Grid Middlewares: Interface between Grid Resources and Applications

A thesis
submitted in partial fulfillment of the requirements
for the award of degree of

Master of Engineering
in
Software Engineering

Under the Supervision of
Dr. (Mrs.) Seema Bawa
Professor & Head
CSED, TIET, Patiala.

Submitted By
Meena Gupta
(Roll No 8043113)

Computer Science & Engineering Department
Thapar Institute of Engineering & Technology
(Deemed University), Patiala-147004

May 2006

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Grid Middlewares: A Interface Between Grid resources and Applications"** in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering at Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Dr. Seema Bawa.

I have not submitted the matter presented in the thesis for the award of any other degree of this or any other university.

(Meena Gupta)

This is to certify that the above statement made by the candidate is correct and true to best of my knowledge.

(Dr. Seema Bawa)
Professor and Head
Computer Science and Engineering Department
Thapar institute of engineering and Technology
PATIALA-147004

Countersigned by

(Dr. Seema Bawa)
Professor and Head
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology

(Dr. T. P. Singh)
Dean of Academic affairs
Thapar Institute of Engineering and Technology
PATIALA-147004

Acknowledgement

With a deep sense of gratitude, I wish to express my sincere thanks to my supervisor, Dr. Seema Bawa, Professor and Head, Computer Science and Engineering Department for her immense help in planning and executing the works in time. The confidence and dynamism with which Dr. Seema Bawa guided the work requires no elaboration.

I am also thankful to Mr. Maninder Singh, Assistant Professor, Computer Science and Engineering Department for his profound knowledge, providing timely assistance and encouragement, which went long way in successful completion of this thesis. His valuable suggestions as final words during the course of work are greatly acknowledged

I am also thankful to Mr. Rajesh Bhatia, P.G. Coordinator, Computer Science and Engineering Department for the motivation and inspiration that triggered me for this thesis work.

I would also like to acknowledge the valuable discussions with the TIET Grid group that helped me to clear the doubts and problems that I faced during the thesis work.

I would also like to thank all the staff members and my co-students who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of this thesis.

I am also thankful to the authors whose works I have consulted and quoted in this work. Last but not the least I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Meena Gupta

(8043113)

Abstract

The Grid is computing and data management infrastructure, which provides the electronic underpinning for a global society in business, government and research. To realize the global grid environment we need a standard architecture to cope up the heterogeneity and the interoperatability among the various sites present in grid infrastructure. The grid architecture is realized using the various components like fabric, user and core level middlewares, applications and the portals.

The middleware component is of extreme importance while building the Grid environment, as the majority of the Grid specific features are implemented using the Grid middleware. The grid middlewares act as glue between the grid resource components and the application components. They provide a high level of abstraction to the end users, hiding the underlying details. There exists a lot of implementation of these components in the market, each focusing on some different aspects.

This thesis discusses these middlewares in detail. It elaborates how middlewares are realizing the architecture to create the grid environment. This creates a lot of confusion for the users to select a particular middleware for the grid setup or to be the part of the grid.

In this thesis work, we have undertaken the problem to compare these middleware on the various factors so that it become easy for the end user to select the middleware to create a grid environment in which he can effectively and efficiently utilize the grid potential.

Table of Contents

List of Figures

Organization of Thesis

The first chapter briefly introduces Grid Computing concepts .It gives various topologies, working areas of Grid Computing and its benefits.

The second chapter of the thesis is devoted to the elements of the grid computing. It tells the various components of the grid computing and focuses on the middleware component importance

In the third chapter the problem has been formulated.

The fourth chapter provides overview of the middlewares in the market. It tells what functions are provided by the middleware and what are the basic features of the middlewares

The fifth Chapter concerns the setting up of grid using middlewares Globus, Nimrod-G, and the comparison between the various middlewares is being done on the basis of the various features as.

Finally, thesis has been concluded in the sixth chapter along with mentioning the scope for future.

Chapter 1: Introduction

Grids are becoming platforms for high-performance and distributed computing. A Grid benefits users by permitting them to access heterogeneous resources, such as machines, data, people and devices that are distributed geographically and organizationally. It benefits organizations by permitting them to offer unused resources on existing hardware and software. They allow users to execute compute intensive problems whose computational requirements cannot be satisfied by a single machine. Grid Computing has emerged as a new and important field and can be visualized as an enhanced form of Distributed Computing. With the advent of new technology, it has been realized that paralleling sequential applications could yield faster results and sometimes at a lower cost. Possessing multiprocessor systems was not possible for everyone. Thus, organizations started looking for a better and feasible alternative. It was found that though every organization possessed a large number of computers, which meant that they had huge processing power, but it remained underutilized. A new type of computing then came into existence and was known as Distributed Computing. In Distributed Computing, the problem to be solved is divided into numerous tasks that are then distributed to various computers for processing. The computers being used are connected through a Local Area Network (LAN). Also the problem needs to be divided into modules that can execute in parallel to each other. Parallelism can be either Data Parallelism or Functional Parallelism. Data Parallelism means that each computer performs the same function but for a different data set; whereas in Functional Parallelism, each computer performs a different function for different or same data set. As more and more resource intensive applications (compute-intensive and data-intensive) were developed, a need for larger amount of resource sharing was felt. Then the Grid computing comes into existence. A Grid enables the selection and sharing of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources, High Performance Computers (HPC), and other resources owned by different organizations for solving compute-intensive and data-intensive problems [2].

Grid computing is the next generation IT infrastructure that promises to transform the way organizations and individuals compute, communicate and

collaborate. It offers untapped processing cycles from networks of computers spanning vast geographical boundaries. Sharing in a Grid is not just a simple sharing of files but of hardware, software, data, and other resources [6]. Thus a complex yet secure sharing is at the heart of the Grid. It needs to be clearly defined as to what resources are going to be shared by the users, who are the users who are allowed to share these resources and under what conditions this sharing takes place by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation [2].

Grid focus on ensembles of distributed heterogeneous resources used as a platform for high performance computing. Grid services utilize the available computational re-sources so that tasks are run on whatever machine currently has available capacity. A Grid also allows a single large computation to be spread across several machines, each of which is executing some portion of the computation.

1.1 History of Grid

The term "**Grid**" was coined in the mid1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. The concept of *Computational Grid* has been inspired by the 'electric power Grid', in which a user could obtain electric power from any power station present on the electric Grid irrespective of its location, in an easy and reliable manner. When we require additional electricity we have to just plug into a power Grid to access additional electricity on demand, similarly for computational resources plug into a Computational Grid to access additional computing power on demand using most economical resource [2].

Technology	**Year**
Networked Operating Systems	1979-81
Distributed operating systems	1988-91
Heterogeneous computing	1993-94
Parallel distributed computing	1995-96
The Grid	1998

Table-1.1 Historical Background of the Grid [11]

The theory behind "Grid Computing." is not to buy more resources but to borrow the power of the computational resources you need from where it's not being

used". Many of the basic ideas behind the Grid have been around in one form or other throughout the history of computing. One of the "novel" ideas of the Grid is **sharing computing power.**

There is a certain amount of "reinventing the wheel" going on in developing the Grid. However, each time the wheel is reinvented; it is reinvented in a much more powerful form, because computer processors, memories and networks improve at an exponential rates [10]. Because of the huge improvements of the underlying hardware (typically more than a factor of 100x every decade), it is fair to say that reinvented wheels are qualitatively different solutions, not just small improvements on their predecessor.

1.2 What is Grid?

Rajkumar Buyya defined the Grid [18] as:

Grid is type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed resources dynamically at run time depending on their availability, capability, performance, cost, user quality-of –self-service requirement.

In other words we can say that Grid is,

- Grid is a service for sharing computer power and data storage capacity over the Internet.
- Grid is beyond simple communication between computer but it aims ultimately to turn the global network of computer into one vast computational
- Grid is to coordinate resources those are not subject to centralized control.
- Grid is to use standard, open, general-purpose protocols and interfaces.
- Grid is to deliver nontrivial qualities of service.

A computational Grid environment behaves like a virtual organization consisting of distributed resources. A **virtual organization** is a set of individuals and institutions defined by a definite set of sharing rules like what is shared, who is allowed to share, and the conditions under which the sharing takes place. A number of VOs exist like the application service providers, storage service providers, *etc.*, but they do not

completely satisfy the requirements of the Grid. The needs of the Grid range from client-server to peer-to-peer architecture, from single user to multi-user systems, from sharing files to sharing resources, *etc*. and all these in a dynamic, controlled, and secured manner. Current distributed computing technologies do not fulfill all these needs. As Grid computing focuses on dynamic and cross-organizational sharing, it enhances the existing distributed computing technologies. Many of the existing technologies like the World Wide Web (WWW), Internet and Peer-to-Peer Computing can be considered as similar to Grid Computing, but differences do exist. Internet and Peer-to-Peer computing have much in common with Grid technologies. **Grid** concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [2].

1.3 Types of Grids

Grid computing can be used in a variety of ways to address various kinds of application requirements. Often, the type of solutions categorizes Grids that they best address. Of course, there are no hard boundaries between these Grid types and often Grids may be a combination of two or more of these [11].

Grids can be classified on the basis of two factors:

- Scale
- Functionality

On basis of scale they can be further classified as:

- Global Grid
- Enterprise Grid
- Cluster Grid

Cluster Grid

Cluster Grid is simplest form of Grid and provides a compute service to the group or department level.

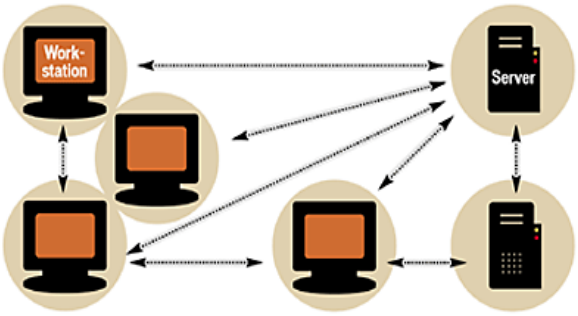

Fig 1.1 Cluster Grid [11]

Enterprise Grids

Enterprise Grids enable multiple project or department to share recourses with in enterprise or campus and not necessary have to address security and other global policy management issues associated with global Grid.

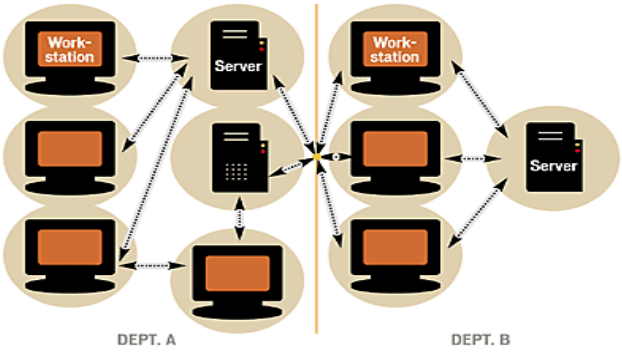

Fig1.2 Enterprise Grid [11]

Global Grids

Global Grids are collection of enterprise and cluster Grid as well as other geographically distributed resources, all of which are agreed upon global usage policies and protocols to enable resources sharing [2].

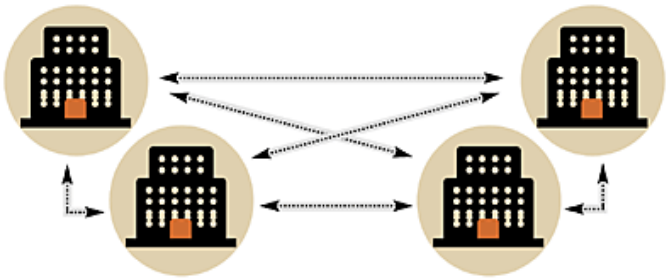

Fig1.3 Global Grids [11]

On basis of Functionality

- Compute Grids
- Data Grids

Compute Grids

A compute Grid is essentially a collection of distributed computing resources, within or across locations that are aggregated to act as a unified processing resource or virtual supercomputer.

Data Grids

A data Grid provides wide area, secure access to current data. Data Grids enable users and applications to manage and efficiently use database information from distributed locations.

1.4 Design Features

An ideal Grid environment will provide access to the available resources in a seamless manner such that physical discontinuities, such as the differences between platforms, network protocols, and administrative boundaries become completely transparent. In essence, the Grid middleware turns a radically heterogeneous environment into a virtual homogeneous one [17].

The following are the main design features required by a Grid environment.

- *Administrative hierarchy*. An administrative hierarchy is the way that each Grid environment divides itself up to cope with a potentially global extent. The administrative hierarchy determines how administrative information flows through the Grid.

- *Communication services*. The communication needs of applications using a Grid environment are diverse, ranging from reliable point-to-point to unreliable multicast communications [17]. The communications infrastructure needs to support protocols that are used for bulk-data transport, streaming data, group communications, and those used by distributed objects. The network services used also provide the Grid with important QoS parameters such as latency, bandwidth, reliability, fault-tolerance, and jitter control.

- *Information services*. A Grid is a dynamic environment where the location and types of services available are constantly changing. A major goal is to make all resources accessible to any process in the system, without regard to the relative location of the resource user [17]. It is necessary to provide mechanisms to enable a rich environment in which information is readily obtained by requesting services. The Grid information (registration and directory) services components provide the mechanisms for registering and obtaining information about the Grid structure, resources, services, and status.

- *Naming services*. In a Grid, like in any distributed system, names are used to refer to a wide variety of objects such as computers, services, or data objects. The naming service provides a uniform name space across the complete Grid

environment. Typical naming services are provided by the international X.500 naming scheme or DNS, the Internet's scheme.

- *Distributed file systems and caching*. Distributed applications, more often than not, require access to files distributed among many servers. A distributed file system is therefore a key component in a distributed system. From an applications point of view it is important that a distributed file system can provide a uniform global namespace, support a range of file I/O protocols, require little or no program modification, and provide means that enable performance optimizations to be implemented, such as the usage of caches [17].

- *Security and authorization*. Any distributed system involves all four aspects of security: confidentiality, integrity, authentication, and accountability. Security within a Grid environment is a complex issue requiring diverse resources autonomously administered to interact in a manner that does not impact the usability of the resources or introduces security holes/lapses in individual systems or the environments as a whole. A security infrastructure is the key to the success or failure of a Grid environment.

- *System status and fault tolerance*. To provide a reliable and robust environment it is important that a means of monitoring resources and applications is provided. To accomplish this task, tools that monitor resources and application need to be deployed [17].

1.5 Working Areas of Grid Computing

Grid computing is the flavor of the day and is being used extensibly. Very soon it is going to change the face of everyday computing. At present, following are the main areas of the Grid in which work is being carried out:

a) **Grid Information Services**- Obtaining high performance execution and results requires a careful selection of computers, networks, and other resources used by the application. In addition to these the protocols and algorithms being used by the

application should also be appropriate. Optimal selection in turn requires that the users have access to accurate and up-to-date information about the structure and state of the resources. The area of Grid Information Services deals with all and tries to satisfy these requirements. [2]

b) **Security**- Security is at the heart of Grid computing. Since the communication being done is across multiple administrative domains (each domain having its own local security), security is an important factor that needs special attention. A need exists to establish secure relationships between the large number of users and resources. The work in this area focuses on developing the security algorithms and also new mechanisms for fine-grained access control in a Grid.

c) **Resource Management/ Scheduling**- The work in this area focuses on providing uniform mechanisms for naming and locating resources on remote systems, and for utilizing these resources for distributed computations. It also takes care of how various resources are scheduled, that is how the available resources have been allocated to the various applications running on the Grid.

d) **Data Access and Management**- Access to distributed data is as crucial as access to distributed computational resources. Distributed applications often require access to large amounts of data and that too located at widely separated locations. The work in this area attempts to identify, prototype, and evaluate the technologies required to support the requirement of accessing widely distributed data for collaborative applications.

e) **Application Development and Programming Environments**- the work in this area focuses on devising methods to integrate Grid technologies or services into the existing commodity technologies and frameworks. This would ease the development of applications that use Grid services, as the Grid services would be integrated into existing application development environments and languages such as JAVA, CORBA, Python, and Perl [2].

1.6 Grid Application Areas

Applications with heavy use of computing resources (e.g., simulations, number crunching, the so-called grand challenge applications), applications using large information resources (e.g., multimedia databases), applications using special sub-applications (e.g., visualization), and applications using special devices (e.g., expensive scanners, laboratory equipment) are candidates for Grids. Especially we mention the following important application areas [23]:

- *Medical Applications:* In diagnostics huge amounts of data are generated at one place by specialized devices. These data have to be transported to the specialists, possibly located at several locations, while the patient might be at a third location. The task of a Grid in this scenario is to prepare and transport the medical data, so that they are available at the right location at the right time. Challenges within this group of applications include security (integrity of data, and making sure that the patient's data do not fall into unwanted hands), synchronization (deliver the right data at the right time to the right destination), etc.

- *Support for multinational enterprises*: Multinational enterprises work at several locations in several time zones. Data, e.g., multimedia data from inspections, must be pre-processed and forwarded to specialists who can take decisions.

- *Multimedia Applications:* Several Multimedia Applications make use of a Grid for processing media streams [23]. Within multimedia QoS control is very important. Applications often include the handling of Digital Rights Management. E.g., multimedia data can be watermarked, scrambled etc. Other typical Grid applications include indexing and data mining of media streams, Multiplayer Games, (scientific) visualization and computer graphics.

- *Applications from bio-informatics*, seismology, meteorology, etc. are data- and computing-intensive, and need often other information resources [23].

1.7 Advantages of Grid Computing

Some of the advantages of Grid Computing are listed below [20][28]:

- Seamless and secure access to large number of geographically distributed resources.
- Reduction in average job response time may occur but an overhead of limited network bandwidth and latency exists.
- Provides users around the world with dynamic and adaptive access to unparalleled levels of computing.
- With the infrastructure provided by the Grid, scientists are able to perform complex tasks, integrate their work and collaborate remotely.
- Grids can lead to savings in processing time.
- Efficient, effective, and economic utilization of available resources.
- Increased availability and reliability of resources.
- Shared access (by multiple users) to large amounts of data.
- Improved methods for collaborative work.
- Unprecedented Price-to-Performance ratio.

In this chapter we give the basic introduction to the grid computing. Here we discuss what the grid computing is, how it evolves and matures. What are the domain areas in which grid can be used, the various topologies of grids, the design features specific to the grid environment and the various benefits it offers to the users.

Chapter 2: Elements of Grid Computing

2.1 Grid Architecture

The Grid architecture is defined in a way that focuses on effective VO operations, which requires the establishment of sharing relationships among the participants of the Grid [8]. For this, the main issue is Interoperability among participants of the Grid. Interoperability can be achieved by setting up standard protocols. To make efficient and effective use of these protocols, standard Application Programming Interfaces (APIs) need to be developed. Such standards based architecture facilitates extensibility, interoperability, and portability. Grid architecture can be considered a layered architecture made up of layers of different widths arranged in an hourglass shape. The Grid architecture is made up of following five layers as shown in Figure 2.1 [2]:

- Fabric Layer
- Connecting Layer
- Resource Layer
- Collective Layer
- Application Layer

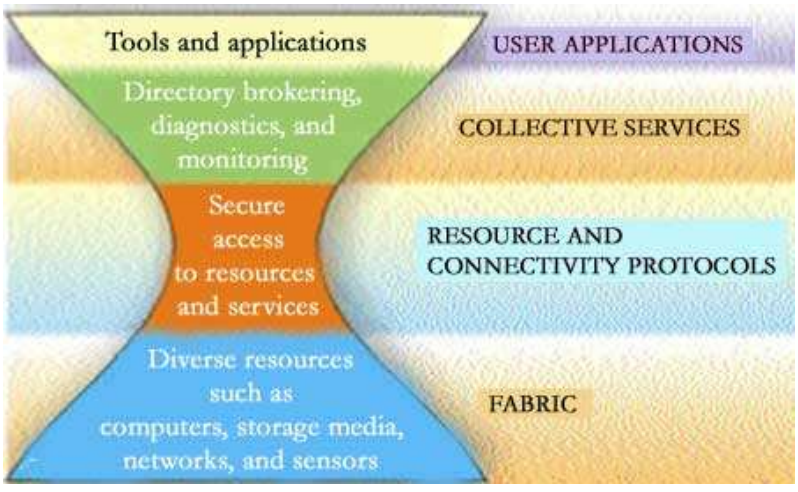

Fig2.1 Grid Architecture [14]

2.1.1 Fabric Layer

The fabric layer defines the basic resources and elements from which a Data Grid is fabricated. A typical computer fabric consists of clusters of computing nodes, where the user jobs are run, and a number of infrastructure elements may include Information servers, Master nodes which co-ordinate computing node clusters for batch and interactive services, Storage servers such as disk servers (e.g. NFS, HTTP, GridFTP) and tape infrastructure (tape servers and robotics) and Network infrastructure, Miscellaneous servers (time servers, password and credential servers) [14].

2.1.2 Connectivity Layer: Manages Communication

The communication layer defines the core communication and authentication protocols required for Grid specific networking services transactions. Communication protocols, which include aspect of networking transport, routing, and naming, assist in the exchange of between different fabric layers of different resources. The authentication protocols built on the top [14] of the network communication services in order to provide the secure authentication and data exchange between users and respective resources.

The communication protocol can work with any of the networking layer protocols that provide the routing, naming, and transport capabilities in networking services solutions.

The most commonly used network layer protocol is TCP/IP Internet protocol stack; however discussion and concept is not limited to that protocol. The authentication solutions require more complex characteristics. The following describes the characteristics for the consideration:

- **Single sign on**- This provides any multiple entities in the Grid to be authenticated once; the user can access any available resources in the Grid fabric layer without any further user authentication intervention

- **Delegation**- This provides the ability to access a resource under the current user set permission set; the resource should be to relay the same user credentials to other resources respective to the chain of access.

- **Integration with local resource specific security solutions**- each resource and hosting has specific security requirements and security solutions that match the local environment. This may include Kerberos security methods, window security methods, Linux security methods, and UNIX security methods [14]. Therefore in order to provide proper security in the Grid fabric model, all Grid solutions must provide integration with the local environment and respectively resources engaged by the security solution mechanisms.

- **User-based trust relationships**- In Grids, establishing a trust relationship between the users and multiple service providers is very critical.

- **Data security**- the data security is important in order to provide the data integrity and confidentiality. The data passing through the Grid computing solution, no matter what complications may exist, should made secure through the various cryptographic algorithms and data encryption mechanisms

2.1.3 Resource Layer

It defines protocols for the secure initiation, monitoring, and control of sharing operations on individual resources. The resource layer protocols call fabric layer function to access and control local resources. The resource layer protocols are concerned with individual resources and ignore issues of global state and atomic actions across distributed actions. There are two primary classes of resource layer protocols [14]. These protocols are key to the operation and integrity of individual resources. These protocols are as follows:

- **Information protocols-** obtains information about the structure and the operation state of resources including configuration usage policies, service –level agreements, and the state of resource. In most situations this information is used to monitor the resource capabilities and availability constraints.

- **Management protocols-** the important functionalities provided by management protocols are:
    - Negotiate access to a shared resource is paramount. The negotiation may include the requirement on the quality of service, scheduling, and other key operational factors.
    - Performing operations on the resources
    - Providing accounting and payment management functions on resource sharing is mandatory.
    - Monitoring the status of the operation, controlling the operation including terminating the operation, and providing asynchronous notifications on operation status.

It is recommended that the resource level protocols should be minimal from a functional overhead

2.1.4 Collective Layer

Collective protocols and services (and APIs) build on resource-layer protocols to enables interaction with multiple resources [9]. Collective layer is responsible for Coordinates multiple resources. It Contains protocol and services concerned with the global state and captures interactions across collections of resources [14].
They implement a wide variety of sharing behaviors without placing new requirements on the resources:

- **Discovery services-** This enables the virtual organizations participants to discover the existence and properties of those specific available VO's constraints.

- **Co-allocation-** These services allow the participants of the virtual organizations to request the allocation of one or more resources for a specific task, during specific period of time, and to schedule those tasks on appropriate resources.

- **Monitoring and diagnosis**- these services afford the virtual organizations resource failure recovery capabilities, monitoring of the networking and device services, and diagnostic services that include logging and intrusion detection.

- **Data replication services**- These services support the management aspects of the virtual organization's storage resources in order to maximize data access performance with respect to the response time, reliability, and costs.

- **Software discovery**- this provides the mechanisms to discover the best software implementations available in the Grid environment, and those available to the platform based on the problem being solved.

- **Workload management**- this provides multistep, asynchronous, multicomponent workflow management. This is a fundamental concern for enabling optimal performance and functional integrity.

- **Community authorization servers**- these servers control the resource access by enforcing community utilization policies and providing these respective access capabilities by acting as policy enforcement agents.

- **Community accounting and payment mechanisms**- these services provide resource utilization metrics, while at the same time generating payment requirements for members of any community.

As we can observe the functionality of collective layer capabilities and efficiencies are built upon the above layers

2.1.5 Application Layer

The final layer in Grid architecture comprises the user applications that operate within a Data Grid environment. Applications are constructed in terms of, and by calling upon, services defined at any layer. At each layer, there are certain protocols that provide access to some useful service: resource management, data access, resource discovery, and so forth. At each layer, APIs may also be defined whose

implementation exchange protocol messages with the appropriate service(s) to perform desired actions [14].

2.2 Grid Components

According to the above discussed Grid architecture the various components s that are necessary to form a Grid are as follows.

- *Grid fabric*. This consists of all the globally distributed resources that are accessible from anywhere on the Internet. These resources could be computers (such as PCs or Symmetric Multi-Processors) running a variety of operating systems (such as UNIX or Windows), storage devices, databases, and special scientific instruments such as a radio telescope or particular heat sensor [17].

- *Core Grid middleware:* This offers core services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security, and aspects of Quality of Service (QoS) such as resource reservation and trading. These services abstract the complexity and heterogeneity of the fabric level by providing a consistent method for accessing distributed resources.

- *User-level Grid middleware:* User level middleware utilizes the interfaces provided by the low-level middleware to provide higher-level abstractions and services. This includes application development environments, programming tools, and resource brokers for managing resources and scheduling application tasks for execution on global resources.

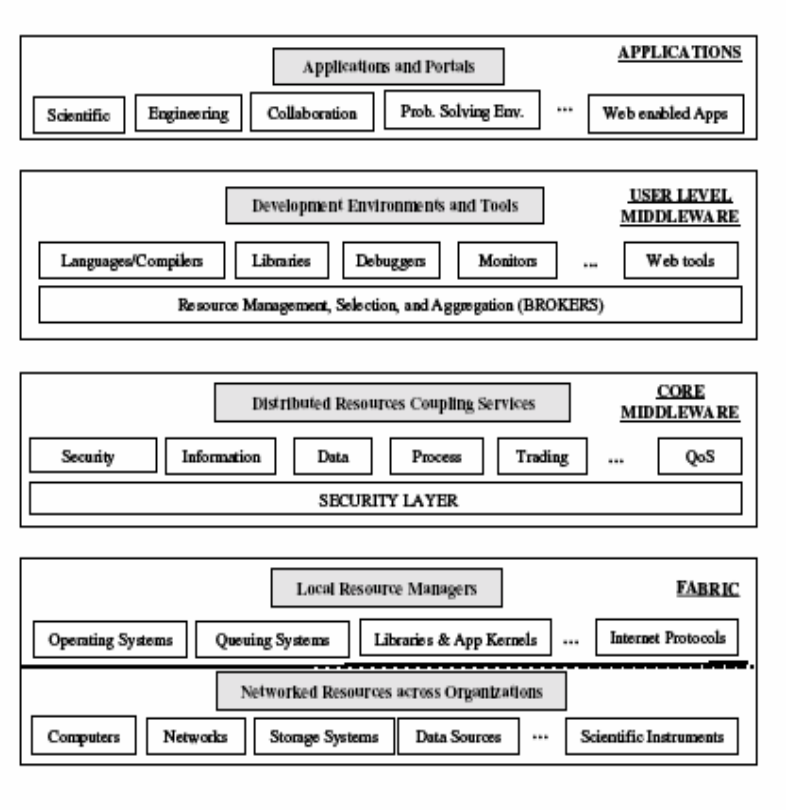

Fig 2.2 Grid components [17]

- *Grid applications and portal:* Grid applications are typically developed using Grid-enabled languages and utilities such as HPC++ or MPI. An example application, such as parameter simulation or a grand-challenge problem, would require computational power, access to remote data sets, and may need to interact with scientific instruments. Grid portals offer Web-enabled application services, where users can submit and collect results for their jobs on remote resources through the Web [17].

2.3 Relationship between Grid components and Grid architecture

The Grid Architecture is achieved using the Grid components. The beneath diagram tries to illustrate how they relate with each other:

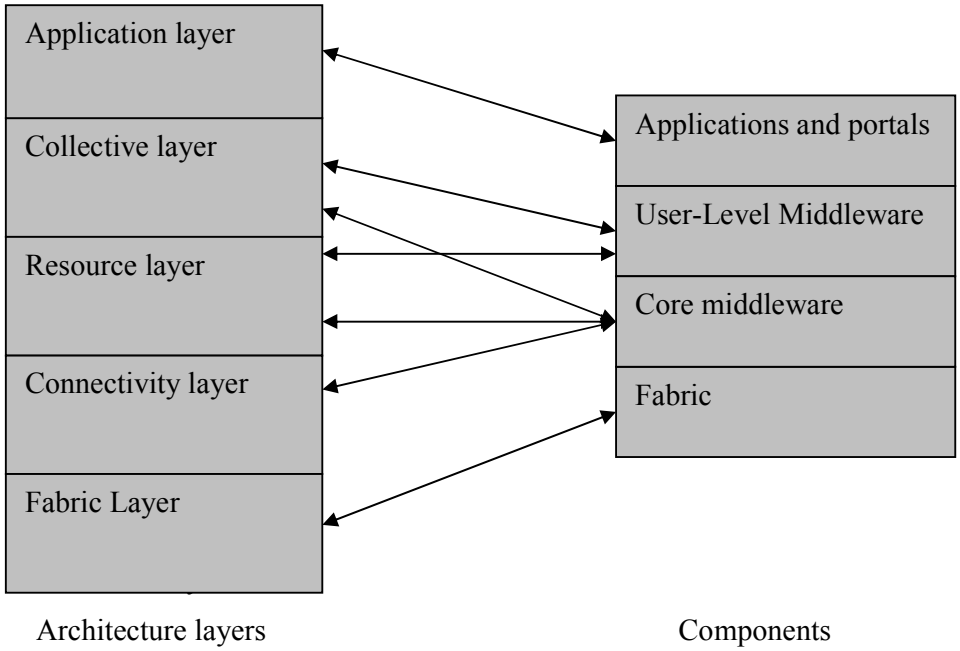

Fig 2.3 Relationship between Grid architecture and components

As depicted in the diagram the core level middleware component deals with the three layers of the architecture and the user level middleware deals with the collective layer and resource layer functions.

The middleware component is of extreme importance while building the Grid environment, as the majority of the Grid specific features are implemented using the Grid middleware.

This chapter discusses about the Grid architecture in depth and the Grid components. In it, we further discuss how the Grid architecture is achieved using the Grid components. It focuses on the importance of the Grid middleware component. In next chapter we well further explore the Grid middleware component.

Chapter 3: Problem Statement

3.1. Problem statement

Grid enables the sharing, selection and aggregation of geographically distributed resourced to perform a task. The grid middlewares act as an interface between the grid applications and resources. The grid middlewares hide the underneath complexities and provide a high level abstraction of the grid. Till today the various implementations of the grid middlewares exist in the market. Each middleware implements the architecture focusing on different aspects. Thus it creates a lot of confusion for the users to select a middleware for creating the grid environment.

We have undertaken the problem to do an in depth study of existing middlewares including Globus, Alchemi, nimrod/G, Condor, get a feel of these selected middlewares by installing them and to compare these middleware on the various factors so that it become easy for the end user to select the middleware to create a grid environment in which he can effectively and efficiently utilize the grid potential. The factors, which we have am considered for the middlewares comparison, are:

- Category
- Security
- Architecture
- Scalability
- Programming environment
- Platform
- Ease of Use
- Scheduling Policy
- Application Type

Chapter 4: Grid Middlewares: A Review

Grids have *middleware stacks*, which are a series of cooperating programs, protocols and agents designed to help users access the resources of a Grid[30]. Grid Middleware refers to the security, resource management, data access, instrumentation, policy, accounting, and other services required for applications, users, and resource providers to operate effectively in a Grid environment. Middleware acts as a sort of 'glue' which binds these services together.Middleware connect applications with resources.

Formally Grid middleware can be define [23] as:

"***A mediator layer that provide a consistent and homogeneous access to resources managed locally with different syntax and access methods***"

Till today several implementation of Grid middleware have been achieved. These implementations have well identified basic services. These middleware implementations are now moving their focus from proprietary/adhoc solutions to standard based solutions. The brief overview of the some popular middleware's Globus, Alchemi, and Condor is discussed in this section.

4.1 Globus

The Globus [29] toolkit is designed to enable people to create computational Grids. It has been developed over several years chiefly at the Argonne National Laboratory Illinois USA. Globus is an open source initiative aimed at creating new Grids capable of the scale of computing seen only in supercomputers up to now. As an open source project any person can download the software, examine it, install it and hopefully improve it. By this constant stream of comments and improvements, new versions of the software can be developed with increased functionality and reliability. In this way the Globus project itself will be on going with constant evolution of the toolkit [22][29].

The Globus architecture has provided the basic building blocks to do most of the things you would wish to when building a Grid. Many companies are working on products to operate commercial Grid solutions based on the Globus architecture. Therefore though the Globus initiative is open source, the solutions used in your department or office in the next ten years will not be the basic Globus toolkit, but a properly supported, debugged (to a relative extent) version installed by people trained to do so. At its roots however will be the same basic architecture that will be shown over the next few sections. [22]

4.1.1 Globus Pyramids

Globus Toolkit has three pyramids of support built on top of a security infrastructure, as illustrated. They are:

- Resource management
- Data management
- Information services

All of these pyramids are built on top of the underlying Grid Security Infrastructure (GSI). This provides security functions, including single/mutual authentication, confidential communication, authorization, and delegation.

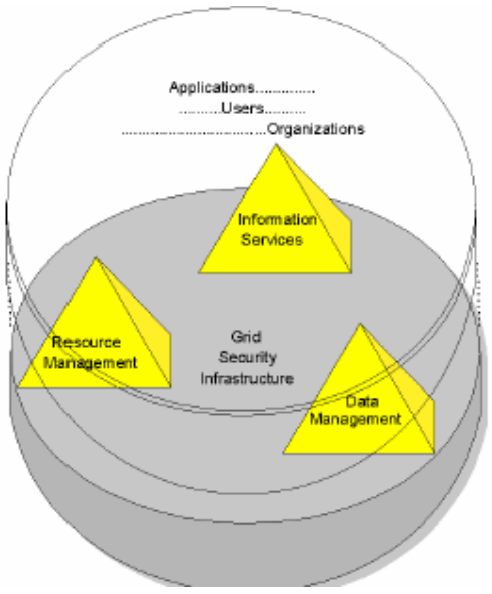

Fig4.1 Globus pyramids [24]

Resource management

The resource management pyramid provides support for:

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

Information services

The information services pyramid provides support for collecting information in the Grid and for querying this information, based on the Lightweight Directory Access Protocol (LDAP).

Data management

The data management pyramid provides support to transfer files among machines in the Grid and for the management of these transfers. [24]

4.1.2 Components of Globus Toolkit

For each pyramid previously presented, Globus provides a component to implement resource management, data management, and information services. The various Components are:

- GRAM
- MDS
- GridFTP
- GSI

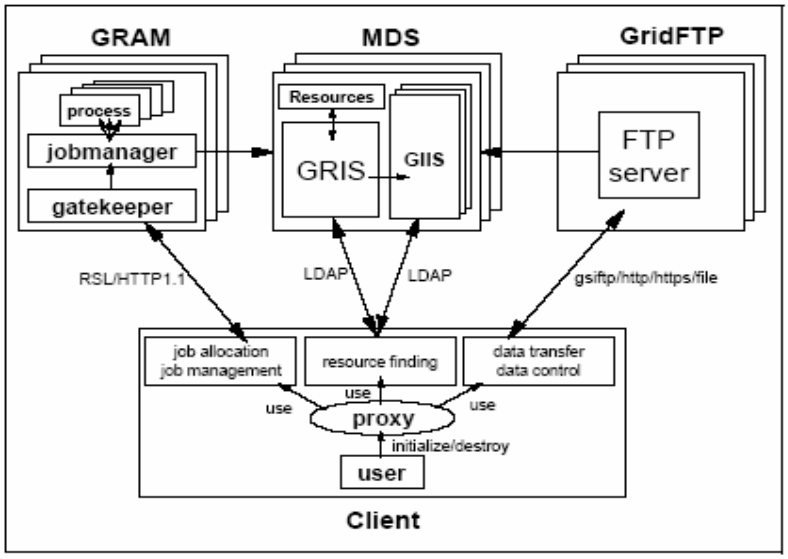

Fig 4.2 System overview of Globus [24]

4.1.2.1 Grid Security Infrastructure (GSI)

GSI provides elements for secure authentication and communication in a Grid. The infrastructure is based on the SSL protocol (Secure Socket Layer), public key encryption, and x.509 certificates. For a single sign-on, Globus add some extensions on GSI. It is based on the Generic Security Service API, which is a standard API promoted by the Internet Engineering Task Force (IETF).

These are the main functions implemented by GSI:

- Single/mutual authentication
- Confidential communication
- Authorization
- Delegation

4.1.2.2 Grid Resource Allocation Manager (GRAM)

GRAM is the module that provides the remote execution and status management of the execution. When a client submits a job, the request is sent to the remote host and handled by the **gatekeeper** daemon located in the remote host. Then the **gatekeeper** creates a job manager to start and monitor the job. When the job is finished, the job manager sends the status information back to the client and terminates [24].

GRAM contains the following elements:

- The **globusrun** command
- Resource Specification Language (RSL)
- The gatekeeper daemon
- The job manager
- The forked process
- Global Access to Secondary Storage (GASS)

The globusrun command

The **globusrun** command submits and manages remote jobs and is used by almost all GRAM client tools. This command provides the following functions:

- Request of job submission to remote machines

Job submission uses security functions (such as GSS-API) to check mutual authentication between clients and servers, and also to verify the rights to submit the job.

- Transfer the executable files and the resulting job-submission output files

The **globusrun** command can get the standard output of job results from remote machines. It uses GASS to provide the secure file transfer between Grid machines.

Resource Specification Language (RSL)

RSL is the language used by the clients to submit a job. All job submission requests are described in RSL, including the executable file and condition on which it must be executed. You can specify, for example, the amount of memory needed to execute a job in a remote machine.

Gatekeeper

The gatekeeper daemon builds the secure communication between clients and servers. The gatekeeper daemon is similar to inetd daemon in terms of functionality. However, gatekeeper provides a secure communication. It communicates with the GRAM client (globusrun) and authenticates the right to submit jobs. After authentication, gatekeeper forks and creates a job manager delegating the authority to communicate with clients.

Job manager

Job manager is created by the **gatekeeper** daemon as part of the job requesting process. It provides the interfaces that control the allocation of each local resource manager, such as a job scheduler like PBS, LSF, or Load Leveler. The job manager functions are:

- Parse the resource language
- Allocate job requests to the local resource managers
- Send callbacks to clients, if necessary
- Receive the status and cancel requests from clients
- Send output results to clients using GASS, if requested

Global Access to Secondary Storage (GASS)

GRAM uses GASS for providing the mechanism to transfer the output file from servers to clients. Some APIs are provided under the GSI protocol to furnish secure transfers. This mechanism is used by the **globusrun** command.

4.1.2.3 Monitoring and Discovery Service (MDS)

MDS provides access to static and dynamic information of resources. Basically, it contains the following components:

- Grid Resource Information Service (GRIS)
- Grid Index Information Service (GIIS)
- Information Provider
- MDS client

As illustrated, the information provider obtains the resource information and it is passed to GRIS. GRIS registers its local information with the GIIS, which also registers with another GIIS, and so on. MDS clients can get the resource information directly from GRIS (for local resources) and/or a GIIS (for Grid-wide resources). **gatekeeper**, and job manager. The MDS uses LDAP, which provides the decentralized maintenance of resource information.[22]

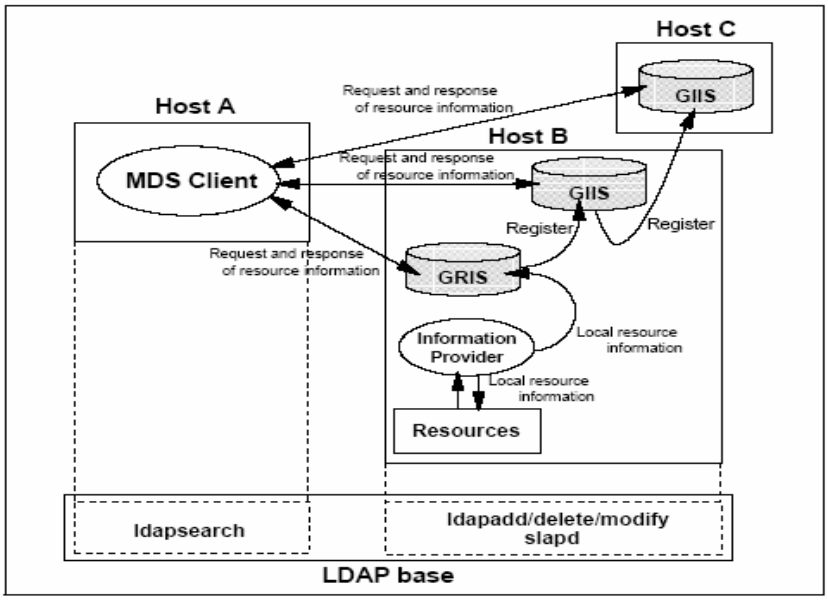

Fig 4.3 MDS Overview [24]

Resource information

Resource information contains the objects managed by MDS, which represent components resources (static and dynamic) as follows:

- Infrastructure components

  For example, name of the job manager or name of the running job
- Computer resources

  For example, network interface, IP address, or memory size

Grid Resource Information Service (GRIS)

GRIS is the repository of local resource information derived from information providers. GRIS is able to register its information with a GIIS, but GRIS itself does not receive registration requests. The local information maintained by GRIS is updated when requested, and cached for a period of time known as the time-to-live (TTL). If GRIS receives no request for the information, the information will time out and be deleted. If a later request for the information is received, GRIS will call the relevant information provider(s) to retrieve the latest information [24].

Grid Index Information Service (GIIS)

GIIS is the repository that contains indexes of resource information registered by the GRIS and other GIISs. It can be seen as a Grid wide information server. GIIS has a hierarchical mechanism, like DNS, and each GIIS has its own name. This means client users can specify the name of a GIIS node to search for information.

Information providers

The information providers translate the properties and status of local resources to the format defined in the schema and configuration files. In order to add your own resource to be used by MDS, you must create specific information providers to translate the properties and status to GRIS.

MDS client

The MDS client is based on the LDAP client command, **ldapsearch**. The MDS client initially performs a search for resource information that you want in your Grid environment.[24]

Hierarchical MDS

The MDS hierarchy mechanism is similar to the one used in DNS. GRIS and GIIS, at lower layers of the hierarchy, register with the GIIS at upper layers. Clients can query the GIIS for any information about resources that build a Grid environment.

4.1.2.4 GridFTP

GridFTP provides a secure and reliable data transfer among Grid nodes. The word GridFTP can refer to a protocol, a server, or a set of tools.[24]

GridFTP protocol

GridFTP is a protocol intended to be used in all data transfers on the Grid. It is based on FTP, but extends the standard protocol with facilities such as multistreamed transfer, auto-tuning, and Globus based security. This protocol is still in draft level.

As the GridFTP protocol is still not completely defined, Globus Toolkit does not support the entire set of the protocol features currently presented. A set of GridFTP tools is distributed by Globus as additional packages. Globus Project has selected some features and extensions defined already in IETF RFCs and added a few additional features to meet requirements from current data Grid projects.

GridFTP server and client

Globus Toolkit provides the GridFTP server and GridFTP client, which are implemented by the in.ftpd daemon and by the **globus-url-copy** command, respectively. They support most of the features defined on the GridFTP protocol. The GridFTP server and client support two types of file transfer: standard and third party. The standard file transfer is where a client sends the local file to the remote machine, which runs the FTP server.

GridFTP tools

Globus Toolkit provides a set of tools to support GridFTP type of data transfers. The gsi-ncftp package is one of the tools used to communicate with the GridFTP Server. This package is available at the following site:

4.2 Alchemi

Alchemi is an open-source .Net based Enterprise Grid computing framework developed by researchers at the GRIDS lab, in the Computer Science and Software Engineering Department at the University of Melbourne, Australia. It allows you to painlessly aggregate the computing power of networked machines into a virtual supercomputer and to develop applications to run on the grid with no additional investment and no discernible impact to users. It has been designed with the primary goal of being easy to use without sacrificing power and flexibility. It has been designed for the Microsoft Windows operating system, which is seen as key factor in industry adoption of grid computing technology since more than 90% of machines worldwide run variants of Windows.

4.2.1 Architecture

Alchemi follows the master-worker parallel programming paradigm [14] in which a central component dispatches independent units of parallel execution to workers and manages them. This smallest unit of parallel execution is a grid thread, which is conceptually and programmatically similar to a thread object (in object-oriented sense) that wraps a "normal" multitasking operating system thread. A grid application is defined simply as an application that is to be executed on a grid [2] and that consists of a number of grid threads. Grid applications and grid threads are exposed to the grid application developer via the object oriented Alchemi .NET API

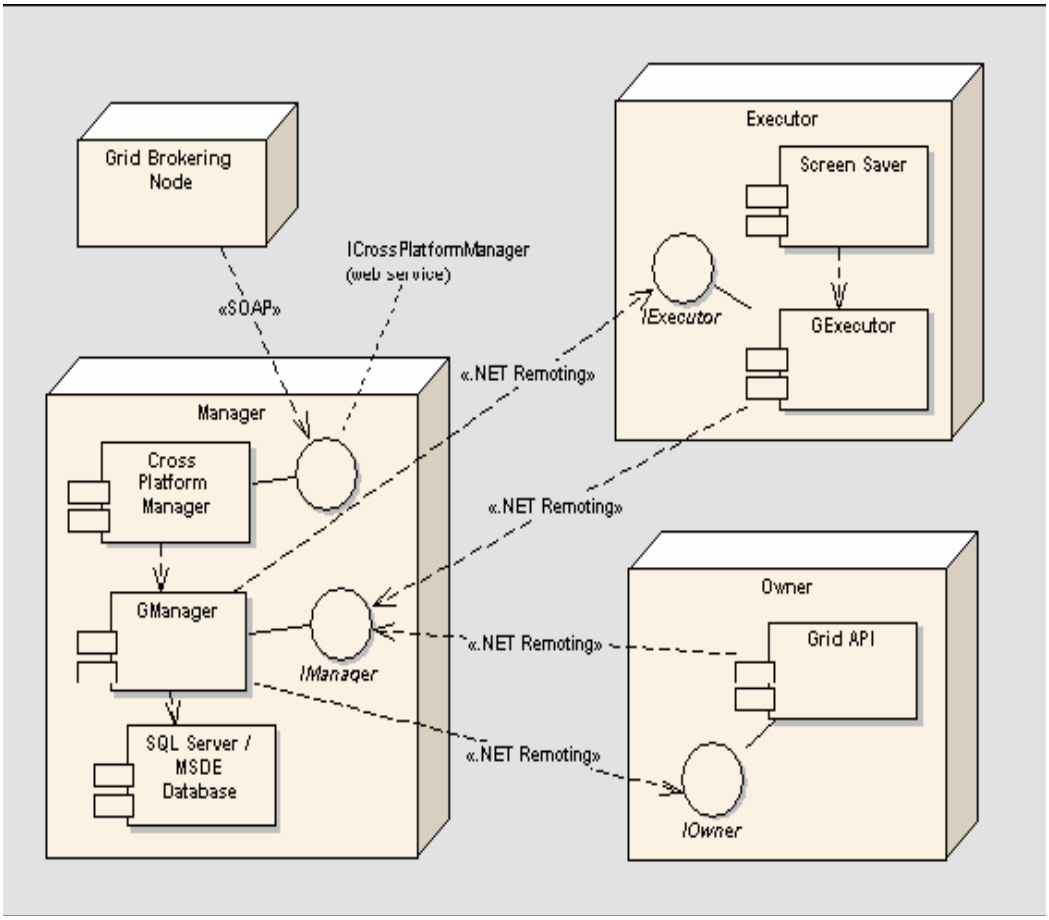

Fig. 4.4 Alchemi architecture [1]

4.2.2 Alchemi components

Alchemi has the following components designed for the grid construction:

- Manager
- Executor
- Owner
- Cross Platform Manger

4.2.2.1 Manger

The Manager manages the execution of grid applications and provides services associated with managing thread execution. The Executors register themselves with the Manager, which in turn keeps track of their availability. Threads received from the Owner are placed in a pool and scheduled to be executed on the various available Executors. A priority for each thread can be explicitly specified when it is created within the Owner, but is assigned the highest priority by default if none is specified. Threads are scheduled on a Priority and First Come First Served (FCFS) basis, in that order. The Executors return completed threads to the Manager, which are subsequently passed on or collected by the respective Owner [2].

4.2.2.2 Executor

The Executor accepts threads from the Manager and executes them. An Executor can be configured to be dedicated, meaning the resource is centrally managed by the Manager, or non-dedicated, meaning that the resource is managed on a volunteer basis via a screen saver or by the user. For non-dedicated execution, there is one-way communication between the Executor and the Manager. In this case, the resource that the Executor resides on is managed on a volunteer basis since it requests threads to execute from the Manager [2]. Where two-way communication is possible and dedicated execution is desired the Executor exposes an interface (**IExecutor**) so that the Manager may communicate with it directly. In this case, the Manager explicitly instructs the Executor to execute threads, resulting in centralized management of the resource where the Executor resides. Thus, Alchemi's execution model provides the dual benefit of:

- Flexible resource management i.e. centralized management with dedicated execution vs. decentralized management with non-dedicated execution; and
- Flexible deployment under network constraints i.e. the component can be deployment as non dedicated where two-way communication is not desired or not possible (e.g. when it is behind a firewall or NAT/proxy server).

Thus, dedicated execution is more suitable where the Manager and Executor are on the same Local Area Network while non-dedicated execution is more appropriate when the Manager and Executor are to be connected over the Internet.

4.2.2.3 Owner

Grid applications created using the Alchemi API are executed on the Owner component. The Owner provides an interface with respect to grid applications between the application developer and the grid. Hence it "owns" the application and provides services associated with the ownership of an application and its constituent threads. The Owner submits threads to the Manager and collects completed threads on behalf of the application developer via the Alchemi API [1].

4.2.2.4 Cross-Platform Manager

The Cross-Platform Manager, an optional sub-component of the Manager, is a generic web services interface that exposes a portion of the functionality of the Manager in order to enable Alchemi to manage the execution of platform independent grid jobs (as opposed to grid applications utilizing the Alchemi grid thread model). Jobs submitted to the Cross-Platform Manager are translated into a form that is accepted by the Manager (i.e. grid threads), which are then scheduled and executed as normal in the fashion described above. Thus, in addition to supporting the grid enabling of existing applications, the Cross-Platform Manager enables other grid middleware to interoperate with and leverage Alchemi on any platform that supports web services (e.g. Gridbus Grid Service Broker) [2].

4.2.3 Alchemi System Configurations

The components discussed above allow Alchemi to be utilized to create different grid configurations: desktop cluster grid, multi-cluster grid, and cross-platform grid (global grid).

Cluster (Desktop Grid)

The basic deployment scenario – a cluster (shown in Figure 3) - consists of a single Manager and multiple Executors that are configured to connect to the Manager. One or more Owners can execute their applications on the cluster by connecting to the Manager. Such an environment is appropriate for deployment on Local Area Networks as well as the Internet.

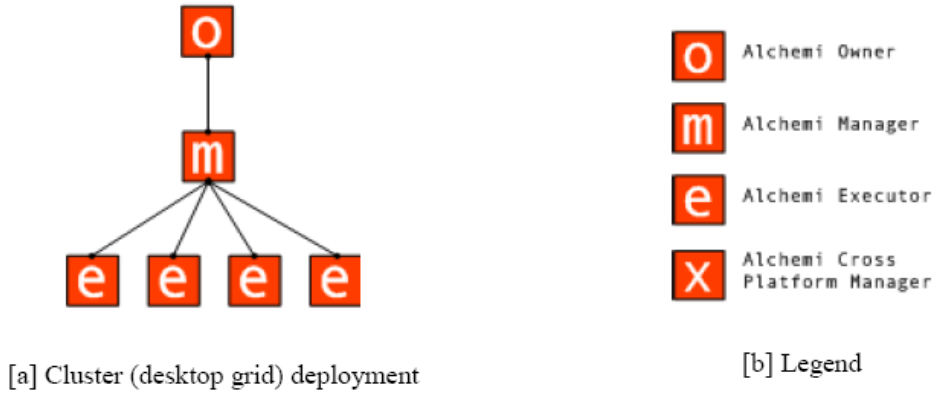

Fig. 4.5 Alchemi desktop Grid [2]

The operation of the Manager, Executor and Owner components in a cluster is as described above.

Multi-Cluster

Connecting Managers in a hierarchical fashion creates a multi-cluster environment. As in a single-cluster environment, any number of Executors and Owners can connect to a Manager at any level in the hierarchy. An Executor and Owner in a multi-cluster environment connect to a Manager in the same fashion as in a cluster and correspondingly their operation is no different from that in a cluster [1].

The key to accomplishing multi-clustering in Alchemi's architecture is the fact that a Manager behaves like an Executor towards another Manager since the Manager

implements the interface of the Executor. A Manager at each level except for the topmost level in the hierarchy is configured to connect to a higher level Manager as an "intermediate" Manager and is treated by the higher level-Manager as an Executor. Such an environment is more appropriate for deployment on the Internet.

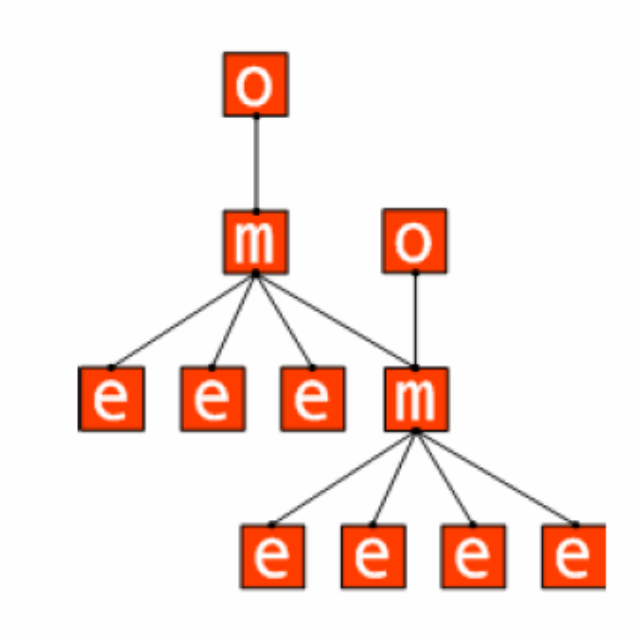

Fig. 4.6 Alchemi multi-cluster Grid [1]

The operation of an intermediate Manager in a multi-cluster environment therefore, must be discussed with respect to the behavior of an Executor and is as follows. Once Owners have submitted grid applications to their respective Managers, each Manager has "local" grid threads waiting to be executed. As discussed, threads are assigned the highest priority by default (unless the priority is explicitly specified during creation) and threads are scheduled and executed as normal by the Manager's local Executors. [1] Note that an 'Executor' in this context could actually be an intermediate Manager, since it is treated as an Executor by the higher-level Manager. In this case after receiving a thread from the higher-level Manager, it is scheduled locally by the intermediate Manager with a priority reduced by one unit and is executed as normal by the Manager's local 'Executors' (again, any of which could be intermediate Managers) [1].

In addition, at some point the situation may arise when a Manager wishes to allocate a thread to one of its local Executors (one or more of which could an intermediate

Manager), but there are no local threads waiting to be executed. In this case, if the Manager is an intermediate Manager, it requests a thread from its higher-level Manager, reduced the priority by one unit and schedules it locally.

In both of these cases, the effect of the reduction in priority of a thread as it moves down the hierarchy of Managers is that the "closer" a thread is submitted to an Executor, the higher is the priority that it executes with. This allows a portion of an Alchemi grid that is within one administrative domain (i.e. a cluster or multi-cluster under a specific "administrative domain Manager") to be shared with other organizations to create a collaborative grid environment without impacting on its utility to local users.

As with an Executor, an intermediate Manager must be configured for either dedicated or non-dedicated execution [1]. Not only does its operation in this respect mirror that of an Executor, the same benefits of flexible resource management and deployment under network constraints apply.

Cross-Platform Grid

The Cross-Platform Manager can be used to construct a grid conforming to the classical global grid model (Figure).

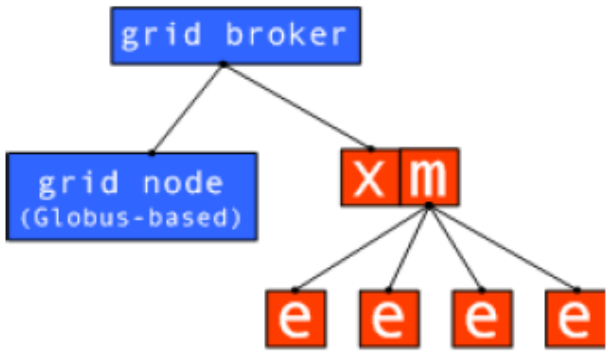

Fig 4.7 Cross Platform grid [1]

A grid middleware component such as a broker can use the Cross-Platform Manager web service to execute cross-platform applications (jobs within tasks) on an Alchemi node (cluster or multi-cluster) as well as resources grid-enabled using other technologies such as Globus [1].

4.3 Condor

Condor is a high-throughput distributed batch computing system. Condor is a sophisticate job scheduler developed by the condor research project at the university of Wisconsin-Madison Department of Computer science. Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion [6].
Users submit their jobs to Condor, and Condor subsequently chooses when and where to run them based upon a policy, monitors their progress, and ultimately informs the user upon completion.

In addition to previous queuing system Condor provides some novel features. It can be used to mange the cluster of dedicated nodes. Condor's several unique mechanisms enable it to harness effectively waste CPU power from otherwise idle desktop workstations

4.3.1 Why use Condor?

Condor takes advantage of computing resources that would otherwise be wasted and puts them to good use. Condor streamlines the scientist's tasks by allowing the submission of many jobs at the same time. In this way, tremendous amounts of computation can be done with very little intervention from the user. Moreover, Condor allows users to take advantage of idle machines that they would not otherwise have access to [27].

Condor provides other important features to its users. Source code does not have to be modified in any way to take advantage of these benefits. Code that can be re-linked with the Condor libraries gains two further abilities: the jobs can produce checkpoints and they can perform remote system calls.

A checkpoint is the complete set of information that comprises a program's state. Given a checkpoint, a program can use the checkpoint to resume execution. For long-running computations, the ability to produce and use checkpoints can save days, or even weeks of accumulated computation time. If a machine crashes, or must be rebooted for an administrative task, a checkpoint preserves computation already completed. Condor makes checkpoints of jobs, doing so periodically, or when the machine on which a job is executing will shortly become unavailable. In this way, the job can be continued on another machine (of the same platform); this is known as process migration [27].

A user submits a job to Condor. The job is executed on a remote machine within the pool of machines available to Condor. Condor preserves minimal impact on and the security of the remote machine through remote system calls. When the job does a system call, for example to do an input or output function, the data is maintained on the machine where the job was submitted [27]. The data is not on the remote machine, where it could be an imposition.

By linking in a set of Condor libraries, system calls are caught and performed by Condor, instead of by the remote machine's operating system. Condor sends the system call from the remote machine to the machine where the jobs were submitted. The system call's function executes, and Condor sends the result back to the remote machine [27].

This implementation has the added benefit that a user submitting jobs to Condor does not need an account on the remote machine.

4.3.2 Condor architecture

Condor’s key activities - job-resource allocation, job startup and execution, and metadata collection and display – are kept separate, allowing compartmentalization of

Condor into clearly defined components, distributed amongst submission site, central manager and execution site, as illustrated in figure: [5]

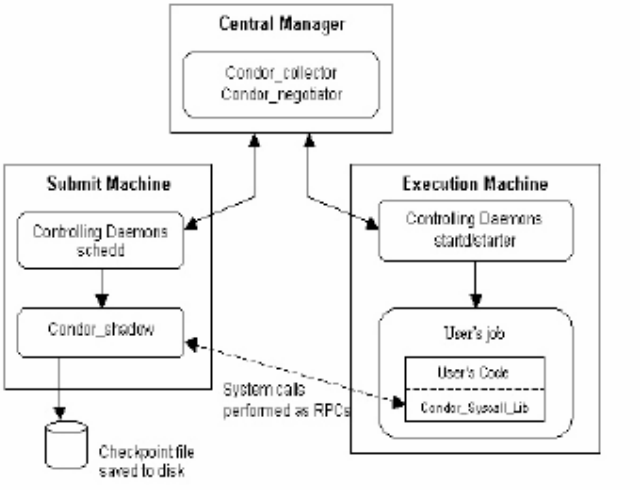

Fig. 4.8 Condor Architecture [5]

Central Manager: For every condor pool a single central manager is responsible for collecting resource characteristics and usage information (i.e. accounting) from all machines in the pool and enforcing *community policies [5]*. It is based on this collected information, and on *user priorities,* that job execution requests can be *matched* to suitable resources for execution during a negotiation cycle.

Submit Machine: This system client allows users to submit jobs to a local virtual 'queue' (scheduler - *schedd*). The scheduler will request resource allocations for its jobs from the central manager during a negotiation cycle [5]. Once a resource has been allocated to a job, the scheduler will spawn a *shadow* daemon responsible for managing the remote execution that job, and perform tasks such as state check pointing, rescheduling the job in case of failure, or perform system calls made by the job running remotely on the local machine.

Execute Machine: The execute machine, represented by the *startd* daemon, runs jobs on behalf of clients [5]. It advertises its capabilities and usage information - as well as requirements and preferences upon a match - to the central manager, and manages the local execution of the job (via a spawned *starter* daemon), whilst protecting resource owner policies (e.g. a job may be vacated if the user touches the keyboard).

4.3.3 Condor daemons

A Condor workstation (machine) [21] runs two Condor daemons, the scheduler daemon *Schedd* and the starter daemon *Startd*. One Condor machine is designated to run Central Manager (CM), which consists of two daemons, the *Negotiator* and the *Collector*. The Condor daemons cooperate with each other by exchanging messages. The tasks of the daemon are:

- The *Schedd* [21][26] maintains a queue of jobs submitted on its machines, prioritizes them and controls the remote startup of these jobs. The requirements of each job are stored in a job context.

- The *Startd* monitors [21][26] the state of its machines, advertises its resources towards the CM, handles the startup and monitors the execution of a job submitted at another machine.

- The *Collector* [21][26] gathers information of the machines in the pool. The information, which is sent by the *Schedd* (job queue information), the *Startd* (machine state and resources), and the *Negotiator* (machine properties), is stored in a machine context.

- The *Negotiator* [15][16] prioritizes the machines, and matches contexts of jobs and contexts of available machines.

4.4 Nimrod-G

Nimrod-G is a tool for automated modeling and execution of parameter sweep applications (parameter studies) over global computational Grids. It provides a simple declarative parametric modeling language for expressing parametric experiments. A domain expert can easily create a plan for a parametric experiment and use the Nimrod-G system to deploy jobs on distributed resources for execution [19]. It uses novel resource management and scheduling algorithms based on economic principles. Specifically, it supports user-defined deadline and budget constraints for schedule optimizations and manages supply and demand of resources in the Grid using a set of resource trading services

4.4.1 Architecture

Nimrod-G has been developed as a Grid resource broker based on the GRACE framework. It leverages services provided by Grid middleware systems such as Globus, Legion, and the GRACE trading mechanisms. The middleware systems provide a set of low-level protocols for secure and uniform access to remote resources; and services for accessing resources information and storage management. The modular and layered architecture of Nimrod-G is shown in Figure 4.2. The Nimrod-G architecture follows hourglass design model that allows its implementation on top of different middleware systems and enables the usage of its services by multiple clients/applications.

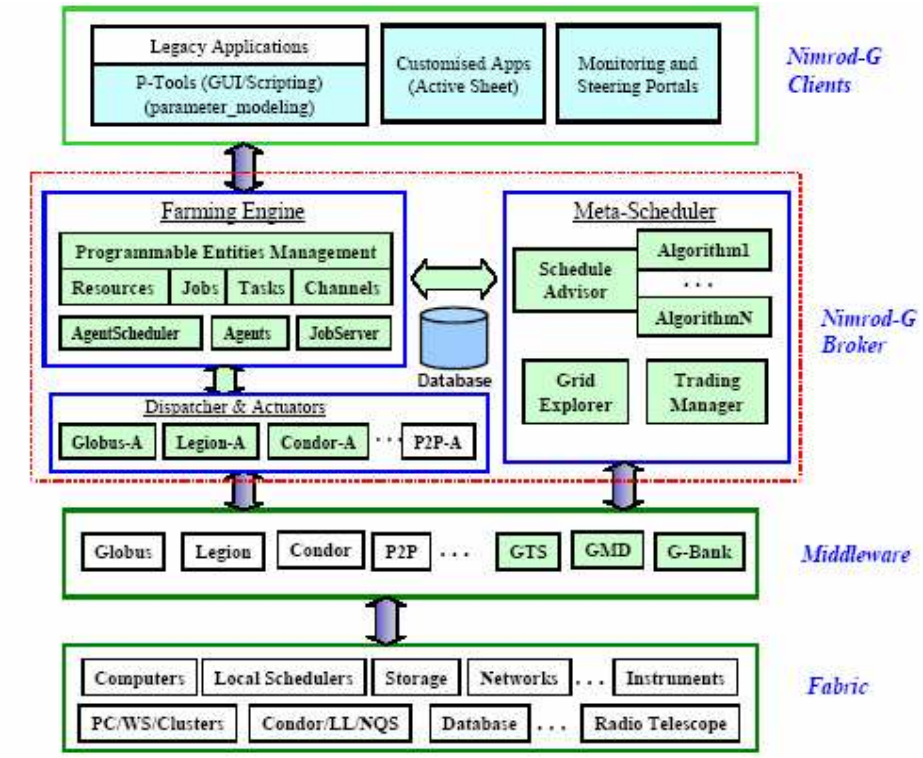

Fig4.9 Nimrod/G Layered Architecture [19]

The key components of Nimrod-G resource broker consist of:

- Nimrod-G Clients, which can be:
    - Tools for creating parameter sweep applications.
    - Steering and control monitors, and
    - Customized end user applications (e.g., Active Sheets).

- The Nimrod-G Resource Broker, that consists of:
    - A Task Farming Engine (TFE),
    - A Scheduler that performs resource discovery, trading, and scheduling,
    - A Dispatcher and Actuator, and
    - Agents for managing the execution of jobs on resources.

The Nimrod-G broker architecture leverages services provided by lower-level different Grid middleware solutions to perform resource discovery, trading, and deployment of jobs on Grid resources[19].

4.5 Grid Middleware Characteristics

The various characteristics, which the grid middleware's should have, are:

- Transparency: The grid middleware should provide the users an environment where they are unaware of the underlying complexities like where the resources are located, who is the owner of that particular resource. The middleware should provide all the users a very seamless access. It should give a fell to the users as if the resources are present at his site only.

- Robustness: As the very nature of the grid, they need to operate in very dynamic environment. The virtual organizations are constructed dynamically at run time depending upon the requirement of a particular application. The chances of the failure of the grid nodes increase; the middleware should be able to deal with the faults. It should be able to dynamically relocate the load if a node becomes unavailable.

- Security: The grid spans over a large geographical area and multiple organizations, which dynamically collaborate at run time. Every organization has its own security policies, authentication mechanisms and security requirements. The middleware must take care of these security policies so that the integrity and the confidentiality of these organizations is maintained.

- Persistency: The middleware should be able to mange the states and keep track of them, so that the user can retrieve the desired information, like how much the job is completed, how long will it still take to finish the job etc, at any time.

- Scalability: the middleware should scale the grids in a very nice manner as the grids are the talk of the day, and they are expanding.

- Ease to use/ program: The middleware should provide the users an environment where they can write code very easily so that it can be run on grid environment. The programming environment should be easy to learn.

In this chapter, we discuss in details about the various grid middleware's in the market. We discuss in depth what is the architecture implementation of all these middleware's with the help of which the grid environment can be realized. Then at the last of chapter we discuss what are the features, which a grid middleware should, met.

Chapter 5: Grid Middleware Set Up & Comparative Analysis

5.1. Globus Grid set up

The globus is a toolkit, which is used to create a computational grid. The following steps were followed for installation of globus:

5.1.1. Downloading GT4

The globus for the Linux was downloaded from the www.globus.org and was saved in /home/globus directory

5.1.2. Installing Prerequisite

Installation of globus needs some prerequisite soft wares for their successful installations so following packages were installed.

- **Apache Ant**

1. Download the Binary Distribution of Ant
2. Extract it into the folder where to install it.
3. set environment ANT_HOME to /usr/local/ant (the installed location)

root# cd /usr/local
root# mkdir apache-ant
root# tar –zxvf apache-ant.tar.gz

- **Sun J2SE**

1. Download J2SE Binary for your platform and do the simple step of extract into the folder.
2. Set environment JAVA_HOME=/usr/local/J2sdk

root# cd /usr/local
root# mkdir j2sdk
root# cd j2sdk

root# tar –zxvf ~/j2sdk-linux.tar.gz

- **PostgreSQL**

PostgreSQL is an open source implementation of RDBMS system supporting TCP/IP communication.

root# cd /usr/local

root# mkdir pgsql

root# cd $HOME

root# tar –zxvf postgresql-base-8.0.3.tar.gz

root# cd postgresql-base-8.0.3

root# ./configure –prefix=/usr/local/pgsql

root# make

root# make install

Configuring PostgreSQL needs the following steps,

add a new user postgres with normal privileges

adduser postgres

Create a Data Directory to hold the database,

mkdir /usr/local/pgsql/data

chown postgres /usr/local/pgsql/data

su – postgres

set environment PGDATA = /usr/local/pgsql/data and also update the PATH variable to $PATH:/usr/local/pgsql/bin in .bshrc

$(post)/sbin/initdb

This command creates a file postgresql.conf in the data directory.

Create new file in /etc/init.d named postgresql

- type the following into that file

su – postgres –c "/usr/local/pgsql/bin/pg_ctl start –o "-i" –l logfile –D /usr/local1/pgsql/data"

create a link in the directory /etc/rc3.d as follows

ln –s ../init.d/postgresql S99postgresql

7. Make postgresql as an executable using these commands,

chmod u+x postgresql

chown postgres postgresql

- **sudo**

sudo is used to run commands that require Super User privilege by the ordinary user itself. It is the important requirement for WS-GRAM. So, configuring sudo is very important for successful execution of jobs using WS-GRAM. Sudo can be installed as follows

1. Download the binary. And Extract it into appropriate folder
2. set PATH if needed
3. Type $(sudo)/sbin/visudo and add the following entry. It will open /etc/sudeors file. Don't use vi to edit this file.

Globus GRAM entries

globus ALL=(meena) NOPASSWD :/opt/globus/GT4.0.0/libexec/globus-gridmap-and-execute –g /etc/grid-security/grid-mapfile /opt/globus/GT4.0.0/libexec/ globus-job-manager-script.pl *

globus ALL=(meena) NOPASSWD: /opt/globus/GT4.0.0/libexec/globus-gridmap-and-execute –g /etc/grid-security/grid-mapfile /opt/globus/GT4.0.0/libexec/globus-gram-local-proxy-tool *

4. visudo itself reports if there are any error in the entries.

5.1.3. Setting appropriate paths

Before getting into configuration of the globus it is required that all paths are set correctly. .bashrc file as the installation of globus cross check these paths :

export JAVA_HOME /usr/local/J2SDK/j2sdk

export JAVA_HOME /usr/local/J2SDK/j2sdk

export ANT_HOME /usr/local1/apache-ant

export GLOBUS_LOCATION= /usr/local1/globus

export PGDATA= /usr/local1/pgsql/data

export PATH = $PATH:$ANT_HOME/bin:$JAVA_HOME/bin

5.1.4. Configuring and Installing Globus

After the globus is downloaded and the prerequisite softwares are installed the globus can be installed using the following steps.

globus$ cd /home/globus

globus$ tar –zxvf gt4.0.0-all-source-installer.tar.gz

globus$ ./configure –-prefix=$GLOBUS_LOCATION

globus$ make | tee build.log

root# make install

5.1.5. Setting up Security

One of the pillars of the grid infrastructure is the security. This security must be well emphasized before allowing any of the resources publicly to the grid. In a grid, each resource must have appropriate certificates to make it trust worthy. These certificates should be issued by an authority organization which is most trusted by the user and also the authority must know well about the user. This organization is in general called a Certificate Authority (CA). Today, every security system will have a head Certificate authority.

- **Creating a new Certificate Authority (CA)**

The certificate authority is the in charge for issuing certificates for host and also the users of the grid. The certificate authority can be a most trusted external organization or even a personal certificate authority can be created using the software called SimpleCA provided along with Globus. Each system in the grid must follow the same CA and a single system can have multiple Cas configured to make it a part of two grids. Now here is list the various steps that are to be followed to install SimpleCA provided with Globus.

1. Run the script $GLOBUS_LOCATION/setup/globus/setup-simple-ca

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

2. gt5.tiet.ac.in is made to act as the chief CA host for the TIET Grid.

4. Subject of the CA installed is “cn=Globus SimpleCA, ou=simpleCA, ou=gt5.tiet.ac.in, o=tiet.ac.in”..

5. Email of administrator of the SimpleCA at gt5.tiet.ac.in is msingh@tiet.ac.in This email can be used to send the certificate requests by the clients.

6. Expiration date of the certificates issued by this CA is 5 years

7. PEM pass phrase is the password or the key that is used to identify the CA. It is important since it has to be issued by the admin whenever he signs the certificates.

8. The CA hash is displayed at the summary of this setup. Note down the hash that displayed there which is used for configuring CA on other systems.

9. Final step is to setup Grid Security Infrastructure(GSI) for the grid.

globus$ cd $GLOBUS_LOCATION/setup/simpleca-<hash>/

globus$ bash setup-gsi –default

- **Obtaining required certificates from CA**

GT4 actually follows mutual authentication for job execution and job submission from one machine to another. Mutual authentication requires security certificates to be compared. These certificates are of three types:

1. Host Certificates

These certificates must be there in each and every system to be a part of the grid. These certificates are read only files even for the root and are can be located in /etc/grid-security

Host certificate is obtained as follows:

1. Login as root,

grid-cert-request –host `hostname`

2. Three files are created in /etc/grid-security directory,

a. hostcert.pem – empty file to be replaced with certificate from CA.

b. hostkey.pem – holding the passphrase in RSA 1024-bit encrypted format

c. hostcert_request.pem – to be send to the CA for a signed certificate.

4. Signing of Host Certificates by CA:

a. grid-ca-sign –in hostcert_request.pem –out hostsigned.pem

b. After that copy the signed cert hostsigned.pem into the /etc/grid-security/hostcert.pem.

2. User Certificates

These certificates are used to identify a particular user in the grid. User certificates are, by default, placed under $(HOME)/.globus directory.

User certificates are obtained as follows:

1. Every user needs a user certificate to run Globus Jobs. First, Login as the user, me as meena login give the following command

grid-cert-request – asks for your desired DN name and pass phrase

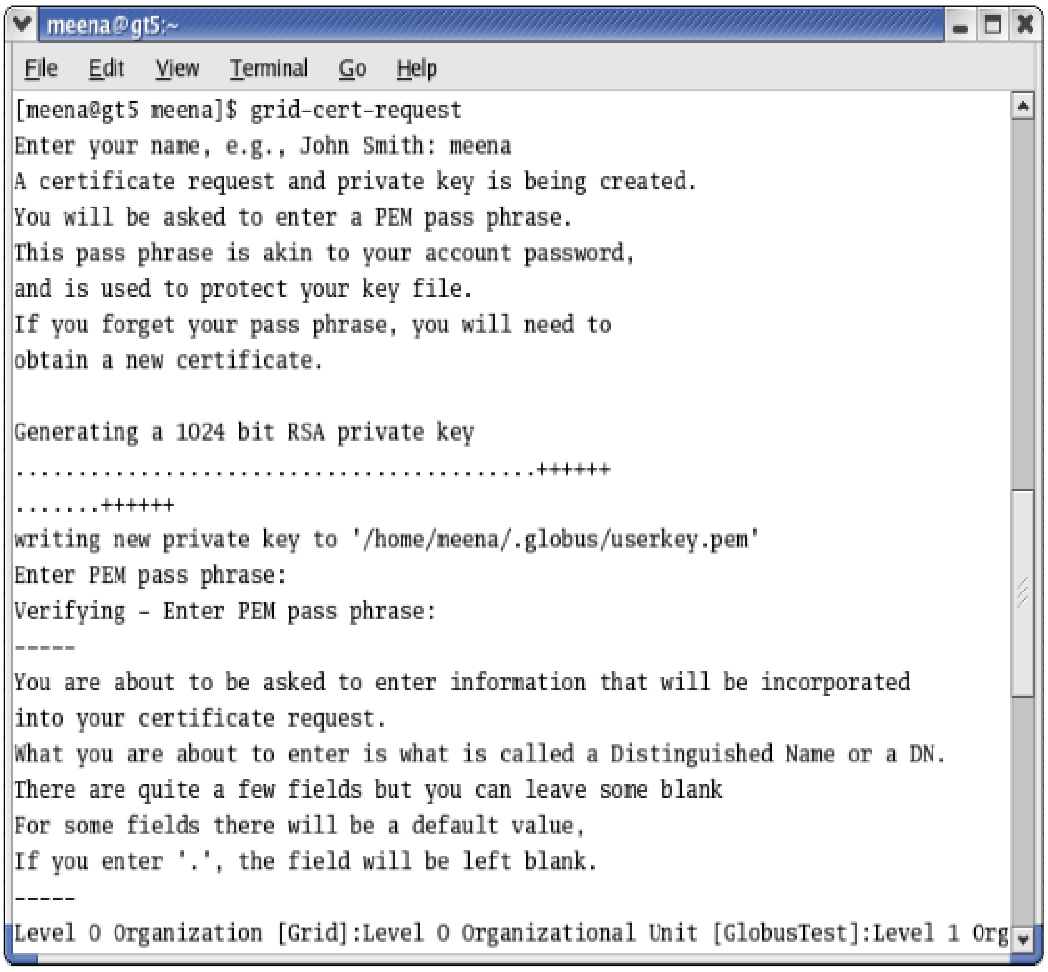

Fig 5.1 User-Certificate request

It will leaves three files created in $(HOME)/.globus directory

a. usercert.pem

b. userkey.pem

c. usercert_request.pem

3. Signing and copying of certificate

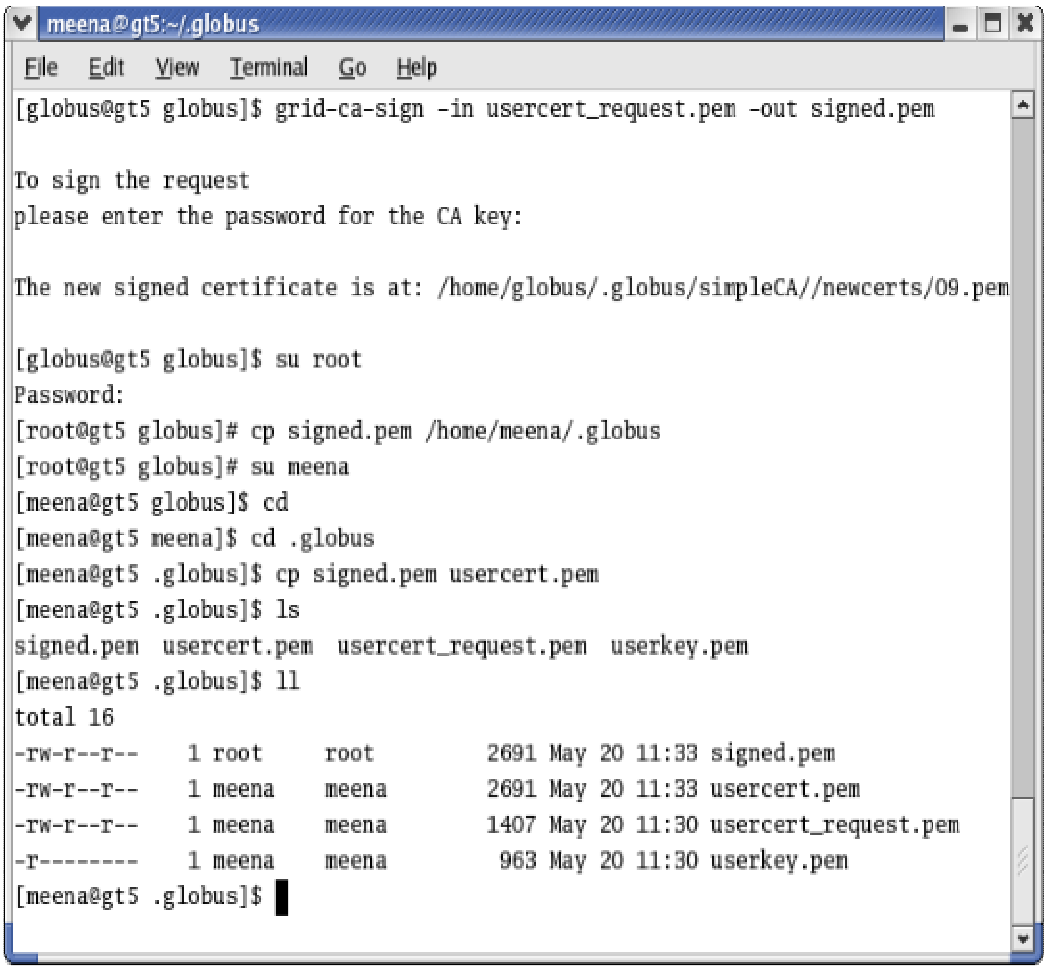

Fig 5.2 Signing User-Certificate

3. Proxy Certificates

These are short-lived certificates, which are available openly and are used for mutual authentication. These certificates are necessary for every operation on Globus. These certificates are usually placed in /tmp directory.

Obtaining proxy certificate

Proxy certificates are at the lowest level of authentication process. It can also be used for testing whether the whole security is configured correctly. The command to do it as follows,

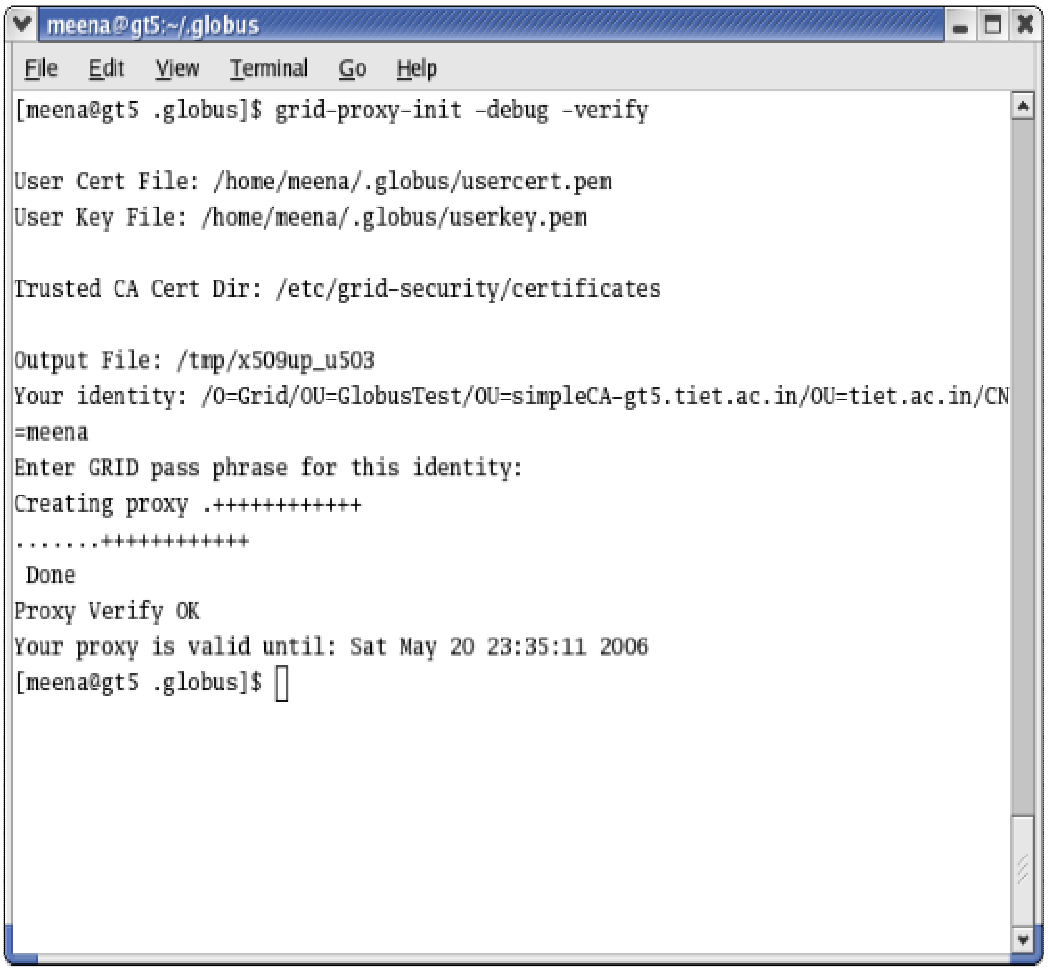

Fig 5.3 Generating Proxy Certificate

The success in this command means that the GSI is configured perfectly and the security is of no problem for this particular user. The final step is to map the DN with a local username in the machine.

- **Making entry in grid-mapfile**

The grid-mapfile is a file which contains the information of the various DN that can use that system as a grid resource, mapped with the local user name.

editing the grid-mapfile for adding the following entry:

"/O=Grid/OU=GlobusTest/OU=simpleCA-gt5.tiet.ac.in/OU=gt5.tiet.ac.in/CN= meena" meena

Finally, check the consistency of the grid-mapfile using the command, root# bash $GLOBUS_LOCATION/sbin/grid-mapfile-check-consistency

5.1.6. Creating container certificates and keys

Since only root has access to host certificates, some of the grid services need a separate host certificates for successful execution.

root# cd /etc/grid-security

root# cp hostkey.pem containerkey.pem

root# cp hostcert.pem containercert.pem

root# chown globus:globus containerkey.pem containercert.pem

5.1.7. Configuring GridFTP and Globus Gatekeeper

These are two basic grid-services offered under the pre-ws part of the globus.

They are:

1. GridFTP
2. Globus Gatekeeper

GridFTP is default FTP used for file transfer between the hosts in the grid. It follows older methodology of Globus using pre-ws.gsiftp is the protocol used with GridFTP. Globus Gatekeeper is what that checks and manages the security between the job submission and execution.

Created a file named gsiftp in /etc/xinetd.d and entered the following into it,

```
service gsiftp
{
instances = 100
socket_type = stream
wait = no
user = root
env +=
GLOBUS_LOCATION=/usr/local/globus
env +=
LD_LIBRARY_PATH=/usr/local/globus/lib
server = /usr/local/globus/sbin/globusgridftp-
server
server_args = -i
log_on_success += DURATION
nice = 10
disable = no
```

}

Create a file named gsigatekeeper in /etc/xinetd.d and enter the following into it,

```
service gsigatekeeper
{
instances = 100
socket_type = stream
wait = no
user = root
env += GLOBUS_LOCATION=/User/local/globus
env += LD_LIBRARY_PATH=/User/local/globus/lib
server = /User/local/globus/sbin/globusgatekeeper
server_args = -conf /Users/globus/globus/etc/globusgatekeeper.
conf
log_on_success += DURATION
nice = 10
disable = no
}
```

Updating /etc/services

This file holds the various reserved ports and the protocols running on those ports. gsiftp and gsigatekeeper are two such TCP protocols. This file is important because if firewall is set, there will be various port restrictions which can be identified from this file. In new Apple Mac Systems, Ipv6 and UDP were configured. So, in these systems you will have two entries of gsiftp and gatekeeper, one for TCP and one for UDP. But this is not necessary,

The entries into the /etc/services,

```
gsiftp 2811/tcp
gsigatekeeper 2119/tcp
```

5.1.8. Configuring RFT

Configuring RFT means that of configuring the PostgreSQL because just creation of new database for RFT is more than enough for successful execution of RFT based services.

Configuring PostgreSQL

All the below operation must be done as root.

Login as postgres user,

1. Enter into PGDATA directory.

2. Edit pg_hba.conf

4. Add the entry at the end as follows; all entries should be separated by a tab

host rftDatabase globus 173.31.5.104 255.255.255.0 trust

Enter the following commands as mentioned,

4. /etc/init.d/postgresql restart

5. createuser globus

Now, Login as globus, and continue,

6. createdb rftDatabase

7. psql –d rftDatabase –f

$GLOBUS_LOCATION/share/globus_wsrf_rft/rft_schema.sql

Now, the database with all the tables required for RFT is created successfully.

5.1.9. Starting Container

To start the container the postgres server must be running and the proxy certificates should be there, and then run the command *globus-start-container*.

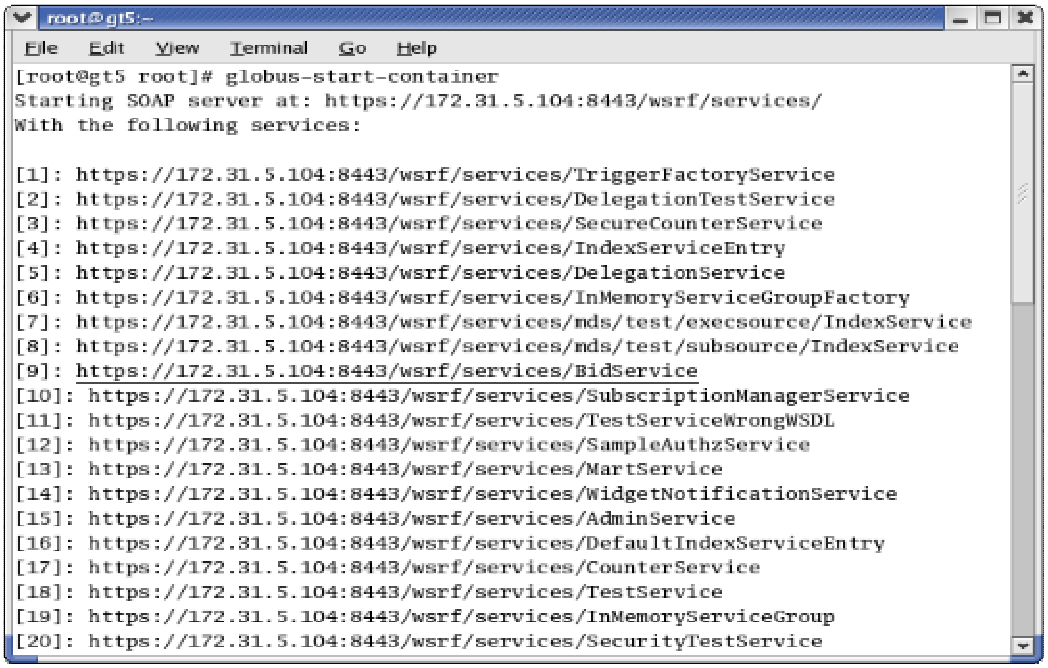

Fig 5.4 Running Container

5.1.10. Running Job on container

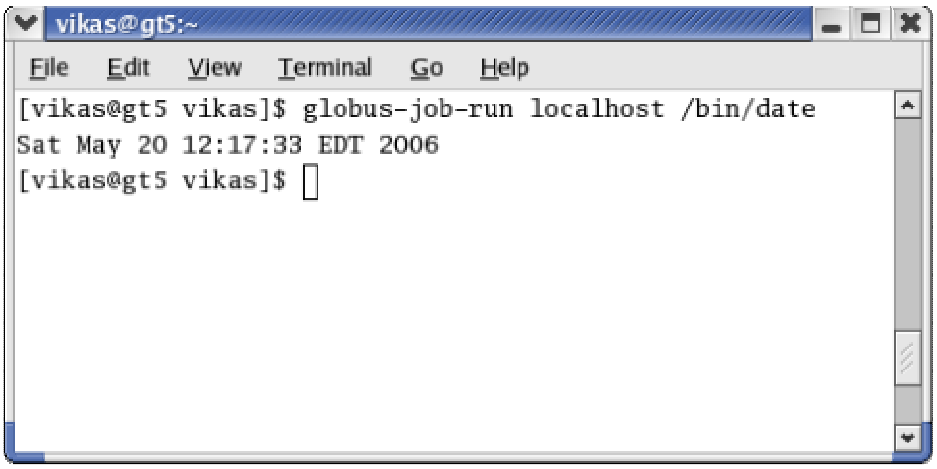

Fig 5.5 Running Job on Globus

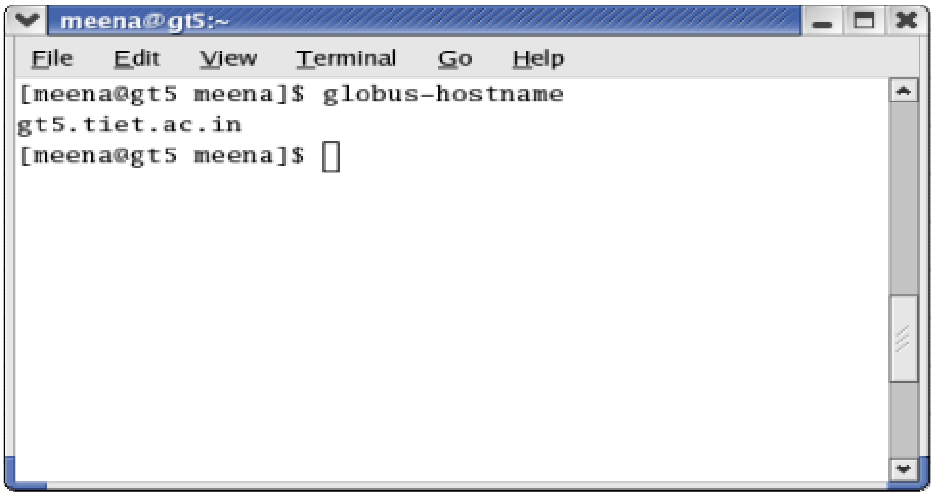

Fig 5.6 Grid Identification

This finishes with the installation of globus on first machine. Globus was installed on two more machines, all steps are same except the security here we use the CA package that was created on the gt5 machine and copy the certificates of meena to the respective directory. And we need to add the corresponding entries of the hosts in /etc/hosts file

5.2 Nimrod/g Installation

After the installation of the globus the nimrod/G broker was installed on the gt5.tiet.ac.in machine following the instructions.

5.3 Alchemi Grid Setup

The fellow colleagues set up the Alchemi Desktop grid. The grid setup up requires the .net framework on all the nodes that need to be part of grid and the sql server on one machine. The details of the grid is as follows:

Central manger

The Tiet-goq0zlzs in research lab is designated as central manger. This node also has Sql server installed.

Executors

The Alchemi Grid consists of the following executors:

- research3
- research4
- Grid
- Software

5.4 Condor pool Setup

The fellow colleagues set up the Alchemi Desktop grid. The installation requires the installation of central manger and the pooled resource workstation. The details of the grid is as follows:

Central manger

The Tiet-goq0zl1zs0 in research lab is designated as central manger. This node also has Sql server installed.

Resource Pool

The Alchemi Grid consists of the following executors:

- research3
- research4

5.5 Comparative analysis of Middlewares

As discussed in the problem statement, the middlewares are compared on the basis of following characteristics:

5.5.1 Category

The category defines whether the middleware is user level or core middleware. The globus, alchemi, condor are core middlewares where as the nimrod/G is a user level middleware. Nimrod/g requires a core middleware for it's functioning. Here we use it with the globus.

5.5.2 Security

Security is key to the grid environment. The globus has GSI pyramid for the security. It uses Public Key infrastructure and X.509 certificates for the authentication purposes. It provides single sign in authentication mechanism and proxy certificates are created and they are delegated. The alchemi has role-based security. The manager is configured to support anonymous or non-anonymous Executors. The Alchemi administrator configures user, group and permission. Users connect to the manger providing the details, and then manager authenticates a user. In Nimrod/G there is as such no security mechanisms. It uses the security provided by the low-level middleware services. Here we have used the globus proxy certificates by setting the variable X509_USER_PROXY.Condor provides high support for authorization, authentication, encryption. When the condor is installed the default configuration setting include none of them. The administrator, through the use of macros, enables these features.

5.5.3 Architecture

The globus follows the hourglass model, in which the core functionalities are in the center. It has a layered and modular architecture. Each module and layer focuses on a particular aspect. The alchemi has hierarchal, master slave architecture. It has a centralized manager and the executors, which connect to this manager. Condor too has a hierarchal architecture. The nimrod/g has a component based layered architecture. These components interact with each other to deliver the functionality.

5.5.4 Scalability

Scalability indicates the ability of a system to increase total throughput under an increased load and when hardware or software resources are added. The globus is very close to the hardware. The scalability is direct result. It can scale till Internet. Alchemi has a centralized manger so the load on the manger increases if the grid is expanded to a large extent. Similarly in the condor the centralized architecture limits the scalability. Nimrod/G is extensible enough to use the underlying middleware services.

5.5.5 Programming environment

Globus provides the replacement libraries for UNIX & C libraries, Special MPI library (MPICH –G), CoG (Commodity Grid) kits in Java, Python, CORBA, Matlab, Java Server Pages, Perl and Web Services. The Alchemi uses the Grid multithreaded model for the execution of the applications. The condor has support for the C, Java, and MPI environment. The Nimrod provides a parametric language to describe the executable written in C or java.

5.5.6 Run Time Platform

The run time platform for the globus is the unix like platforms and the windows. The Alchemi run on the windows environment having .net framework installed. The Condor has support for both widows and unix platforms. The run time platform plays an important role. As if a user familiar with linux environment cannot use the alchemi.

5.5.7 Ease of use/Understand

The globus is very close to hardware level. It uses the complex calls and the commands to run the jobs and it operates on the command line interface. So it is not user friendly. The alchemi has a graphical interface and offers the simple API to use the environment. The condor too works only on command line interface and user need to learn commands to operate.

5.5.8 Scheduling policy

Globus does not have its own scheduler it uses the above layer functionalities for the scheduling. The Condor Scheduling policy is Performance centric i.e. it tries to maximize the throughput of overall system. The nimrod/g has market driven scheduling policy i.e. it uses the economical principles. Alchemi has system centric Scheduling policy

5.5.9 Type of applications

The condor is used for high throughput computing applications. The nimrod/g is used for scientific parametric sweep applications. The globus and alchemi are used for the applications requiring large computations.

5.5.10 Implementation Technologies

These all are open source projects. The globus has been implemented using C and java technologies. The alchemi has been implemented using the C#, perl. The condor has been implemented using C and java.

Middleware / Property	Globus	Alchemi	Condor	Nimrod/G
Category	Core Level	Core Level	Core Level	User Level
Security	PKI, X.509 No Authorization	Role Based Security	Both Authentication and Authorization	Underlying Core middleware
Architecture	Layered & Modular	Hierarchal	Hierarchal	Component based
Scalability	High	Limited	Limited	Extensible to underlying Middleware
Programming Environment	C, Java, Cog, Matlab, Perl, Corba, MPICH-G Python,	C#, OOP, threaded approach	C, MPI, JAVA	Parametric Language
Platform	Windows, Unix Like	Windows	Unix, Windows	Unix
Ease of Use	Low	High	Low	High
Scheduling Policy	Uses user level middleware	System Centric	System centric	Market driven
Application Type	Computational	HTC	Computational	Parametric
Implementation technology	C, java	C#, Perl	C, Java	-

Chapter 6: Conclusion and Scope of Future Work

6.1 Conclusion

This thesis provides the introduction to the grid computing. It emphasizes the importance of the middleware components, discusses the major grid middleware technologies and finally the grid environment has been realized using these middlewares. At last, the comparative analysis between these middlewares has been done on the basis of various factors.

This thesis provides, the in depth detail of the Grid Middlewares that act as an interface between the resources and the applications. This thesis provides the detailed explanation how the grid has been set up using the Globus middleware.

This thesis provides the architectural and philosophical differences, between these middleware technologies and thus educates the grid community about the choices available.

6.2 Scope of Future Work

All the above-discussed Grid middlewares are the build on the same idea to provide the end-uses an environment where they can utilize the unused power of there resources. At present the interoperatability between these technologies is little, thus the organizations using the different middlewares can't collaborate. Lots of efforts should be put on in the interoperatability so that further organizations can collaborate. The Grid bus broker is such an imitative which allows the collaboration of these underlying middlewares.

.

References

1. Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "*Alchemi: NET-based Grid Computing Framework and its Integration into Global Grids"* Technical Report, GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, December 2003. http://www.alchemi.net/files/alchemi_techreport.pdf
2. Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "*Alchemi: A .NET-Based Enterprise Grid Computing System*", Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), June 27-30, 2005, Las Vegas, USA.
   http://www.gridbus.org/%7Eraj/papers/alchemi_icomp05.pdf
3. B. Bansal, "*Design and Development of Grid Portal*", thesis Submitted at Computer Science & Engineering Deptt., TIET Patiala, May 2005
4. Bart Jacob, "*Grid computing: What are the key components?"* http://www-128.ibm.com/developerworks/Grid/library/gr-overview/
5. Clovis Chapman1, Paul Wilson2, "*Condor services for the Global Grid: Interoperability between Condor and OGSA*", Proceedings of the 2004 UK e-Science All Hands Meeting, ISBN 1-904425-21-6, pages 870-877, Nottingham, UK, August 2004
   http://www.cs.wisc.edu/condor/doc/condor-ogsa-2004.pdf
6. Douglas Thain, Todd Tannenbaum, and Miron Livny, "*Distributed Computing in Practice: The Condor Experience*" *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.
   http://www.cs.wisc.edu/condor/publications.html
7. Foster, I., Kesselman, C., Nick, J. and Tuecke, S**.**"The Anatomy of the Grid: Enabling Scalable Virtual Organizations" *International J. Supercomputer Applications*, 15(3), 2001.
   www.globus.org/alliance/publications/papers/**anatomy**.pdf
8. Foster, I., Kesselman, C., Nick, J. and Tuecke, S**. "**The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration**"**, Open grid service Infrastructure WG Global Grid Forum
   www.globus.org/research/papers.html

9. Foster, I., **"**What is the Grid? A Three Point Checklist," GRIDToday, July 20, 2002
www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf
10. Francois Grey, Matti Heikkurinen, Rosy Mondardini, Robindra Prabhu, "Brief History of Grid"
http://Gridcafe.web.cern.ch/Gridcafe/Gridhistory/history.html
11. Gregor von laszewaski, Ian Foster, Argonne National Laboratory, 9700 South Cass Avenue, Argonne IL 60439, U.S.A, Designing Grid Based Problem solving Environments
www-unix.mcs.anl.gov/~laszewsk/papers/cog-pse-final.pdf
12. Jay Haunger, Matt Haynos, "A visual tour of Open Grid Services Architecture",
http://www-128.ibm.com/developerworks/webservices/library/gr-visual/
13. Jean-Christophe Durand, "*Grid Computing: A Conceptual And Practical Study*" , University de Lausanne, Switzerland, November 2004
http://www.hec.unil.ch/cms_inforge/Durand.pdf
14. Jospeh, Joshy, Fellenstine Craig, "*Grid Computing*", Prentice Hall/IBM Press, Edition 2004 India
15. Livny and R. Raman. The GRID: Blueprint for a New Computing Infrastructure, chapter High Throughput Resource Management, pages 311–336. Morgan Kaufman, 1999.
16. Livny, M., Raman, R. and Solomon, M. *Matchmaking: Distributed Resource Management for High Throughput Computing*. Proc. of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, IL., July 28th-31st, 1998.
http://www.cs.wisc.edu/condor/doc/hpdc98.pdf
17. Mark Baker, Rajkumar Buyya, and Domenico Laforenza , " Grids and Grid technology for wide-area distributed computing", Software Practice and experience, 2002
http://www.Gridbus.org/papers/Gridtech.pdf
18. Rajkumar Buyya and Srikumar venugopal , "A Gentle *Introduction to Grid* Computing and Technologies"
http://www.buyya.com/papers/GridIntro-CSI2005.pdf
19. Rajkumar Buyya, "The Nimrod-G Resource Broker: an economic Based Grid Scheduler" http://www.buyya.com/thesis/Gridbroker.pdf

20. Reinefed Alexander , Position Paper, www.egrid.org/ec_initiatives/reinefeld.html
21. R.J.M.Boer, "Resource Management in the Condor System", Delft University of Technology, Netherlands and National Institute for Nuclear Physics and High Energy Physics, Netherlands, May-1996
22. Russell Lock, "*An introduction to the Globus toolkit",* http://www.comp.lancs.ac.uk/computing/research/cseg/projects/dirc/papers/An% 20 ntroduction%20to%20the%20Globus%20toolkit.doc
23. Thierry Prioi, "Grid Middleware", Advanced Grid Reseach Workshops through Europeon and Asian Co-operation,
24. Viktors Berstis, et al, "*Introduction to Grid Computing with Globus* ", http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf
25. Wolfgang Leister, Shahrzade Mazaher, et al, "Grid and related technologies", http://publications.nr.no/Grid-report.pdf
26. "Condor R Version 6.6.10 Manual", Condor Team, University of Wisconsin–Madison, January 31, 2006

27 http://www.cs.wisc.edu/condor/overview/

28 www.entropia.com/pdf/DCGridBrouchre0402.pdf

29 www.globus.org

30 www.wiki.gridpp.ac.uk/wiki/Grid_middleware

Papers Communicated/Published

1 Meena Gupta, Seema Bawa "Grid Computing: Evolution and standards ", National conference on Technologies of 21st Century (NCTC'06) organized by Srijan: A forum of Akhil Bhartiya Vidyarthi Parishad held at Punjab university, Chandigarh, India April 8-9, 2006. [Published in conference Proceedings].

2 Meena Gupta, Seema Bawa "Grid Middlewares: Comparative analysis", CoreGRID Workshop on Grid Middleware (GMW2006) organized by the CoreGRID Network of Excellence in conjunction with the European Conference on Parallel Computing Euro-Par 2006 I, Dresden, August 28-29, 2006. [Communicated]