser. You must type in the distinguished name exactly as it appears in the certificate. Not doing so will result in an authentication failure
grid-mapfilecheck-consistency	checks the consistency of the Grid mapfile. Options:-help, -usage (displays help); -version (displays version); -mapfile FILE, -f FILE Path of gridmap to be used.
grid-mapfiledelete-entry	deletes an entry from the Grid mapfile. Options: -help, -usage (displays help); -version (displays version); -dn <DN> Distinguished Name (DN) to add; -ln <local name> Local Login Name (LN) to map DN to; -dryrun, -d (shows what would be done but will not delete the entry; -mapfile file, -f file (path of gridmap file to be used)

Appendix B(Source Code in C++)

++++++++++++++++++++++++++++++++++**grid_com.h**++++++++++++++++++++++++++++++
This file is my header file in which i included necessary standard headers files and declare the global variables and methods.
++

```
#ifndef GRID_COM_H
#define GRID_COM_H
#include <cstdio>
#include <iostream>
#include <globus_common.h>
class GRID_COM {
globus_mutex_t mutex;
globus_cond_t cond;
globus_bool_t done;
public:
GRID_COM() {
globus_mutex_init(&mutex, GLOBUS_NULL);
globus_cond_init(&cond, GLOBUS_NULL);
done = GLOBUS_FALSE ;
};
~GRID_COM() {
globus_mutex_destroy(&mutex);
globus_cond_destroy(&cond);
};
globus_bool_t IsDone();
virtual void setDone();
virtual void Continue();
virtual void Wait();
};

globus_bool_t GRID_COM::IsDone() { return done; };
void GRID_COM::setDone() {
globus_mutex_lock(&mutex);
done = GLOBUS_TRUE;
globus_cond_signal(&cond);
globus_mutex_unlock(&mutex);
}
void GRID_COM::Continue() {
globus_mutex_lock(&mutex);
done = GLOBUS_FALSE;
globus_mutex_unlock(&mutex);
```

```
}
void GRID_COM::Wait() {
globus_mutex_lock(&mutex);
while(!IsDone())
globus_cond_wait(&cond, &mutex);
globus_mutex_unlock(&mutex);
};
#endif
```

++++++++++++++++++++++++++++++++++++**asyn_gram_job.h**++++++++++++++++++++++++++++++++

This header file implements necessary call back asynchronous functions and creates a callback entry.

++

GRID_GRAM_JOB

```
#ifndef ASYN_GRAM_JOB_H
#define ASYN_GRAM_JOB_H
#include <cstdio>
#include <string>
#include "globus_gram_client.h"
#include "GRID_COM.h"
class ASYN_GRAM_JOB : public GRID_COM {
char* job_contact;
char* callback_contact;                    /* This is the identifier for asyn_gram_job_request */
bool failed;                                // used to check if a job has failed
public:
ASYN_GRAM_JOB();
~ASYN_GRAM_JOB();
bool Submit(string,string);
void Cancel();
void SetJobContact(const char*);
void Wait();
void Continue();
void SetFailed();
bool HasFailed();
};


namespace asyn_gram_job {

static void callback_func(void * user_callback_arg,
char * job_contact,
int state,
int errorcode)
{
ASYN_GRAM_JOB* Monitor = (ASYN_GRAM_JOB*) user_callback_arg;
switch(state)
{
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN:
cout << “Staging file in on: “ << job_contact << endl;
break;
```

```
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_OUT:
cout << “Staging file out on: “ << job_contact << endl;
break;
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
break; /* Reports state change to the user */
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE:
break; /* Reports state change to the user */
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
cerr << “Job Failed on: “ << job_contact << endl;
Monitor->SetFailed();
Monitor->setDone();
break;                                                    /* Reports state change to the user */
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
cout << “Job Finished on: “ << job_contact << endl;
Monitor->setDone();
break; /* Reports state change to the user */
}
}
static void request_callback(void * user_callback_arg,
globus_gram_protocol_error_t failure_code,
const char * job_contact,
globus_gram_protocol_job_state_t state,
globus_gram_protocol_error_t errorcode)
{
ASYN_GRAM_JOB* Request = (ASYN_GRAM_JOB*) user_callback_arg;
cout << “Contact on the server “ << job_contact << endl;
Request->SetRequestDone(job_contact);
}
}
ASYN_GRAM_JOB::ASYN_GRAM_JOB() {
};

ASYN_GRAM_JOB::~ASYN_GRAM_JOB() {
};
void ASYN_GRAM_JOB::SetRequestDone( const char* j) {
job_contact = const_cast<char*>(j);
request_cb.setDone();
}
void ASYN_GRAM_JOB::Submit(string res, string rsl) {
failed=false;
globus_gram_client_callback_allow(asyn_gram_job::callback_func,
(void *) this,
```

```
&callback_contact);
int rc = globus_gram_client_register_job_request(res.c_str(),
rsl.c_str(),
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
callback_contact,
GLOBUS_GRAM_CLIENT_NO_ATTR,
asyn_gram_job::request_callback,
(void*) this);
if (rc != 0) /* if there is an error */
{
printf("TEST: gram error: %d - %s\n",
rc,
globus_gram_client_error_string(rc));                    /* translate the error into english */
return;
}
};
void ASYN_GRAM_JOB::Wait() {
request_cb.Wait();
GRID_COM::Wait();
                                /* Free up the resources of the job_contact, as the job is over, and the contact is now useless.*/
globus_gram_client_job_contact_free(job_contact);
request_cb.Continue();
GRID_COM::Continue();
};
void ASYN_GRAM_JOB::Cancel() {
int rc;
printf("\tTEST: sending cancel to job manager...\n");
if ((rc = globus_gram_client_job_cancel(job_contact)) != 0)
{
printf("\tTEST: Failed to cancel job.\n");
printf("\tTEST: gram error: %d - %s\n",
rc,
globus_gram_client_error_string(rc));
}
else
{
printf("\tTEST: job cancel was successful.\n");
}
};
void ASYN_GRAM_JOB::SetFailed() {
failed=true;
}
```

```
bool ASYN_GRAM_JOB::HasFailed() {
return failed;
}
#endif
```

++**callback.h**++++++++++++++++++++++++++++++

This header file implements node reservations as well as asynchronous call back technique

++

```
#ifndef ASYN_GRAM_JOBS_CALLBACK_H
#define ASYN_GRAM_JOBS_CALLBACK_H
#include <cstdio>
#include <string>
#include <map>
#include "globus_gram_client.h"
#include "GRID_COM.h"
class ASYN_GRAM_JOBS_CALLBACK;
class ASYN_GRAM_JOB;
class ASYN_GRAM_JOBS_CALLBACK {
globus_mutex_t  JobsTableMutex;
char*  callback_contact;       /* This is the identifier for  the callback, returned by   asyn_gram_job_request */
map<string,ASYN_GRAM_JOB*> JobsTable;
void Lock();
void UnLock();
public:
ASYN_GRAM_JOBS_CALLBACK();
~ASYN_GRAM_JOBS_CALLBACK();

void Add(string,ASYN_GRAM_JOB*);
void Remove(char*);
char* GetURL();
ASYN_GRAM_JOB*  GetJob(char*);
};
class ASYN_GRAM_JOB : public GRID_COM {
char* jobcontact;
bool failed;
ASYN_GRAM_JOBS_CALLBACK* callback;
public:
ASYN_GRAM_JOB(ASYN_GRAM_JOBS_CALLBACK* f) :
failed(false),jobcontact(NULL),callback(f) {};
~ASYN_GRAM_JOB() {};
bool Submit(string,string);
void Cancel();
void SetJobContact(char*);
void Wait();
```

```
void SetFailed();
bool HasFailed();
};

namespace asyn_gram_job {
static void callback_func(void * user_callback_arg,
char * job_contact,
int state,
int errorcode)
{
ASYN_GRAM_JOBS_CALLBACK* Monitor = (ASYN_GRAM_JOBS_CALLBACK*)
user_callback_arg;
ASYN_GRAM_JOB* job = Monitor->GetJob(job_contact);
switch(state)
{
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN:
cout << "Staging file in on: " << job_contact << endl;
break;
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_OUT:
cout << "Staging file out on: " << job_contact << endl;
break;
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
break; /* Reports state change to the user */

case GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE:
break; /* Reports state change to the user */
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
Monitor->Remove(job_contact);
job->SetFailed();
job->setDone();
cerr << "Job Failed on: " << job_contact << endl;
break; /* Reports state change to the user */
case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
cout << "Job Finished on: " << job_contact << endl;
Monitor->Remove(job_contact);
job->setDone();
break;                                          /* Reports state change to the user */
}
}
static void request_callback(void * user_callback_arg,
globus_gram_protocol_error_t failure_code,
const char * job_contact,
```

```
globus_gram_protocol_job_state_t state,
globus_gram_protocol_error_t errorcode) {
ASYN_GRAM_JOB* Request = (ASYN_GRAM_JOB*) user_callback_arg;
cout << "Contact on the server " << job_contact << endl;
if (failure_code==0) {
Request->SetJobContact(const_cast<char*>(job_contact));
}
else {
cout << "Error during the code submission" << endl << "Error Code:" <<
failure_code << endl;
Request->setDone();
Request->SetFailed();
}
}
}
void ASYN_GRAM_JOB::SetJobContact(char* c) {
jobcontact=c;
callback->Add(c,this);
};
bool ASYN_GRAM_JOB::Submit(string res, string rsl) {
failed=false;
int rc = globus_gram_client_register_job_request(res.c_str(),
rsl.c_str(),
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
callback->GetURL(),
GLOBUS_GRAM_CLIENT_NO_ATTR,

asyn_gram_job::request_callback,
(void*) this);
if (rc != 0) /* if there is an error */
{
printf("TEST: gram error: %d - %s\n",
rc,
globus_gram_client_error_string(rc));                    /* translate the error into english */
return true;
}
else {
return false;
};
};
void ASYN_GRAM_JOB::Wait() {
GRID_COM::Wait();
```

```
                    /* Free up the resources of the job_contact, as the job is over, and  the contact is now useless.*/
if (jobcontact!=NULL) {
globus_gram_client_job_contact_free(jobcontact);
jobcontact=NULL;
};
Continue();
};
void ASYN_GRAM_JOB::Cancel() {
int rc;
printf("\tTEST: sending cancel to job manager...\n");
if ((rc = globus_gram_client_job_cancel(jobcontact)) != 0)
{
printf("\tTEST: Failed to cancel job.\n");
printf("\tTEST: gram error: %d - %s\n",
rc,
globus_gram_client_error_string(rc));
}
else
{
printf("\tTEST: job cancel was successful.\n");
}
};
void ASYN_GRAM_JOB::SetFailed() {
failed=true;
};
bool ASYN_GRAM_JOB::HasFailed() {
return failed;
};
ASYN_GRAM_JOBS_CALLBACK::ASYN_GRAM_JOBS_CALLBACK() {
globus_mutex_init(&JobsTableMutex,GLOBUS_NULL);
globus_gram_client_callback_allow(
asyn_gram_job::callback_func,
(void *) this,
&callback_contact);
cout << "Gram contact " << callback_contact << endl;
};
char* ASYN_GRAM_JOBS_CALLBACK::GetURL() {
return callback_contact;
};
ASYN_GRAM_JOB* ASYN_GRAM_JOBS_CALLBACK::GetJob(char* s) {
return JobsTable[s];
};
```

```
ASYN_GRAM_JOBS_CALLBACK::~ASYN_GRAM_JOBS_CALLBACK() {
cout << callback_contact << " destroyed" << endl;
globus_gram_client_callback_disallow(callback_contact);
globus_free(callback_contact);
globus_mutex_destroy(&JobsTableMutex);
};
void ASYN_GRAM_JOBS_CALLBACK::Add(string jobcontact,ASYN_GRAM_JOB* job) {
Lock();
JobsTable[jobcontact]=job;
UnLock();
};
void ASYN_GRAM_JOBS_CALLBACK::Remove(char* jobcontact){
Lock();
JobsTable.erase(jobcontact);
UnLock();
};
void ASYN_GRAM_JOBS_CALLBACK::Lock() { globus_mutex_lock(&JobsTableMutex); };
void ASYN_GRAM_JOBS_CALLBACK::UnLock() { globus_mutex_unlock(&JobsTableMutex);
};
```

++**myBroker.h**++++++++++++++++++++++++++++++

This header file implements necessary functions to connect to LDAP Server.

++

```
/* gris_search.c */
#ifndef  MY_BROKER_H
#define MY_BROKER_H
#define GRID_INFO_HOST "alpha.itso.grid.com"
#define GRID_INFO_PORT 2135
#define GRID_INFO_BASEDN "Mds-Vo-name=local, o=Grid"

#include<ldap_config.h>
#include "globus_common.h"
#include "globus_gram_client.h"
#include "globus_module.h"
/* LDAP stuff */
#include "lber.h"
#include "ldap.h"
#include<iostream>
#include<list>
#include<cstddef>
#include <cstdlib>
#include <cmath>
#include <string>
#include <vector>
#include <algorithm>
                                        /*  this is the GIIS server, but it could be the
                                            GRIS server if  only talking to a local machine
                                            remember the port numbers are different */

/*************************************************************************************************
 This is a implementation of a broker. It checks all available  Linux nodes and their CPU usage .It uses GetLinuxNodes(). The first
 parameter is a vector of strings that will contain the list of nodes available  in returns. The second parameter is the number of nodes requested.

*************************************************************************************************/
using namespace std;
namespace j_broker {

class Host {
```

```
string hostname;
long cpu;
public:
Host(string h,int c) : hostname(h), cpu(c) {};
~Host() { };
string getHostname() { return hostname; };
int getCpu() { return cpu; };
};
bool predica(Host* a, Host* b) {                    /* this function compares the processors‘ s speeds and return the best */
return (a->getCpu() > b->getCpu());
}
void GetLinuxNodes(vector<string>& res,int n)
{
LDAP * ldap_server;
LDAPMessage * reply;

LDAPMessage * entry;
char * attrs[1];
//char * server = GRID_INFO_HOST;
 char* server="alpha.itso.grid.com";
//int port = atoi(GRID_INFO_PORT);
int port=2135;
char * base_dn = "Mds-Vo-name=local, o=Grid";


                                                    /* list of attributes that we want included in the search result */
attrs[0] = GLOBUS_NULL;
globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
/* Open connection to LDAP server */
if ((ldap_server = ldap_open(server,port)) == GLOBUS_NULL)
{ char* lde1="error1";
ldap_perror(ldap_server, lde1);
exit(1);
}
                                                        /* Bind to LDAP server */


if (ldap_simple_bind_s(ldap_server," "," ") != LDAP_SUCCESS)
{ char* lde2="error21";
ldap_perror(ldap_server, lde2);
ldap_unbind(ldap_server);
exit(1);
```

```
}

                                                    /* do the search to find all the Linux available nodes*/
//string filter= "(objectClass=MdsComputer)(Mds-Os-name=Linux)";
string filter= "(&(Mds-Os-name=Linux)(Mds-Host-hn=*))";
if (ldap_search_s(ldap_server, base_dn,LDAP_SCOPE_SUBTREE,
const_cast<char*>(filter.c_str()), attrs, 0,&reply) != LDAP_SUCCESS)
{
ldap_perror(ldap_server, "ldap_server");
ldap_unbind(ldap_server);
exit(1);
}
vector<Host*> nodes;

                                                /* go through the entries returned by the LDAP server. for each
                                                   entry, we must search for the right attribute and then get the
                                                   value associated with it */
for (entry = ldap_first_entry(ldap_server, reply);
entry != GLOBUS_NULL;
entry = ldap_next_entry(ldap_server, entry) )
{
 cout << endl << ldap_get_dn( ldap_server, entry ) << endl;
BerElement * ber;
char** values;
char * attr;
char * answer = GLOBUS_NULL;
string  hostname;
int cpu;
int cpu_nb;
long speed;
char* Mds_Host_hn="Mds-Host-hn";
char* Mds_Cpu_Free_15minX100="Mds-Cpu-Free-15minX100";

for (attr = ldap_first_attribute(ldap_server,entry,&ber);
attr != NULL;
attr = ldap_next_attribute(ldap_server,entry,ber) )
{
values = ldap_get_values(ldap_server, entry, attr);
answer = strdup(values[0]);
ldap_value_free(values);
```

```
char* Mds_Host_hn="Mds-Host-hn";
if (strcmp(Mds_Host_hn,attr) == 0)
hostname=answer;
char* Mds_Cpu_Free_15minX100="Mds-Cpu-Free-15minX100";

if (strcmp(Mds_Cpu_Free_15minX100,attr)==0)
cpu=atoi(answer);
char* Mds_Cpu_Total_count="Mds-Cpu-Total-count";
if (strcmp(Mds_Cpu_Total_count,attr)==0)
cpu_nb=atoi(answer);
char* Mds_Cpu_speedMHz="Mds-Cpu-speedMHz";

if (strcmp(Mds_Cpu_speedMHz,attr)==0)
speed=atoi(answer);
cout<<attr<<":"<<answer<<endl;
//printf("%s  %s \n", attr, answer);
}
                                                    /* it check if we can really use this node*/
if (!globus_gram_client_ping(hostname.c_str()))
nodes.push_back(new Host(hostname,speed*cpu_nb*cpu/100));
};
sort(nodes.begin(),nodes.end(),predica);
vector<Host*>::iterator i;
for(i=nodes.begin();(n>0) && (i!=nodes.end());n--,i++){
res.push_back((*i)->getHostname());
 //cout<< (*i)->getHostname() <<" " << (*i)->getCpu() << endl;
delete *i;
}
for(;i!=nodes.end();++i)
delete *i;
ldap_unbind(ldap_server);
globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
}; /* get_ldap_attribute */
};
#endif

/* int main( int argc,char** argv)
{
        using namespace j_broker;
        vector<string> Y;
        GetLinuxNodes(Y,atoi(argv[1]));
        vector<string>::iterator i;
```

```
        for(i=Y.begin();i!=Y.end();++i)
        cout<< *i << endl;
        return 0;
} */
```

++**myBroker.cc**++++++++++++++++++++++++++++++++

This is the main file

++

```
#include <string>
#include <vector>
#include <myBroker.h>
#include "globus_gram_client.h"
#include “callback.h”
#include "asyn_gram_job.h"
using namespace j _broker;
int main(int argc, char ** argv)
{
vector<string> Nodes;
GetLinuxNodes(Nodes,3);
                                                        // Quickly check if we can run a job
globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
ASYN_GRAM_JOB job;
vector<string>::iterator i;
for(i=Nodes.begin();i!=Nodes.end();++i) {
cout << "Try to submit on " << *i << endl;
job.Submit(*i,"&(executable=/bin/hostname)");
job.Wait();
if (!job.HasFailed())
break;
};
globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
}
```