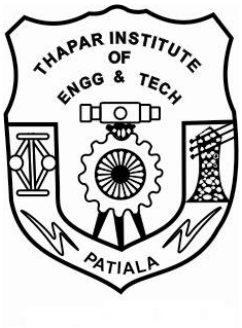
Quantifying Impacts of Resource Heterogeneity on Grid Performance

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering



By:
Amarjit Manjotra
(8043101)

Under the supervision of:
Ms. Inderveer Chana
(Senior Lecturer)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA – 147004

MAY 2006

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Quantifying the Impacts of Resource Heterogeneity on Grid Performance",** in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering at Computer Science & Engineering Department of Thapar Institute of Engineering & Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision and guidance of **Ms. Inderveer Chana, Senior Lecturer, CSED.**

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other University.

Amarjit Manjotra

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

(Ms. Inderveer Chana)
Senior Lecturer
Computer Science & Engineering Department
Thapar Institute of Engineering & Technology
Patiala-147004

Countersigned By:

(Dr. (Mrs.) Seema Bawa)
(Professor & Head)
Computer Science & Engineering. Department

Thapar Institute of Engg and Tech
Patiala-147004

(Dr. T.P. Singh)
(Dean)
Academic Affairs

Thapar Institute of Engg and Tech
Patiala-147004.

Acknowledgement

At the completion of my report, I take the opportunity to express my deep sense of gratitude to my supervisor and mentor Ms. Inderveer Chana, Senior lecturer, CSED for her invaluable guidance, constant encouragement, confidence, support, sharing with me her knowledge and experience and abiding inspiration. She was always available to me with her immense reservoir of vast learning.

I wish to express my deep gratitude to Dr. Seema Bawa, Professor and Head, Computer Science and Engineering Department for her guidance, suggestions and support throughout the Thesis work.

I am also thankful to Mr. Maninder Singh, Assistant Professor, Network Manager, Computer Science and Engineering Department for providing timely technical assistance and encouragement which went long way in successful completion of my thesis.

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

Amarjit Manjotra
Amarjit Manjotra

(8043101)

Abstract

Grid computing has evolved into an important discipline within the computer industry by differentiating itself from distributed computing through an increased focus on resource sharing, co-ordination, manageability and high performance. Grid computing combines open, shared, geographically distributed and heterogeneous resources to achieve high computational performance. The objective of the grid computing is to solve large problems which can not be solved by single CPU by achieving high computing performance by optimal use of geographically distributed heterogeneous idle resources. These resources may belong to different institutions, different domains, have different usage policies and pose different requirements on acceptable requests. However, the major challenges in such highly heterogeneous and complex computing environment are to design an efficient resource allocation and management infrastructure. So resource allocation strategies for such system should be smart, efficient, robust, and scalable.

Resources are heterogeneous due to differences in hardware components, differences in grid software environments, and different administration have different policies for sharing of resources. Therefore, resource heterogeneity, dynamic load on resources, task runtime prediction uncertainty, task-to-resource ratio and resource sharing in the grid environment affects application performance. In this thesis, we have focused on grid environments typically populated with large number of heterogeneous resources.

We have investigated and quantified the impacts of resource heterogeneity by executing grid application on homogeneous and heterogeneous resources. Results of our analysis show; when we increase resource heterogeneity in grid environment, the performance of the grid environment decreases as compared to homogeneous environment.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Introduction Chapter 1

In today's world of high speed computing, computers have become extremely powerful. Even home-based workstations are powerful enough for running complex applications. Taking into account that most of these machines are connected to a LAN or the Internet or even a WAN, companies came to the conclusion that rather than buying brand new machines, they could make use of unutilized resources available on those networks to solve their high demanding applications. These are the roots of what is called "Grid Computing".

Grid computing [1][2][3] has become popular in recent years; grid computing is the collection of open, shared, geographically distributed, and heterogeneous resources [4]. These resources are used to achieve high computational performance. These resources may belong to different institutions [5], have different usage policies and pose different requirements on acceptable requests.

So for managing these resources, resource management policies are required [6][7]. However, the major challenges in such highly heterogeneous and complex computing environment are to design an efficient resource allocation and management infrastructure. It has been shown that resource heterogeneity impacts the resource allocation in quite significant ways in terms of performance, reliability, robustness, scalability and fault tolerance. So resource allocation strategies for such system should be smart, efficient, robust, and scalable.

In this chapter we define the motivation of this research and then identify the research questions. After that, we present brief introduction of approach to answer these questions and finally give framework of this thesis.

1.1 Motivation

Grid environments are typically federations, in which subsets of the heterogeneous resources are under different local administration. The objective of the grid computing is to solve large problems which can not be solved by single CPU by achieving high computing performance by optimal use of geographically distributed heterogeneous idle

resources [8][9]. In large-scale open heterogeneous resources, shared grid environments, locating resources, selecting the best set of resources to run a distributed application [10], and mapping application components to these components are known to be difficult problems whose solutions are critical for the application performance [11][12].

Much research has been devoted to resource discovery and resource selection. Beyond large-scale, an added difficulty comes from the fact that the environment is shared with heterogeneous resource availabilities. Grid application performance is critical in grid computing environment so to achieve high performance we need to understand the factors that can affect the performance of an application.

1.2 The Problem Description

In open resource, shared grid environments, resources are subject to sharing among different users and applications. There are number of factors, which can affect the grid application performance like resource heterogeneity, dynamic load on resources, task runtime prediction uncertainty, task-to-resource ratio and resource sharing in the Grid environment. In this thesis, we have focused on open, shared grid environments typically populated with large numbers of heterogeneous resources. Resources are heterogeneous due to differences in hardware components (processors, storage devices, display terminal, motherboard etc) [13], differences in grid software environments (Globus Toolkit (GT), Sun Grid Engine, Alchemi, Legion, UNICORE etc), operating system (Windows, Linux, UNIX, Solaris, MacOS etc) and different administration have different policies for sharing of resources could limit the computing capabilities.

This work investigates how resource heterogeneity affects grid application performance and further it quantifies the impact of resource heterogeneity on grid application performance. This report will answer the following questions:

To what extent Hardware Heterogeneity affect grid application performance in grid environment?

To what extent Software Heterogeneity affect application performance in grid environment?

1.3 Approach

To quantify the affect of resource heterogeneity on grid application performance, we need to setup an experimental framework. So we have defined an experimental framework that allows us to examine the different factors affecting application performance in the open, shared and heterogeneous resources in grid environment. Within this framework we characterize the application and the resources. Our framework is composed of the application, the resource broker, and the computing environment. By following this experimental framework, we have installed all the required softwares at Centre of Excellence n Grid Computing, TIET, Patiala and build testbeds for experiments. We have chosen an application to run in this grid environment. Then finally we have executed the application in homogeneous as well as heterogeneous grid environment and tried to quantify the affects of resource heterogeneity. To allow more controllable and repeatable experiments, this framework is realized with a simulator that interprets real load data.

1.4 Organization of the Dissertation

The remainder of the thesis is organized as follows.

Chapter 2 discusses the literature survey of grid computing.

Chapter 3 discusses detail description of resource heterogeneity in grid computing.

Chapter 4 elaborates the problem statement.

Chapter 5 presents the experimental framework, Implementation and results in detail.

Chapter 6 discusses conclusion and suggestions for future research directions.

Grid computing systems are being positioned as the high performance computing infrastructure of the future that will be used to solve grand challenge problems and also provide the infrastructure for wide area distributed network computing. This chapter presents literature survey of grid computing. Focus of this chapter is on background and basics concepts of grid computing.

2.1 History

Grid computing [1] is the next major revolution in information technology after the advent of the Internet. Grid computing is an important and emerging new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. The concept of Grid Computing is come from the Power Grid (Electricity), in power grid the electricity comes from the various power stations and consumer just plug in and consumes the electricity by simply plug in and pay according to the consumption. Similarly, grid computing is collection of various distributed and heterogeneous shared resources, which are used to achieve high performance computing. The ancestor of the Grid is Metacomputing [4]. This term was coined in the early 1980s by NCSA Director, Larry Smarr. The idea of Metacomputing was to interconnect supercomputer centers in order to achieve superior processing resources. One of the first infrastructures in this area, named Information Wide Area Year (I-WAY) [4], was demonstrated at Supercomputing 1995. This project strongly influenced the subsequent Grid computing activities. In fact one of the researchers who lead the project I-WAY was Ian Foster who along with Carl Kesselman published in 1997 a paper that clearly links the Globus Toolkit, which is currently the heart of many Grid projects, to Metacomputing. The Foster-Kesselman duo organized in 1997, at Argonne National Laboratory, a workshop entitled “Building a Computational Grid”. At this moment the term “Grid” was born. The workshop was followed in 1998 by the publication of the book “The Grid: Blueprint for a New Computing Infrastructure” by Foster and Kesselman themselves. For these reasons they are not only to be considered the fathers of the Grid but their book, which in the

meantime was almost entirely rewritten and re-published in 2003, is also considered the "Grid Bible".

2.2 What is Grid?

Back in 1998, Carl Kesselman and Ian Foster attempted a definition in the book "The Grid: Blueprint for a New Computing Infrastructure." They wrote: "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." [1].

Dependable: Users need to be sure they will receive predictable and high levels of performance from components of the Grid.

Consistency: It is fundamental for the Grid to provide stable and regular services (even in presence of system heterogeneity).

Pervasive: Availability of services at any time, no matter what the environment might be.

Inexpensive: The aim of the Grid is to be broadened and thus it has to be inexpensive for members.

The combination of dependability, consistency and pervasiveness causes the Grid to have a computational effect on how computation is performed.

2.3 A Grid Checklist

Ian Foster proposed the three-point checklist [1] for determining whether a system is a Grid. He suggested that the essence of the definitions above can be captured in a simple checklist, according that a Grid is a system that:

1) Coordinates resources that are not subject to centralized control: A Grid integrates and coordinates resources and users that live within different control domains—for example, the user's desktop vs. central computing; different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings.

2) Using standard, open, general-purpose protocols and interfaces: A Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. It is important that these protocols and interfaces be *standard* and *open*.

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

2.4 Characteristics of Grid Computing

Important characteristics of grid computing are [14]:

Multiple Administrative Domains and Autonomy: Grid resources are geographically distributed across multiple administrative domains and owned by different organizations. The autonomy of resource owners needs to be honored along with their local resource management and usages policies.

Heterogeneity: A Grid involves a multiplicity of resources that are heterogeneous in nature and will encompass a vast range of technologies.

Scalability: A Grid might grow from a few integrated resources to millions. This raises the problem of potential performance degradation. Consequently, applications that require a large number of geographically located resources must be designed to be latency and bandwidth tolerant.

Dynamicity or Adaptability: In a Grid, resource failure is the rule rather than the exception. In fact, with so many resources in a Grid, the probability of some resource failing is high. Resource managers or applications must tailor their behavior dynamically and use the available resources and services efficiently and effectively.

2.5 Grid Components

Grid computing can be divided into six major components [13]. A detailed description of each of those is given in the subsequent sections and thus only a brief introduction of the reasons why they are considered as major is given here.

2.5.1 User Interface

Accessing information on the grid is fairly important, and the user interface [13] component handles this task for the user. It often comes in one of two ways:

An interface provided by an application that the user is running.

An interface provided by the grid administrator, much like a Web portal that provides access to the applications and resources available on the grid in a single virtual space.

2.5.2 Security

Computers on a grid are networked and running applications; they can also be handling sensitive or extremely valuable data, so the security component of grid computing is of crucial concern. This component includes elements such as encryption, authentication, and authorization. The use of a user interface as a portal is also important because it can help users to learn how to query the grid.

2.5.3 Workload Management

Applications that a user wants to run on a grid must be aware of the resources that are available; this is where a workload management service comes in handy. An application can communicate with the workload manager to discover the available resources and their status.

2.5.4 Scheduler

A scheduler is needed to locate the computers (resources) on which to run an application, and to assign the jobs required i.e. scheduler map the heterogeneous resources with application's tasks. This can be as simple as taking the next available resource, but often this task involves prioritizing job queues, managing the load, finding idle machines.

2.5.5 Data Management

If an application is running on a system that doesn't hold the data the application needs, a secure, reliable data management facility takes care of moving that data to the right place across various machines, encountering various protocols.

2.5.6 Resource Management

To handle such core tasks as launching jobs with specific resources, monitoring the status of those jobs, and retrieving results, a resource management facility is necessary. It's important to remember that grid computing doesn't operate in a vacuum–just the opposite. It potentially involves every protocol and computer technology in operation today.

2.6 Why Grid Computing?

Computers have been proven to be very efficient to solve complex scientific problems. They are used to model and simulate problems of a wide range of domains; for instance medicine, engineering, security control and many more. Although their computational capacity has shown greater capabilities than the human brain to solve such problems, computers are still used less than they could be. One of the most important reasons to this lack of use of computational power is that, despite the relatively powerful computing environment one can have, it is not adapted to such complicated computational purposes. The following are given the reasons for why we need grid computing [13]:

2.6.1 Exploit unused resources

In most organizations, computing resources are underutilized. Most desktop machines are busy less than 25% of the time (if we consider that a normal employee works 7 hours a day and that 42 hours a week and that there are 168 hours per week) and even the server machines can often be fairly idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usages. The easiest use of Grid computing would be to run an existing application on several machines. The machine on which the application is normally run might be unusually busy, the execution of the task would be delayed. Grid Computing should enable the job in question to be run on an idle machine elsewhere on the network [13].

2.6.2 Increase Computation

To provide users with more computational power, some crucial areas have to be considered:

2.6.2.1 Hardware Improvement

Microprocessor architecture and other resource capabilities of personal computers continuously increase to provide users with additional power.

2.6.2.2 Periodic Computational Needs

Some applications only need computational power once in a while. The systems are fully utilized at that times and idle the rest of the time.

2.6.2.3 Capacity of Idle Machines

Machines are often idle and thus their computational power is free to use. The idea would be to use it only during that idle time and leave once the computer is in use.

2.6.2.4 Sharing of Computational Results

The key to more sharing may be the development of collaboratories centers without walls, in which the nation's researchers can perform their research without regard to geographical location-interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries. Considering these areas of interest, communication networks in place could act as a medium to provide access to advanced computational capabilities, regardless of the location of resources.

2.7 Evolution of Grid Architecture

In 1998, it was stated that "a computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." [1] This definition was primarily centered on the computational aspects of grids. Later iterations broadened this definition with more focus on coordinated resource sharing and problem solving in multi-institutional virtual organizations.

2.7.1 The Grid Problem

Grid computing has evolved into an important discipline within the computer industry by differentiating itself from distributed computing through an increased focus on resource sharing, co-ordination, manageability, and high performance. The focus on resource sharing is called the grid problem [5], which can be defined as the set of problems associated with resource sharing among a set of individuals or groups. This sharing of resources, ranging from simple file transfers to complex and collaborative problem solving, is accomplished under controlled and well-defined conditions and policies. In this context, the critical problems are resource discovery, authentication, authorization,

and access mechanisms. Resource sharing is further complicated when a grid is introduced as a solution for utility computing, where commercial applications and resources become available as shareable and on-demand resources.

2.7.2 Virtual Organization

A *virtual organization* (VO) [5] is a dynamic group of individuals, groups, or organizations that define the conditions and rules for sharing resources. The concept of the VO is the key to grid computing. All VOs share some characteristics and issues, including common concerns and requirements that may vary in size, scope, duration, sociology, and structure. The members of any VO negotiate the sharing of resources based upon the rules and conditions defined by the VO, and the members then share the resources in the VO's constructed resource pool.

2.7.3 Grid Architecture

The establishment, management, and exploitation of dynamic, cross-organizational VO sharing relationships require new technology. This technology is structured in terms of a Grid architecture that identifies fundamental system components, specifies the purpose and function of these components, and indicates how these components interact with one another.

In defining Grid architecture, the effective VO operation requires that every user of grid be able to establish sharing relationships among any potential participants. Interoperability is thus the central issue to be addressed. In a networked environment, interoperability means common protocols. Hence, Grid architecture is protocol architecture, with protocols defining the basic mechanisms by which VO users and resources negotiate, establish, manage, and exploit sharing relationships. A standards-based open architecture facilitates extensibility, interoperability, portability, and code sharing; standard protocols make it easy to define standard services that provide enhanced capabilities.

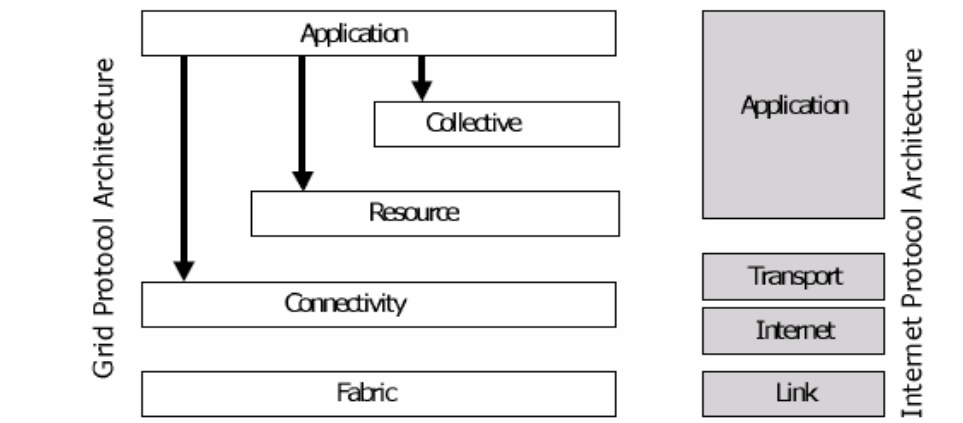


Figure 2.1: The layers of the grid architecture and its relationship to the Internet Protocol architecture [3]

The requirement of constructing of Application Programming Interfaces and Software Development Kits to provide the programming abstractions required to create a usable Grid. Together, this technology and architecture constitute what is often termed middleware. The goal of Grid architecture is not to provide a complete enumeration of all required protocols (and services, APIs, and SDKs) but rather to identify requirements for general classes of component. The result is an extensible, open architectural structure within which can be placed solutions to key VO requirements. Grid architecture organizes components into layers [5], as shown in Figure 2.1. Components within each layer share common characteristics but can build on capabilities and behaviors provided by any lower layer.

2.7.3.1 Fabric Layer: Interfaces to Local Resources

The fabric layer defines the interface to local resources, which may be shared. This includes computational resources, data storage, networks, catalogs, software modules, and other system resources [3].

2.7.3.2 Connectivity Layer: Communicating Easily and Securely

The connectivity layer defines the basic communication and authentication protocols required for grid-specific networking service transactions. Communication protocols enable the exchange of data between Fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources. Communication requirements include

transport, routing, and naming. While alternatives certainly exist, the assumption here is that these protocols are drawn from the TCP/IP protocol stack: specifically, the Internet (IP and ICMP), transport (TCP, UDP), and application (DNS, OSPF, RSVP, etc.) layers of the Internet layered protocol architecture. This is not to say that in the future, Grid communications will not demand new protocols that take into account particular types of network dynamics.

2.7.3.3 Resource Layer: Sharing Single Resources

Resource layer uses the communication and security protocols (defined by the connectivity layer) to control secure negotiation, initiation, monitoring, accounting, and payment for the sharing of functions of individual resources. The resource layer calls the fabric layer functions to access and control local resources. This layer only handles individual resources, ignoring global states and atomic actions across the resource collection pool, which are the responsibility of the collective layer.

Resource layer protocols can be distinguished in two primary classes:

Information protocols are used to obtain information about the structure and state of a resource, for example, its configuration, current load, and usages policy (e.g., cost).

Management protocols are used to negotiate access to a shared resource. Since management protocols are responsible for instantiating sharing relationships, they must serve as a "policy application point," ensuring that the requested protocol operations are consistent with the policy under which the resource is to be shared. Issues that must be considered include accounting and payment. A protocol may also support monitoring the status of an operation and controlling (for example, terminating) the operation.

2.7.3.4 Collective Layer: Coordinating Multiple Resources

While the resource layer manages an individual resource, the collective layer is responsible for all global resource management and interaction with collections of resources. This protocol layer implements a wide variety of sharing behaviors using a small number of resource-layer and connectivity-layer protocols.

2.7.3.5 Applications Layer

The application layer enables the use of resources in a grid environment through various collaboration and resource access protocols. The final layer in our Grid architecture comprises the user applications that operate within a VO environment. Figure 2.2 illustrates an application programmer's view of Grid architecture. Applications are constructed in terms of, and by calling upon, services defined at any layer. Each layer has well-defined protocols that provide access to some useful service: resource management, data access, resource discovery, and so on. At each layer, APIs may also be defined whose implementation (ideally provided by third-party SDKs) exchange protocol messages with the appropriate service(s) to perform desired actions.

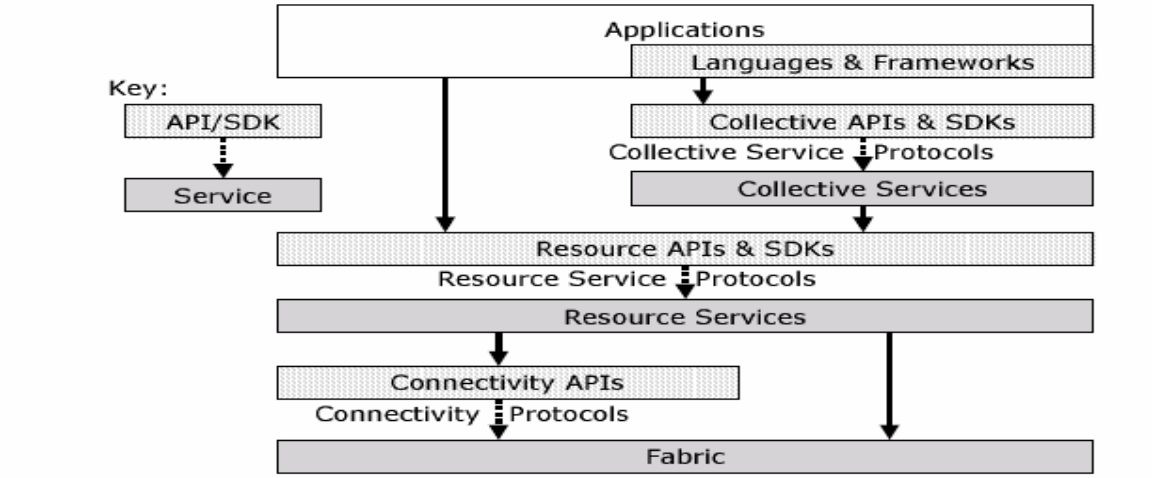


Figure 2.2: Illustrates an application programmer's view of Grid architecture [3]

The label "applications" show in a single layer in Figure 2.2 may in practice call upon sophisticated frameworks and libraries. These frameworks may themselves define protocols, services, and/or APIs.

2.7.4 Service-Oriented Architecture (SOA) Model

A service-oriented architecture (SOA) [5] is a specific type of distributed system framework, which maintains agents that act as "software services," performing well-defined operations. We define a SOA as a loosely coupled architecture with a set of abstractions relating to components granular enough for consumption by clients and accessible over the network with well-defined policies as dictated by the components.

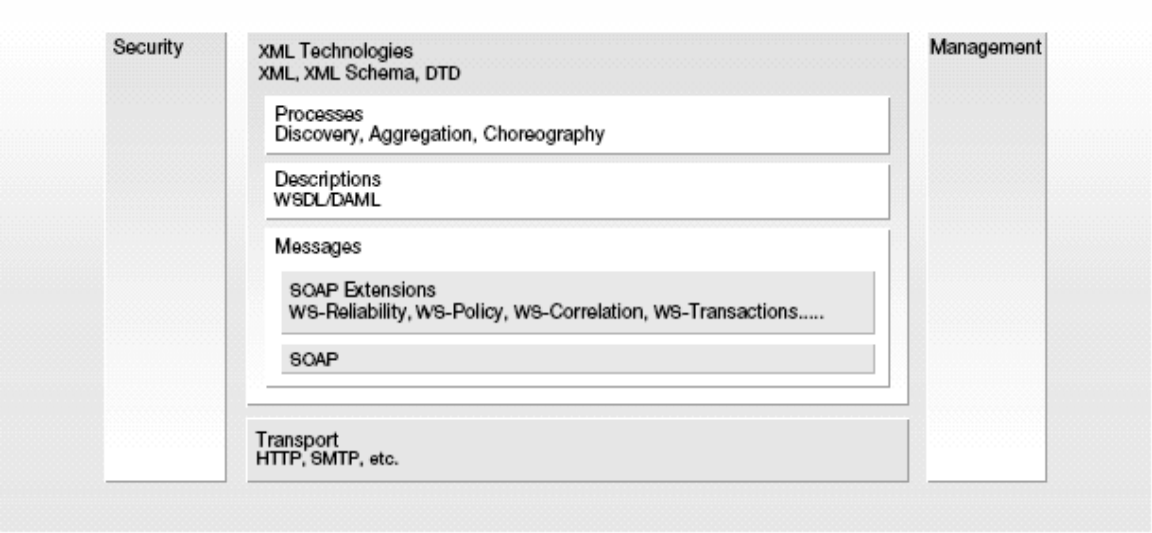


Figure 2.3: The Web Services Architecture (WSA) and the respective layers in the WSA framework [5]

Thus, a service acts (in some manner) as a user-facing software component of an application or resource. This paradigm of functionality enables the users of that application (or resource pool) to be concerned only with the operational description of the service. In addition, SOA stresses that all services have a network-addressable interface and communicate via standard protocols and data formats called messages.

The Web Services Architecture (WSA) helps to enable and define SOA, where services interact by exchanging XML (Extensible Markup Language) messages, as shown Figure 2.3.

The main characteristics of the WSA are:

It is based on XML technologies such as XML Information Model, XML Base, and XML Schema.

It is independent of the underlying transport protocols (e.g., HTTP [HyperText Transfer Protocol] or SMTP [Simple Mail Transfer Protocol]) and the transport selected as a runtime binding mechanism.

It uses XML message exchanges, while providing the extensibility model for this message exchange pattern to adapt to various message interchange requirements (e.g., security, reliability, correlation and privacy.) These interoperable messages could be created using Simple Object Access Protocol (SOAP) and SOAP extensions. SOAP

extension mechanisms and XML information models are used to construct Web services extensions.

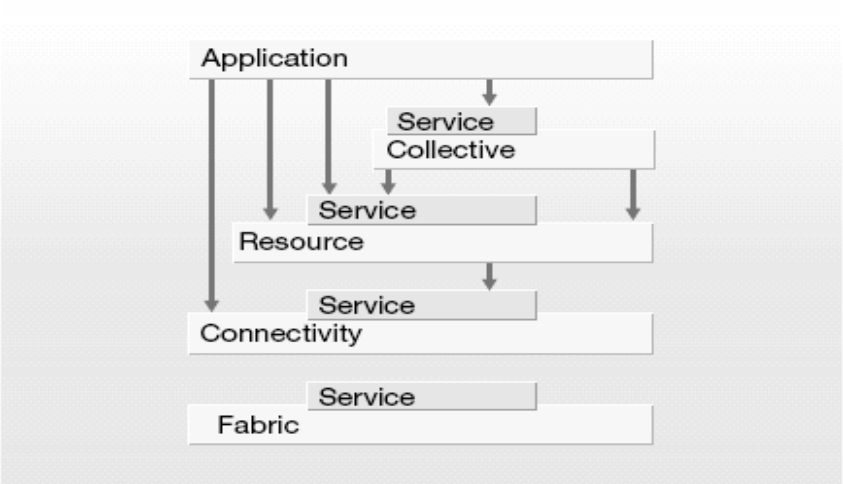


Figure 2.4: Service-oriented grid architecture with service interfaces [5]

Service capabilities are described using description languages such as Web Services Description Language (WSDL).

It uses the service discovery, workflow choreography, and transaction/state management that are built on the XML capabilities and lower-layer specifications in the architecture layer.

The emergence of the SOA concept helps grid resources to advertise their capabilities through standard interfaces defined as part of their service extensions. This enables grid users to integrate the resources through open-standards interfaces.

2.8 Standard Platform for Grid Computing

Foster et al. described the Open Grid Services Architecture (OGSA) [5] [6] as a solution to the open standardization problem. This architectural concept is a result of the alignment of existing grid standards with emerging SOAs, as well as with the Web. OGSA provides a uniform way to describe grid services and define a common pattern of behavior for these services. It defines grid-service behavior, service description mechanisms, and protocol binding information by using Web services as the technology enabler. This architecture uses the best features from both the grid-computing community and the Web-services community.

2.8.1 OGSA Architecture

OGSA is a layered architecture [5], as shown in Figure 2.5, with clear separation of the functionalities at each layer. As seen in the Figure 2.5, the core architecture layers are the Open Grid Services Infrastructure (OGSI) and OGSA platform services [5][6].

The platform services establish a set of standard services including policy, logging, service level management, and other networking services. High-level applications and services use these lower-layer platform core components to become a part of a resource-sharing grid.

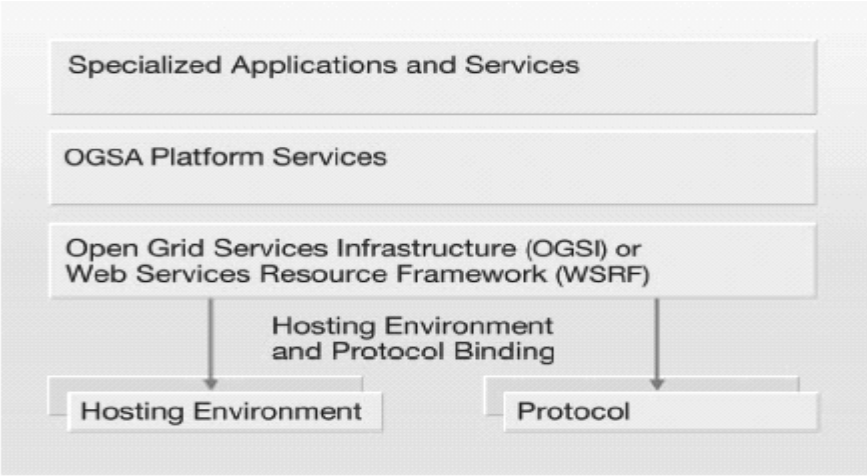


Figure 2.5: OGSA Platform Architecture [5]

The Global Grid Forum (GGF) has adopted the OGSA platform. There have been many activities in the GGF to define the use cases and core platform services, but there has been little activity related to the platform binding and resource modeling and profiling areas.

2.8.2 Management in OGSA

The following are list of main requirements for management in OGSA [6]. These requirements are especially important in a large-scale, distributed environment with no centralized notion of control, such as a Grid:

Scalability

Interoperability

Security

Reliability

Policy

Performance Monitoring

2.9 Grid Classification

It is now interesting to introduce the diverse uses of Grid Computing. The technology is not limited to computational power sharing or shared data storage (the better known applications), it can as well be used for joint services, beliefs and hardware sharing. On the usages bases we can classify the grid computing into three categories [13]:

Computational Grid

The possibility for Grid adherents to make use of the massive parallel CPU capacity available on the Grid is one of the most attractive features of the Grid. This feature is of particular interest for Research Centers that process on massive data (human genome, cosmic signals...). First, they have the opportunity to considerably gain in computing time; second, they could avoid money expenditures for brand new machines that would otherwise be required to process such data. This resource is provided by all 'hosts' connected to the Grid by the means of their personal hardware. Processors of each personal computer adhering to the System supplies any demanding member with his (her) computational power. These processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. Hence, special attention has to be given to which node the application should be run onto and how to efficiently use this node.

2.9.2 Data Grid

An increasing number of scientific applications generate a large volume of data. Researchers that need to access these terabytes of data are often distributed over the world and they often need to gain access to data located in different sites. To overcome these problems, we could make use of enormous unused disk drive capacity on the Grid. One of the concept that is to take into consideration when deploying the Grid is to facilitate the use of those underutilized and distributed storage capacity.

A grid that completely integrates data storage (i.e. total virtual space viewed as one single storage space) is often referred to as "data grid". For instance, if a batch job needs to read

a large amount of data, this data could be automatically replicated at various strategic points on the grid. Thus, if the job must be executed on a remote machine on the grid, the data is already there and does not need to be moved to it at every request. This offers clear performance benefits as well as security issues since copies of files and data can be saved on distant machines.

2.9.3 Service Grid

In this section we introduce several services that can be achieved with a Grid Architecture. We previously presented the two better known Grid utilizations but the subsequent paragraphs show that a Grid can as well be used with many other purposes [13].

2.9.3.1 Distributed Computing

The Grid is used to aggregate substantial computational resources to process applications that cannot be solved by a single machine. Applications are distributed over grid machines that can either be supercomputers in a country or personal computers within a company (the use depends on the size of the application to be run). In this case, the resource that is required is the CPU of each machine connected to the Grid. Every single workstation runs part(s) of the application locally and returns its results to a central scheduler that aggregates these messages. The scheduling phase is a challenging issue to enable Distributed Computing which needs to be carefully looked at.

2.9.3.2 On-demand Computing

Grid capabilities are used here to substitute requirements of resources that cannot be managed locally. This is a short-term alternative that supplies the node with data storage, computation, and software. In the case of Computing on Request, nodes share resources for a short period when one of the member asks for it, to reduce the cost of the process.

2.9.3.3 Collaborative Computing

A collaborative Computing application is an application that requires communication between humans. These applications often have a structured virtual space which is shared among members of the Grid. One such application could be an international project where Research Centers of distant countries would need to access identical data, run

procedures on supercomputers. In this case, talking about communication between humans, we understand that there is a need for real-time treatments imposed by human perceptual capabilities and a need to deal with the heterogeneity of the requirements as well.

2.9.3.4 Data Intensive Computing

The focus is here on synthesizing new information from data that is maintained in geographically distributed storage devices. This synthesis can be both computationally and communication intensive.

2.10 Summary

In this chapter, we have described multiple aspects of Grid Computing. Grid Computing is definitely a promising tendency to solve high demanding applications and all kinds of problems. There is a natural convergence of Grid services and web services. The Grid architecture and global standards serve a major role in determining the adoption rate of Grids in the commercial world. These standards are still evolving. This chapter contains a discussion of the issues of management that are specific to a Grid, and especially to OGSA. We defined the terms and describe the requirements of management as they relate to a Grid, and then discussed the individual interfaces, services, activities, etc. that are involved in Grid management, including both management within the Grid and the management of the Grid infrastructure.

In next chapter we will discuss resource heterogeneity and performance monitoring techniques.

"Computational Grids", comprising of ensembles of distributed computation sites, remote instruments, data archives, networks, and other resources, are becoming an increasingly prevalent and critical platform for high- performance computing. However application performance on a Grid is often difficult to achieve due to resource heterogeneity and variability in the deliverable performance of shared Grid components. In this chapter, we present literature survey of resource heterogeneity and performance monitoring tools.

3.1 Shared Resources

A grid is a collection of machines typically referred to as “nodes”, “resources”, “clients”, “hosts”, and other similar terms. These collections contribute any combination of resources to the grid as a whole and may have associated user-based access and usages restrictions. Resources can be categorized as follows [15]:

Computation: This refers to the computing cycles provided by the processors of the machines shared on the grid.

Storage: This refers to the quantity of storage, whether permanent or temporary, provided by each of the machines for grid use. The grid typically provides an integrated view of this storage with the goal of increasing the capacity, performance, reliability and sharing of data.

Communication: This refers to the data communication capacity (bandwidth), external communication access and redundant communication paths of a grid.

Software and licenses: A grid might have software with associated license restrictions installed on some of its nodes. This software can be treated as a resource and taken into consideration in grid job schedulers, especially if the license restricts the number of allowed installations.

Specialized devices: This refers to specialized devices, typically connected to machines shared on the grid, shared as standalone entities.

These resources are used to accomplish the grid applications.

3.2 Classification of Grid Application

Grid applications use above mentioned resources can be classified as follows [16]:

Loosely Coupled: Applications in this class are made up of "bags-of-tasks" with low memory requirements and small amounts of data for each task, and little communication required between tasks. Thus, they are compute intensive and are suitable for execution on wide-area clusters connected via low bandwidth/high latency networks, as demonstrated by the SETI@HOME project [17]. Many examples of such applications arise in the fields of bioinformatics and molecular biology.

Pipelined: Applications in this class digest streaming and/or real-time data. These algorithms are often very memory and data intensive (requiring the memories of more than one machine), and have coarse-grained inter-task communication, while the constituent tasks are highly parallel. Their data and memory requirements are more challenging than CLASS (1) applications, while their task-communication demands are somewhat more challenging. Typical examples in this class are the real-time signal processing and subsequent storage of data captured from satellites, remote sensors including microscopes etc. as in BIRN, MADCAP and ROADNET applications.

Tightly Synchronized: Applications in this class have frequent inter-task synchronization. They may also have significant computation, and memory/data usages. Thus they may be data-intensive and have the challenges of CLASS 2 but they place additional demands on the communications infrastructure due to their frequent inter-task communication. Examples of applications in this class are climate, physics, and molecular models employing explicit iterative methods.

Widely Distributed: Applications in this class search, update, and/or unify distributed databases. These typically have small compute, data, and memory requirements, but need to work seamlessly across the Grid environment to access databases that are variously owned and updated these are "data-related" and not always data-intensive.

3.3 Performance Issues

Key issues that need to be dealt with are heterogeneity, reliability, application composition, scheduling, resource management and security. In literature survey, the focus was on open resource, shared grid environments typically populated with large numbers of heterogeneous hosts with volatile CPU usages. There are many difficulties in running tightly synchronous grid applications in the open shared resource Grid environment, applications most commonly found in production grid environments. Applications such as the Encyclopedia of Life [11] or Folding@Home [12] are prime candidates for studying because these applications comprise of independent tasks that can be mapped to Grid resources (also known as scheduling) without synchronization between resources. Scheduling independent tasks onto Grid resources efficiently remain a challenge because the scheduler faces uncertainties in task runtime predictions and uncertainties in the resource volatility. In literature survey, we studied the factors [18] which can affect Grid application performance:

Dynamic Resource Load
Run-time Prediction uncertainty
Task-to-resource ratio
Resource sharing
Resource Heterogeneity

3.3.1 Dynamic Resource Load

In open resource, shared grid environments, resources are shared among different users and applications. Sharing of the resources invariably leads to fluctuations in resource load, which in turn affects application performance. Heavily loaded resources would lead naturally to poor application performance in spite of superior static resource attributes such as CPU speed or memory capacity. While much research has focused on selecting the best resources, dynamic resource information has strong influences on application as well.

In a Grid environment, dynamic resource data such as percent CPU usages per host or bandwidth between nodes as well as static resource attributes such as CPU speed and operating system using an Information Service such as the Globus Monitoring and Discovery System (MDS) [19]. Dynamic resource information is very important for

dynamic resource load in the grid environment and would be used by a resource broker in selecting resources for applications or by a scheduler to map tasks among different grid resources.

3.3.2 Run-time Prediction Uncertainty

Grid applications running on computational grids comprise independent tasks that can be scheduled on distributed resources. Tasks are scheduled on resources based on their predicted runtimes. To map application components to resources, any scheduler, whether or not using dynamic resource information, require knowledge of the runtimes of those components. Because the precise runtimes of the application components is most often not available or unknown prior to its execution, a runtime prediction model is required to supply the scheduler with the runtimes of the application components. As with any prediction model, the task runtime prediction model is subject to some variance: some tasks will finish ahead of its predicted time while some tasks will require more than its predicted time. Much research has focused on exploiting the predicted variance in the resource availability [20] or incorporated the task runtime prediction variance into the scheduling strategy.

3.3.3 Task-to-resource Ratio

In a grid environment, often the user has a choice in the number of resources to use as well as the number of tasks to use for the overall job and thus can decide on the task-to-resource ratio. Task-to-resource ratio effect application performance when numbers of tasks are more than available resources and also dynamic resource data affects application performance.

3.3.4 Resource Sharing

By definition, open resource, shared Grid environment are subject to sharing. Clearly more users or applications sharing the same resources would place a heavier load and lower application performance. What is not clear is whether an integrated scheduler that schedules all tasks for different applications is necessary for optimal application performance or whether a reservation system can provide better application performance than a best effort system.

3.3.5 Resource Heterogeneity

Grid computing is combination of heterogeneous resources like heterogeneity in operating systems, processors, network bandwidth, devices speed, scheduling policies etc. In large-scale open resource, shared grid environments, locating heterogeneous resources, selecting the best set of resources to run a distributed application, and mapping application components to these heterogeneous resources are known to be difficult problems whose solutions are critical for the application performance.

During thesis work, the focus is on resource heterogeneity. Grid is heterogeneous from several aspects [21]:

Data Representation: different data representation on different machines which requires data conversion either in MPI library or in the application.
Various processor speeds.
Difference in available memory requires the application to have a smart initial and load distribution and/or dynamic load balancing.
The differences on the communication characteristics for communication between different sites. In this latency and bandwidth get main focus.
Different operating systems
Different Grid software environments like Globus Toolkit (GT), Sun Grid Engine, Alchemi, Legion, UNICORE etc.

3.4 Classification of resource heterogeneity

During our thesis work we are concentrating only on resource heterogeneity. Heterogeneity of the resources can be classified in to three categories [13]:
Hardware heterogeneity
Software heterogeneity
Policy heterogeneity

3.4.1 Hardware Heterogeneity

Differences in hardware components among Grid nodes could limit the sharing of computing capabilities. During our study, we determined barriers due to hardware differences. Hardware heterogeneity can be classified as following:

3.4.1.1 Processor

The needs to achieve high performance and to solve large scale computational problems, distributed network computing environments have become a cost-effective and popular choice to attend such demand computations [22]. Unlike past supercomputers, a cluster or grid computing system can be used as multi-purpose computing platform, to run diverse high performance parallel applications.

In the case of computational grids, the processor typically is the component that can vary from one machine to another. Nowadays, most cluster systems are built using Intel or AMD CISC architecture computers, running a variant of open source operating systems [23]. Its gender (AMD or Intel), clock cycles and brand can be completely different and thus make job submission parameters vary from one to another. The heterogeneity of the processors can take different forms. The processors can be of different architectures. They may be of the same architecture but of different models. They may be of the same architecture and model but running different operating systems. They may be of the same architecture and model and running the same operating system but configured differently or using different basic software (compilers, run-time libraries, etc). All the differences can have an impact on performance and some other characteristics of the processors. An immediate implication from the heterogeneity of processors in a network of computers is that the processors run at different speeds. A good parallel application for a homogeneous distributed memory multiprocessor system tries to evenly distribute computations over available processors. This very distribution ensures the maximal speedup on the system consisting of identical processors. If the processors run at different speeds, faster processors will quickly perform their part of computations and begin waiting for slower ones at points of synchronization and data transfer. Therefore, the total time of computations will be determined by the time elapsed on the slowest processor. In other words, when executing parallel applications, which evenly distribute computations among available processors, the heterogeneous network will demonstrate the same

performance as a network of interconnected identical processors equivalent the slowest processor of the heterogeneous network of computers.

3.4.1.2 Communication Networks

In terms of parallel programming, the most demanding is a multi-user heterogeneous cluster made up of processors of different architectures interconnected via heterogeneous communication network. Cluster environment consists of PCs or workstations that are interconnected among themselves using high-speed networks, e.g., Gigabit Ethernet, SCI, Myrinet and Infiniband. The structure of the communication network reflects the evolution of the organization rather than its current snapshot. All the factors make the communication network of the grid extremely heterogeneous and irregular. Some communication links in this network may be of very low latency and/or low bandwidth and some are very fast. Therefore, a good parallel application for the heterogeneous network must distribute computations unevenly taking into account the difference in processor speed. The faster the processor is, the more computations it must perform. Ideally, the volume of computation performed by a processor should be proportional to its speed.

3.4.1.3 Storage Media

Now, if we consider Data Grids, we are interested in hard drives and volatile memory (fast memory). Thousands of different hardware combinations can be realized with these two sets. A hard drive can work in IDE or SCSI with a certain speed (rpm: rounds per minute) and volatile memory can as well vary in speed, in gender (DDRAM, RIM, SDRAM...) or in capacity (32, 64,128, 256, 512 or 1024 MB). We see that all these combinations are different and would execute data differently. Finally, if we consider Access Grids, display components are of concern.

3.4.1.4 Display Terminal

A display has a maximal resolution (XGA, SXGA or WXGA) and a maximal number of colors, a graphic card have specific memory capacity and features, and so do sound components. We clearly see that heterogeneous hardware bounds grid capabilities which emphasize the crucial phase of node selection.

3.4.2 Software Heterogeneity

Similarly, software diversity can be a problem for users to send jobs to distant machines. An application definitely needs to be executed by some specific software and requires that any machines that is willing to accomplish it possesses the right material. Software heterogeneity can be classified as following:

3.4.2.1 Operating System

Operating System plays very important role in grid computing. But in grid environment the nodes are operated on different operating systems. Some product might be executable the same way under Mac OS or UNIX but not under Windows. The architectural design of different operating systems effect the performance of grid applications. For example, Windows 2000 operating system architecture [24] is a hybrid architecture that is comprised of client/server, layered, object-oriented, and symmetric multiprocessing architecture principles. Although it uses objects to represent shared system resources, Windows 2000 is not strictly an object-oriented operating system. The bulk of the code is written in C for both tool availability and portability, and C does not directly support object-oriented constructs. Windows 2000 borrows from the features of object-oriented languages. Its features include 32-bit addressing, virtual memory support, system security and enhanced system integrity, platform independence, and built-in modular networking. Linux [25] gained in popularity because it has always been distributed as free software. Since the source code is readily available, users can freely change the kernel to suit their needs. However, it is important to understand how the Linux kernel has evolved and how it currently works before new system programs are written. A concrete architecture based on the Linux kernel source code can provide a reliable and up-to-date reference for Linux kernel hackers and developers. Linux is a Unix-compatible system. Most of the common UNIX tools and programs now run under Linux. Linux was originally developed to run on the Intel 80386 microprocessor. The original version was not readily portable to other platforms because it uses Intel's specific interrupt handling routines.

3.4.2.2 Grid Middlewares

Different Grid software environments like Globus Toolkit (GT), Sun Grid Engine, Alchemi, Condor, Legion, UNICORE etc are present middlewares; these middlewares are heterogeneous because they have different designs and written in different programming languages. In the case of Access Grid, all nodes ought to own the required software to communicate with each other. Different grid middlewares are explained as follow:

3.4.2.2.1 Globus: A Toolkit for Grid Computing

Globus provides a software infrastructure that enables applications to view distributed heterogeneous computing resources as a single virtual machine. The Globus project is an American multi-institutional research effort that seeks to enable the construction of computational Grids. A central element of the Globus system is the Globus Toolkit [26], which defines the basic services and capabilities required for constructing computational Grids. The toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, data management, resource reservation, and communications. The toolkit provides a bag of services from which developers of specific tools or applications can select, to meet their own particular needs. Globus is constructed as a layered architecture in which higher-level services can be developed using the lower level core services. Its emphasis is on the hierarchical integration of Grid components and their services. This feature encourages the usages of one or more lower level services in developing higher-level services. Resource and status information is provided via an LDAP-based network directory called Metacomputing Directory Services (MDS). MDS consists of two components, Grid Index Information Service (GIIS) and Grid Resource Information Service (GRIS). GRIS implements a uniform interface for querying resource providers on a Grid for their current configuration, capabilities, and status. GIIS pulls the information from multiple GRIS services and integrates it into a single coherent resource information database. The resource information providers use a push protocol to update GRIS. Thus MDS follows both push and pull protocols for resource dissemination. Higher-level tools such as resource brokers can perform resource discovery by querying MDS using LDAP protocols. The MDS namespace is organized hierarchically in the form of a tree structure. Globus offers QoS in the form of resource reservation.

3.4.2.2.2 Sun Grid Engine

Sun's Grid Engine is a Distributed Resource Management tool; it provides load management across heterogeneous, distributed computing environments. Sun made the source available under the Grid Engine project [28]. The Grid Engine project is an open source community effort, sponsored by Sun, to further distributed computing applications and services. The open source version of the software is known as Grid Engine. Grid Engine is generally installed across a cluster of machines using a shared file system. It can be configured to work across disparate file systems. Users submit jobs to Grid Engine and Grid Engine manages the allocation of jobs to machines within the cluster. This ensures the resources are used more productively therefore increasing availability. There are various components that make up a Sun Grid Engine cluster.

(A) Hosts: There are different types of host in a Grid Engine cluster:

Master Host: The master host acts as the core of this centralized system. The daemons qmaster and schedd run on the master host. There is only one master host.

Execution Host: An execution host is a node within the cluster that is capable of executing jobs. Each execution host runs the execution daemon execd.

Administration Host: Administration hosts are nodes that are given permission to alter the configuration of the Grid Engine cluster. By default the master host is also an administrative host.

Submit Host: Submit hosts are nodes within the cluster that users are allowed to submit jobs from to the Grid Engine. By default the master host is also a submit host.

Shadow Host: This node monitors the master host. If the master host fails, the shadow host takes over. If there is more than one shadow host an election protocol ensures only one takes over as master.

(B) Daemons: Daemons are processes that run background on nodes of the Grid Engine cluster.

Master Daemon: The qmaster daemon is central to the Grid Engine. It handles all requests to the Grid Engine Database and coordinates with the scheduler daemon and execution daemons.

Scheduler Daemon: The schedd is responsible for retrieving the current state of the cluster from qmaster and deciding which queue a job should be assigned to.

Execution Daemon: The execd is responsible for running jobs for the queue(s) configured for the execution host it runs on. It receives requests from qmaster, such as to run a job, and sends job reports and load reports back to the qmaster.

Shepherd Daemon: An individual shepherd is started by the execd as the parent process of each job. The shepherd starts the job and collects resource usages statistics once the job finishes.

Communication Daemon: The commd is used to separate communication from processing. All messages between daemons and/or hosts are sent via the commd.

Shadow Daemon: The shadowd runs on a shadow host. This daemon monitors the qmaster daemon. If the shadowd detects the qmaster has failed all running shadowds decide which is to take over as the new qmaster.

(C) Queues: Queue represents a class of jobs allowed to execute concurrently on an execution host. Jobs do not wait in a queue, once dispatched to a queue the job is associated with that queue and runs on that execution host. If the queue is suspended then so are any running jobs. The only place jobs wait are in the scheduler's job pending list. Queues are configured with various attributes, for example the number of processors. Job requirements have to fit within queue attributes for that queue to be considered as a suitable place for the job to run. If there is more than one suitable queue the scheduler will base its decision in a configurable manner: it can choose least loaded queue or based on queue sequence number.

(D) Complexes: Complexes are used to provide all the useful information about attributes of resources used within Grid Engine. They are also used as the framework for the Grid Engine's Consumable Resource facility, where attributes have an associated capacity that is monitored. They are used to describe queue, host and the global cluster attributes. The description includes name, shortcut, type, value, relation, requestable, consumable and default.

3.4.2.2.3 Alchemi

There is rapidly emerging interest in grid computing from commercial enterprises. A Microsoft Windows based grid computing infrastructure will play a critical role in the industry-wide adoption of grids due to the large-scale deployment of Windows within enterprises. This enables the harnessing of the unused computational power of desktop

PCs and workstations to create a virtual supercomputing resource at a fraction of the cost of traditional supercomputers. However, there is a distinct lack of service oriented architecture-based grid computing software in this space. To overcome this limitation, Windows-based grid computing framework called Alchemi [29] [30] [31] implemented on the Microsoft .NET Platform.

The Microsoft .NET Framework provides a powerful toolset that can be leveraged for all of these, in particular support for remote execution (via .NET Remoting and web services), multithreading, security, asynchronous programming, disconnected data access, managed execution and cross-language development, making it an ideal platform for grid computing middleware.

The Alchemi grid computing framework was conceived with the aim of making grid construction and development of grid software as easy as possible without sacrificing flexibility, scalability, reliability and extensibility. The key features supported by Alchemi are:

Internet-based clustering of desktop computers without a shared file system.

Federation of clusters to create hierarchical, cooperative grids.

Dedicated or non-dedicated (voluntary) execution by clusters and individual nodes.

Object-oriented grid thread programming model (fine-grained abstraction).

Web services interface supporting a grid job model (coarse-grained abstraction) for cross-platform interoperability e.g. for creating a global and cross-platform grid environment via a custom resource broker component.

The architecture of Alchemi follows the master-worker parallel programming paradigm 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 the 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 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.

The following are the components of Alchemi:

(A) Manager: 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.

(B) 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.

(C) 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.

(D) 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).

3.4.2.2.4 Condor: Cycle Stealing Technology for High Throughput Computing

Condor [31] [32] is a high-throughput computing environment developed at the University of Wisconsin at Madison, USA. It can manage a large collection of computers such as PCs, workstations, and clusters that are owned by different individuals. Although it is popularly known for harnessing idle computers CPU cycles (cycle stealing), it can be configured to share resources. The Condor environment follows a layered architecture and offers powerful and flexible resource management services for sequential and parallel applications. The Condor system pays special attention to the computer owner's rights and allocates their resources to the Condor pool as per the usages conditions defined by resource owners. Through its unique remote system call capabilities, Condor preserves the job's originating machine environment on the execution machine, even if the originating and execution machines do not share a common file system and/or user ID scheme. Condor jobs with a single process are automatically check-pointed and migrated between workstations as needed to ensure eventual completion. The Condor has been extended to support submission of jobs to resources Grid-enabled using Globus services [57]. Condor can have multiple Condor pools and each pool follows a flat machine organization. The Condor collector, which provides the resource information store, listens for advertisements of resource availability. A Condor resource agent runs on each machine periodically advertising its services to the collector. Customer agents advertise their requests for resources to the collector. The Condor matchmaker queries the collector for resource discovery that it uses to determine compatible resource requests and offers. The agents are then notified of their compatibility. The compatible agents then contact each other directly and, if they are satisfied, then the customer agent initiates computation on the resource. Resource requests and offers are described in the Condor classified advertisement (ClassAd) language. ClassAds use a semi-structured data model for resource description. Thus, no specific schema is required by the matchmaker allowing it to work naturally in a heterogeneous environment. The ClassAd language includes a query language as part of the data model, allowing advertising agents to specify their compatibility by including constraints in their resource offers and requests. The matchmaker performs scheduling in a Condor pool. The matchmaker is responsible for initiating contact between compatible agents. Customer agents may advertise resource

requests to multiple pools with a mechanism called flocking, allowing a computation to utilize resources distributed across different Condor pools.

The Condor system has recently been enhanced to support creation of personal condor pools. It allows the user to include their Globus-enabled nodes into the Condor pool to create a "personal condor" pool along with public condor pool nodes. The Grid nodes that are included in a personal condor pool are only accessible to the user who created the pool. Condor can be considered as a computational Grid with a flat organization. It uses an extensible schema with a hybrid namespace. It has no QoS support and the information store is a network directory that does not use X.500/LDAP technology. Resource discovery is achieved through centralized queries with periodic push dissemination. The scheduler is centralized.

3.4.3 Policy Heterogeneity

With the same approach as with the two previous classes, the policy can vary between members of a Grid. If no general policy is defined for Grid users, each of them can define his (her) own policy that is likely to differ from one node to another. At a higher level, organizations that are to share resources are expected to have there own policies and thus resource sharing is problematical. Grid computing requires collaborative resource sharing within a Virtual Organization (VO) and between different VOs. Resource providers and request submitters who participate within a VO share resources by defining how resource usages takes place in terms of where, what, who, and when it is allowed. Accordingly, we assume that policies may be represented in a three dimensional space consisting of resource provider (and property), request submitter, and time.

3.5 Increasing Order of Heterogeneity

Next we would like to classify the Grid systems in the increasing order of heterogeneity [22] and complexity and briefly characterize each heterogeneous system. The classes are:

- Clusters
- Organizational networks of computers (Intra grid)
- Global general purpose networks of computers (Inter grid)

3.5.1 Cluster Computing

Heterogeneous cluster is still a dedicated computer system designed mainly for high performance parallel computing, which is obtained from the classical homogeneous cluster architecture by relaxing one of its three key properties and leading to the situation when:

Processors in the cluster may not be identical.

The communication network may have a regular but heterogeneous structure (for example, it can consist of a number of faster communication segments interconnected by relatively slow links).

The cluster may be a multi-user computer system (but still dedicated to high performance parallel computing). As we have discussed, this, in particular, makes the performance characteristics of the processors dynamic and non-identical.

3.5.2 Organizational Networks of Computers (Intra grid)

Local network of computers (LNC) consists of diverse computers interconnected via mixed network equipment. By its nature, LNCs are multi-user computer systems. Therefore, just like highly heterogeneous clusters, LNCs consist of processors of different architectures, which can dynamically change their performance characteristics, interconnected via heterogeneous communication network. Unlike heterogeneous clusters, which are a parallel architecture designed mainly for high performance computing, LNCs are general-purpose computer systems typically associated with individual organizations. This impacts the heterogeneity of this architecture in several ways. First of all, the communication network of a typical LNC is not that regular and balanced as in heterogeneous clusters. The topology and structure of the communication network in such a LNC is determined by many different factors, among which high performance computing is far away from being a primary one if considered at all. The primary factors include the structure of the organization, the tasks that are solved on computers of the LNC, the security requirements, the construction restrictions, the budget limitations, the qualification of technical personnel, etc. An additional important factor is that the communication network is constantly developing rather than fixed once and forever. The development is normally occasional and incremental. Therefore, the structure of the communication network reflects the evolution of the organization rather than its current snapshot. All the factors make the communication network of the LNC

extremely heterogeneous and irregular. Some communication links in this network may be of very low latency and/or low bandwidth.

Secondly, different computers may have different functions in the LNC. Some computers can be relatively isolated. Some computers may provide services to other computers of the LNC. Some computers provide services to both local and external computers. This makes different computers have different levels of integration into the network. The heavier the integration is, the more dynamic and stochastic is the workload of the computer, and the less predictable become its performance characteristics. Another aspect of this functional heterogeneity is that a heavy server is normally configured differently compared to ordinary computers.

Thirdly, in general-purpose LNCs different components are not as strongly integrated and controlled as in heterogeneous clusters. LNCs are much less centralized computer system than heterogeneous clusters. They consist of relatively autonomous computers, each of which may be used and administered independently by its users. As a result, their configuration is much more dynamic than that of heterogeneous clusters. Computers can come and go just because their users switch them on and off, or re-boot them.

3.5.3 Grid Computing (Inter Grid)

Unlike a local networks of computers, all components of which are situated locally a global network of computers (GNC) includes computers that are geographically distributed. There are three main types of GNCs, which are briefly presented in the increasing order of their heterogeneity. The first type of GNCs is a dedicated system for high performance computing that consists of several interconnected homogeneous distributed memory multiprocessor systems or/and heterogeneous clusters. Apart from geographical distribution of its components, such a computer system is similar to heterogeneous clusters.

GNC of the second type is an organizational network. Such a network comprises geographically distributed computer resources of some individual organization, such as a company or university department. The organizational network can be seen as a geographically extended LNC. It is typically very well managed by a strong team of hardware and software experts. Its level of integration, centralization and uniformity is often even higher than that of LNCs. Therefore, apart from geographical distribution of

their components, organizational networks of computers are quite similar to local networks of computers.

Finally, GNC of the third type is a general-purpose global network of computers. Such a network consists of individual computers interconnected via Internet. Each of the computers is managed independently. This is the most heterogeneous, irregular, loosely integrated and dynamic type of heterogeneous network.

3.6 Performance Measurement Techniques

In grid computing, the performance of grid application is very important aspect to achieve. On the bases of performance, we can know weather grid computing environment provide desired results or not. The following are the performance measurement techniques used in grid environment:

Gloperf and Network Weather System (NWS): Network performance measurement tool for grid environment

In grid computing environments, network bandwidth discovery and allocation is a serious issue. When starting a grid application, how do users know which hosts have suitable bandwidth among them? In the absence of any type of bandwidth reservation mechanisms, how should applications initially configure and subsequently adapt themselves to prevailing network conditions? While running applications can monitor their own performance, they cannot readily measure performance in other parts of the environment. In light of these considerations, a tool for monitoring network performance in grid computing environments is needed that is part of the grid infrastructure itself. However, performance monitoring is not enough if the information is to be used to make resource allocation decisions. Instead, a scheduler (be it a human being or an automatic scheduling program) must use the performance data that has been collected to predict what bandwidth will be available when an application will consume it. Gloperf bandwidth information can be used to make these predictions.

To address these needs, Gloperf was developed as part of the Globus grid computing toolkit with consideration of all fundamental issues: Accuracy vs. Intrusiveness, Scalability, Portability, Fault Tolerance, Security, Measurement Policy, Data Discovery, Access and Usability.

Gloperf was designed for ease of deployment to enhance portability. It makes simple, end-to-end TCP measurements such that no special host permissions or specific knowledge of link topology is required. Besides measuring what applications are more likely to experience, this also enhances Gloperf's fault-tolerance. Data discovery and access is provided by storing Gloperf data in a directory service that uses a well-known naming schema. Scalability of such a tool is extremely important since grid environments have already been built that encompass thousands of nodes.

Gloperf was removed from Globus Toolkit when version 1.1.0 released. Now instead of Gloperf, Network Weather Service (NWS) [33] was introduced. NWS is far more advanced than Gloperf, both in its ability to do measurement of network (and other) loads, and its ability to do predictions based on those measurements. In addition, by using LDAP interface NWS produce network information, so that applications can get to the network information through the standard Globus MDS interfaces.

3.6.2 Grid Harvest Service (GHS): A Performance System of Grid Computing

Conventional performance evaluation mechanisms focus on dedicated distributed systems. Grid computing infrastructure, on another hand, is a shared collaborative environment constructed on autonomic virtual organizations. The non-dedicated characteristic of Grid computing prevents the leverage of conventional task scheduling systems. The conventional parallel processing scheduling methods cannot apply directly to a Grid environment where computing resources are autonomic shared. The key to Grid task scheduling is to understand the usages pattern and predict the availability of computing and communication resources, and to find their influence on the application performance. Some latest Grid tools, such as Network Weather Service (NWS), have been developed to meet the need. However, these tools are for short-term resource availability (generally in tens of seconds). Another solution is adapting resource reservation to reduce the complexity of resource management in a shared environment. This approach requires resource owners to have good planning on their own tasks and suffers in system utilization. This approach is useful for high priority tasks or to show the potential of Grid computing, it but has difficulties to be employed in a general enterprise environment. Also, resources reservation will be more effective if it is based on resource availability prediction.

For a long-term, application-level performance prediction and task scheduling system, the Grid Harvest Service (GHS) [34] system is developed for Grid computing. The "long-term" signifies that the system is designed for large applications that need hours of computations and is in contrast to the current Grid performance systems such as NWS.
The Grid Harvest Service system comprises of five primary subsystems: Performance evaluation, Performance measurement, Task allocation, Task scheduling, Execution management. Coordinately, they provide appropriate services to harvest Grid computing.

3.6.3 Performance Contracts: Predicting and Monitoring Grid Application Behavior

Given the dynamic nature of computational grids, both the applications running on the grid and the underlying infrastructure must be able to adapt to changing resource availability to sustain a predictable and desired level of application performance.
Performance Contract [35] introduce to predicting application performance and enabling the detection of unexpected execution behavior, typically caused by unanticipated load on shared grid resources. In which resource, application, and performance commitments are specified.
An application runs on a set of heterogeneous resources (e.g., processors and network links), each of which have certain capabilities (e.g., maximum floating point rate or available bandwidth). The application is executed with certain problem parameters (e.g., matrix size or image resolution), and it achieves some measurable and desired performance (e.g., render r frames per second or finish iteration i in t seconds) during its execution. A performance contract states that given a set of resources with certain capabilities, for particular problem parameters, the application will exhibit a specified, measurable performance. The term contract specifications to refer collectively to the resources, capabilities, problem parameters, and performance. A performance model generates a performance prediction for an application based on specified resources, capabilities, and problem parameters. To verify a contract, one must continually monitor the application and system during runtime to verify that the contract specifications are being met. The contract can be violated both by the application and the system, e.g. if the system resources don't deliver the expected performance.

3.10 Summary

This chapter presented literature survey of heterogeneous resources as Grid computing is collection of open, shared, geographically distributed and highly heterogeneous resources. Every thing used in grid computing is resources like CPU cycle, memory, network bandwidth, softwares and special devices etc. In the open, shared resource Grid environments, applications face number of challenges, resource heterogeneity is one of the most important challenges. Resource heterogeneity classified into three categories as hardware heterogeneity, software heterogeneity and policy heterogeneity. In hardware heterogeneity we discussed difference in hardware components like processors, RAM, Hard disk, display terminal etc. In software heterogeneity we discussed different operating systems, different middlewares like Globus toolkit, Alchemi, Sun grid engine and Condor. Resources heterogeneity effect the grid application performance. Finally we discussed various performance measurement techniques and tools like Gloperf and Network Weather System (NWS) are used to measure short-term network characteristics (network latency and bandwidth), Harvest Service (GHS) system is developed for a long-term, application-level performance prediction and task scheduling system. Performance Contracts is used to predict and Monitor Grid Application Behavior.

In next chapter we are going to discuss the description of thesis problem and motivation.

Problem Description Chapter 4

Grid computing enables sharing, selection and aggregation of large collections of geographically and organizationally distributed heterogeneous resources to increase computational, and storage power, resource accessibility and utilization for solving large-scale data intensive problems in science, engineering and commerce. But performance of application in such environment remains challenge. This chapter presents detail description of thesis problems and approach.

4.1 Problem Statement

In grid environments, the shared resources are heterogeneous in nature, which in turn affects application performance. Different CPU speed, different architecture of operating systems, data representation on different machines, different sharing policies and communication speed of different communication mediums i.e. latency and bandwidth. Heterogeneity in resources would lead naturally to affect application performance.

The focus of our study is to consider factors affecting application performance in the grid environment. Resource heterogeneity is the one of the most important factor which can affect the performance of the grid application. We classify the resource heterogeneity in three categories as hardware heterogeneity, software heterogeneity and policy heterogeneity. The main objective of thesis is to "*Quantifying the impacts of resource heterogeneity on grid application performance*". As a result the following questions have been addressed in this thesis:

To what extent Hardware Heterogeneity affect grid application performance in grid environment?

To what extent Software Heterogeneity affect application performance in grid environment?

4.1.1 Quantify the Impact of Hardware Heterogeneity

Grid computing is collection of different hardware components. Differences in hardware components in grid environment affect the grid application performance. We will be trying to quantify the affects of hardware heterogeneity on application performance by giving the answer to the following questions:

To what extent processors from different vendors, with different speed, design and architecture affect the application performance?

To what extent storage media (RAM and hard disk etc) from different vendors, with different storage capacity, affect the application performance?

4.1.2 Quantify the Impact of Software Heterogeneity

In grid environment, software diversity can be a problem for users to send jobs to distant machines because grid environment is built up from heterogeneous softwares (Operating system, Grid Middleware etc). Grid softwares and operating systems play very important role in performance of grid computing. But software heterogeneity affects grid application performance. So we will be trying to quantify the affects of software heterogeneity on application performance by giving the answer of the following questions:

To what extent operating systems from different vendors, with different design and architecture affect the application performance?

To what extent the different grid middlewares (Alchemi, Globus Toolkit, Condor, and Sun Grid Engine etc) affect the application performance?

As we already mentioned, the objective of the grid computing concept is to achieve high performance computing by optimal use of geographically distributed, shared and heterogeneous ideal resources on internet. But the performance of the grid application degraded due to heterogeneity in resources. So there is need to know, up to what extent resources heterogeneity affect the application performance. So during thesis work we will try to quantify the affect of resource heterogeneity on grid application performance.

4.2 Approach

To quantify the affect of resource heterogeneity on grid application performance, we need to setup an experimental framework. So we have defined an experimental framework that allows us to examine the different factors affecting application performance in the open, shared and heterogeneous resources in grid environment. Within this framework we characterize the application and the resources. Our framework is composed of the application, the resource broker (Grid bus broker, Nimrod), and the computing environment. Grid application is combination of independent tasks or jobs.

These tasks are executed as different thread on different nodes of grid. The resource broker is optional component of our framework. The computing environment contains various grid middlewares, CPU, operating systems and Storage media (RAM/Hard Disk).

By following this experimental framework, we have installed all the required softwares at Centre of Excellence n Grid Computing, TIET, Patiala and build testbeds for experiments. We have chosen an application (prime number generator) to run in this grid environment. Then finally we have executed the application in homogeneous as well as heterogeneous grid environment and tried to quantify the affects of resource heterogeneity. To allow more controllable and repeatable experiments, this framework is realized with a simulator that interprets real load data.

4.3 Summary

In this chapter we have discussed detail description of the thesis problem and approach to handle this problem.

In next chapter we are going to develop methodology to answer these problems.

This chapter discusses the implementation, grid environment details and experimental framework. It presents a scenario where an application executed in heterogeneous grid environment demonstrates up to what extent application performance affected by resources heterogeneity.

5.1 Setup of Grid Environment

Carl Kesselman and Ian Foster wrote "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." For setting grid environment we need hardware as well as software. In hardware we need computer systems, LAN cards, network wires, printers and more devices and in software we require operating systems, database packages, grid middleware packages and many more packages.

The first step to solve our problems (described in chapter 4) is setup heterogeneous grid environment. For setting up heterogeneous grid environment we need hardware as well as softwares as per following details:

5.1.1 Hardware Requirements

We are focusing on resource heterogeneity so we need heterogeneous hardware components in our grid environment like different processors (processors from different vendors, design, architecture and different processing speed), different capacity of Storage media (RAM and Hard disk), and different capacity of communication medium (different latency and bandwidth).

5.1.2 Software Requirements

We need following software to setup grid environment:

Operating system: Windows 2000, XP, Linux, Solaris.

Grid Middlewares: Alchemi, Globus toolkit, Sun Grid Engine, Condor etc Installation processes of Alchemi and GT4 are present in appendix A, B respectively.

Resource brokers: GridBus broker, Nimrod etc (optional), installation process of GridBus broker is present in appendix C.

5.2 Experimental Framework

After installation we need experimental framework for examining different factors that affects the application performance. Key elements of the framework include the application, GridBus broker, grid middlewares, operating system, CPU, memory (main, secondary memory), data representation, and display terminal. For the application, we consider compute intensive ones comprising independent tasks. In our experimental framework the software heterogeneity represented as heterogeneous grid middlewares like Alchemi, Globus Toolkit, Condor and Sun Grid Engine etc, and heterogeneous

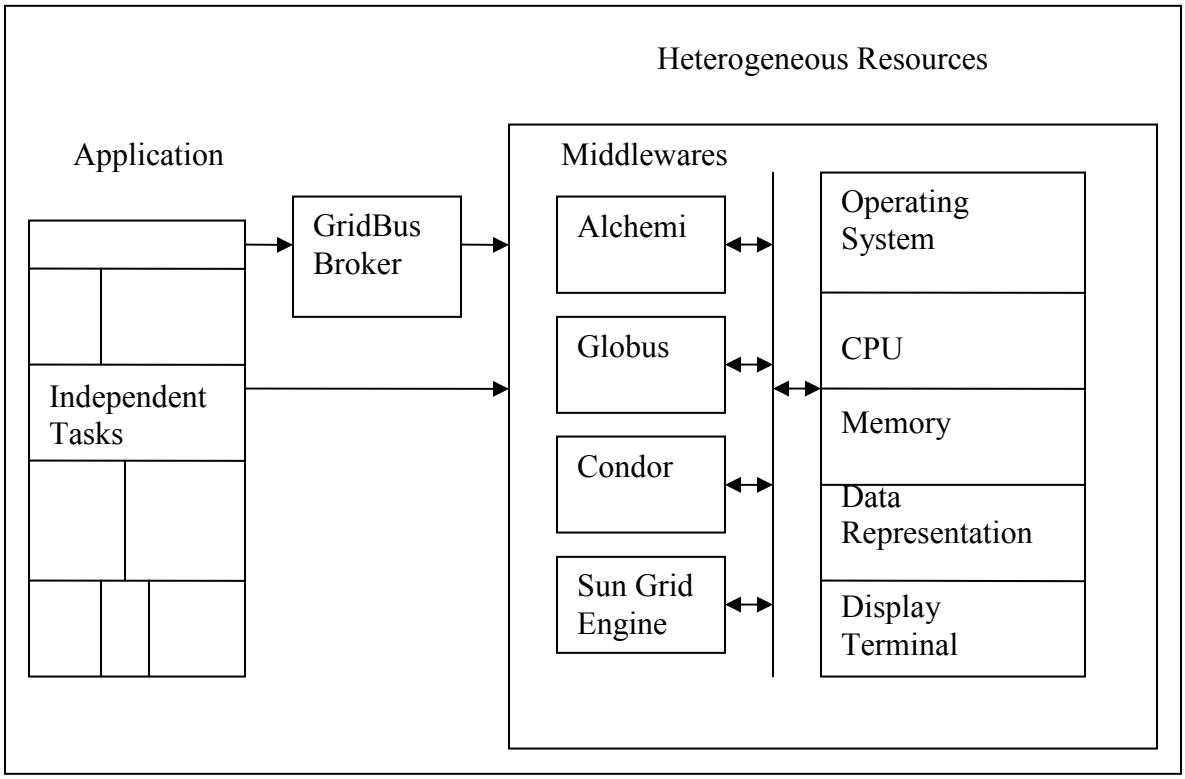


Figure 5.1: Experimental framework

Operating Systems like Windows 2000, XP, Linux, Solaris and CPU (processors) like Pentium, Celeron and AMD etc and storage media like RAM, Hard Disks etc. GridBus broker is optional component of our experimental framework. The GridBus Broker [36] is a software component which matches users' job requirements to the available resources and schedules the job to fulfill those requirements. The Broker therefore, takes care of resource discovery, job scheduling, execution, monitoring and gathering of output. It provides an abstraction to users particularly those who are not familiar with the complexity of Grid environments.

5.3 Application for Experiment

We have used Prime number generator application for our experiment. For execution of this application we have to give range of the numbers, from this range it selects ten random numbers and runs ten threads corresponding to those numbers on member nodes of grid and check weather those numbers are prime or not and count the present prime numbers.

5.4 Methodology and Results

We started our experiment from homogeneous grid environment and slowly increased the heterogeneity in resources.

5.4.1 Testbed for Homogeneous Grid Environment and Results

For homogeneous grid environment we have chosen three homogeneous nodes research3, research4 and tiet-goq0zl1zs0 and each node has following configurations:

Node Name	Processor	RAM/Hard Disk	Operating System	Grid Middleware
research3	Intel Pentium 4 3.00 GHz,	512 MB/ 80 GB	Windows 2000	Alchemi
research4	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Windows 2000	Alchemi
tiet-goq0zl1zs0	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Windows 2000	Alchemi

Table 5.1: Homogeneous grid environment configurations

Results: We started experiment by executing Prime Number Generator application on each node separately and all of the results presented in table 5.2 represent the average of three separate runs. Then we added one more homogeneous node and executed the same application, again the results are average of three separate runs. Again one more homogeneous node added and executed the same application and results are average of three separate runs.

The results are shown in following tables and figures:

No. of Node	Name of Node(s)	Average Execution Time in Seconds
1	Tiet-goq0zlzs0/ reseach3/reseach4	4.4843750
2	Tiet-goq0zlzs0, research3/ Tiet-goq0zlzs0, research4/ reseach3, reseach4	3.3593750
3	Tiet-goq0zlzs0, research3, Research4	2.8281250

Table 5.2: Results of homogeneous grid environment

When we run an application on single node then it takes maximum time to complete application and maximum power (processing speed) usages. A power usage by single node to complete an application is approximately 85% as shown in figure 5.2.

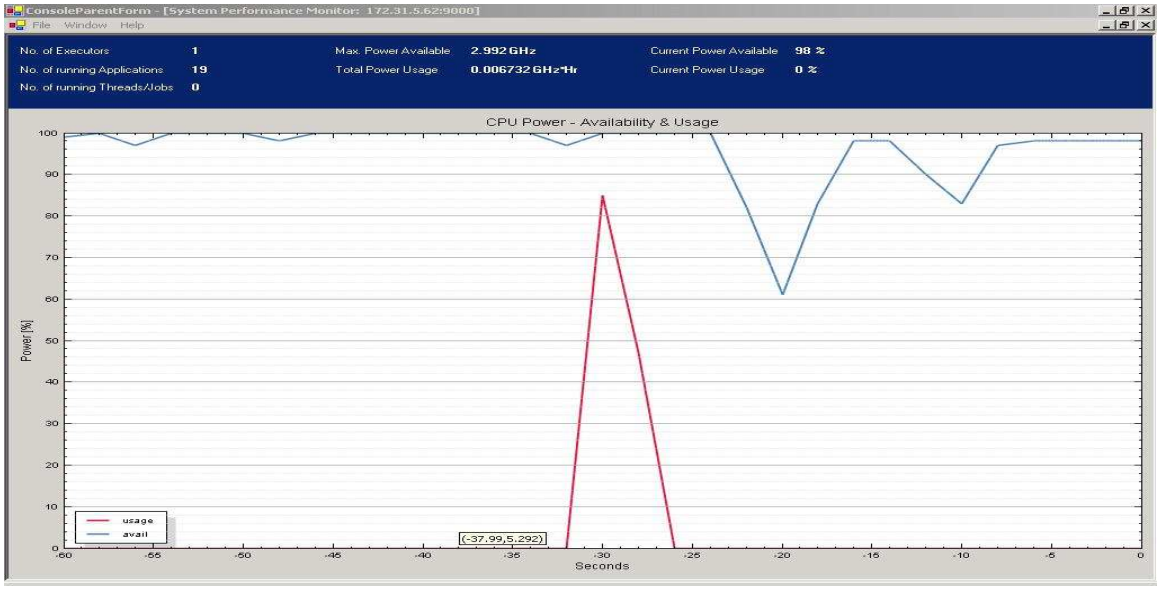


Figure 5.2: Power usages by single node

When we run an application on two nodes the power usages is approximately half as compare to the power usages by single node to complete same application. . A power usage by two nodes to complete an application is approximately 42% as shown in figure 5.3.

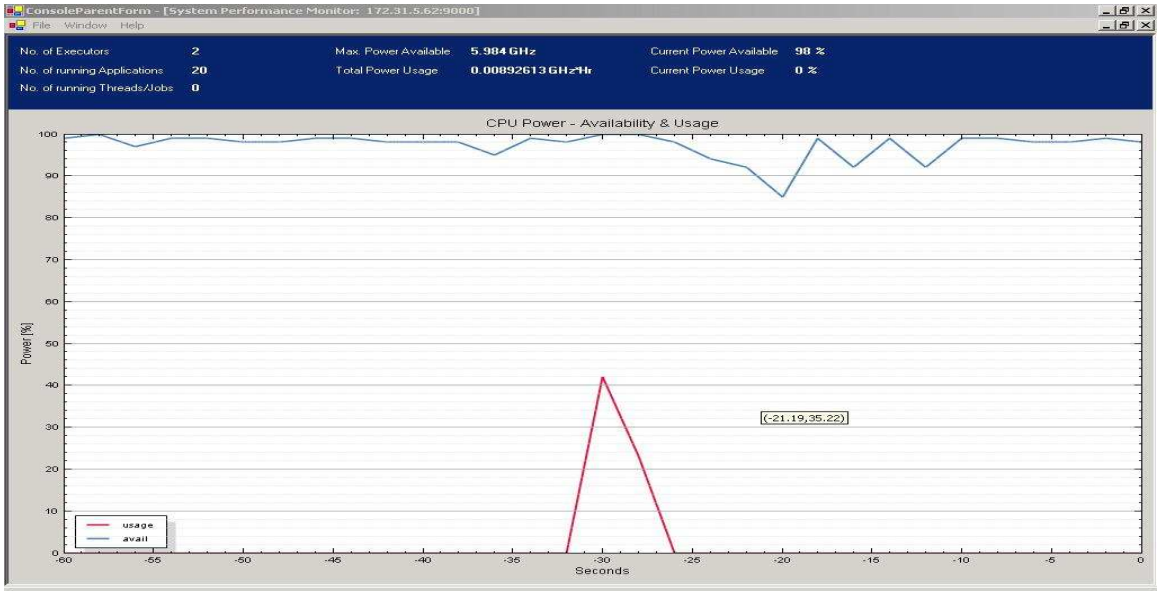


Figure 5.3: Power usages by two homogeneous nodes

When we run an application on three nodes the power usages is approximately one fourth as compare to the power usages by single node to complete same application. . A power usage by three nodes to complete an application is approximately 22% as shown in figure 5.4.

Figure 5.4: Power usages by three nodes

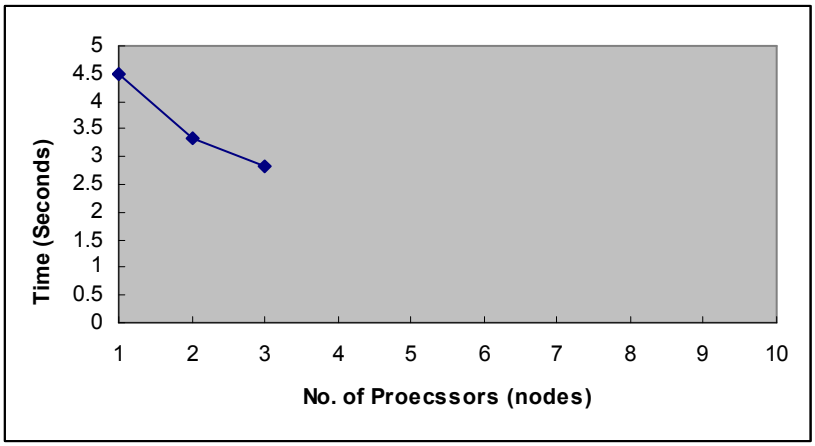


Figure 5.5: Performance evaluation of homogeneous grid environment

The above results are concluded in figure 5.5 as by increasing number of homogeneous resources, the time taken to complete the application decreases and performance of the grid increases.

5.4.2 Testbed for Heterogeneous (processors and RAM) Grid Environment and Results

Next we introduced heterogeneity in grid environment by using heterogeneous processors and RAM and rest of the components are homogeneous. Here we introduced two heterogeneous nodes namely "grid" and "software". "grid" node has Intel Pentium 4 with 1.4 GHz speed and 128MB RAM. "software" node has Intel Pentium 4 with 2.80 GHz speed and 256 MB RAM. The configurations of these heterogeneous resources are given in table 5.3.

Node Name	**Processor**	**RAM/Hard Disk**	**Operating System**	**Grid Middleware**
Tiet-goq0zlzs0	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Windows 2000	Alchemi
grid	Intel Pentium 4 1.4 GHz	128 MB/ 80 GB	Windows 2000	Alchemi
software	Intel Pentium 4 2.80 GHz	256 MB/ 80 GB	Windows 2000	Alchemi

Table 5.3: Heterogeneous hardware (processors, RAM) configurations

Results: We started experiment by executing Prime Number Generator application on each node separately and all of the results presented in table represent the average of three separate runs. Then we combined two heterogeneous nodes (grid, software) and executed the same application, again the results are average of three separate runs. Then we executed same application on three nodes (Tiet-goq0zlzs0, grid, and software) and results are average of three separate runs. The results are shown in table 5.4:

No. of Node	Name of Node(s)	Average Execution Time in Seconds
1	Tiet-goq0zlzs0	4.4843750
1	grid	8.3125000
1	software	4.6595035
2	grid, software	4.2598030
3	tiet-goq0zlzs0, grid, software	3.2954000

Table 5.4: Results of Heterogeneous hardware (processors, RAM) configurations in grid environment

Power usages by a node (grid) to complete an application is approximately 90% as shown in figure 5.6 and it took maximum time (8.3125000 seconds) to complete the application because speed of the processor and size of the RAM is half of other nodes.

Figure 5.6: Power usages by "grid" nodes

When we have executed an application on two nodes (grid and software) the power usages is approximately half as compare to the power usages by single node to complete same application. . A power usage by two nodes to complete an application is approximately 45% as shown in figure 5.7.

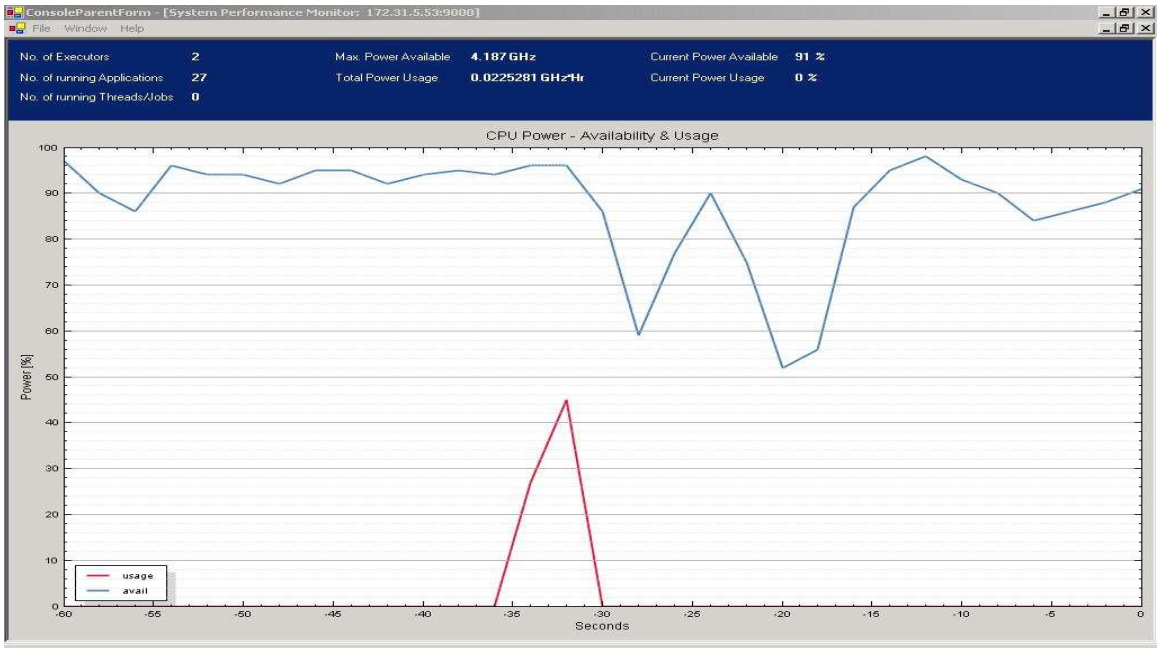


Figure 5.7: Power usages by "grid and software" node

A power usage by three nodes (tiet-goq0zlzs0, grid, software) to complete the same application is approximately 22% as shown in figure 5.8.

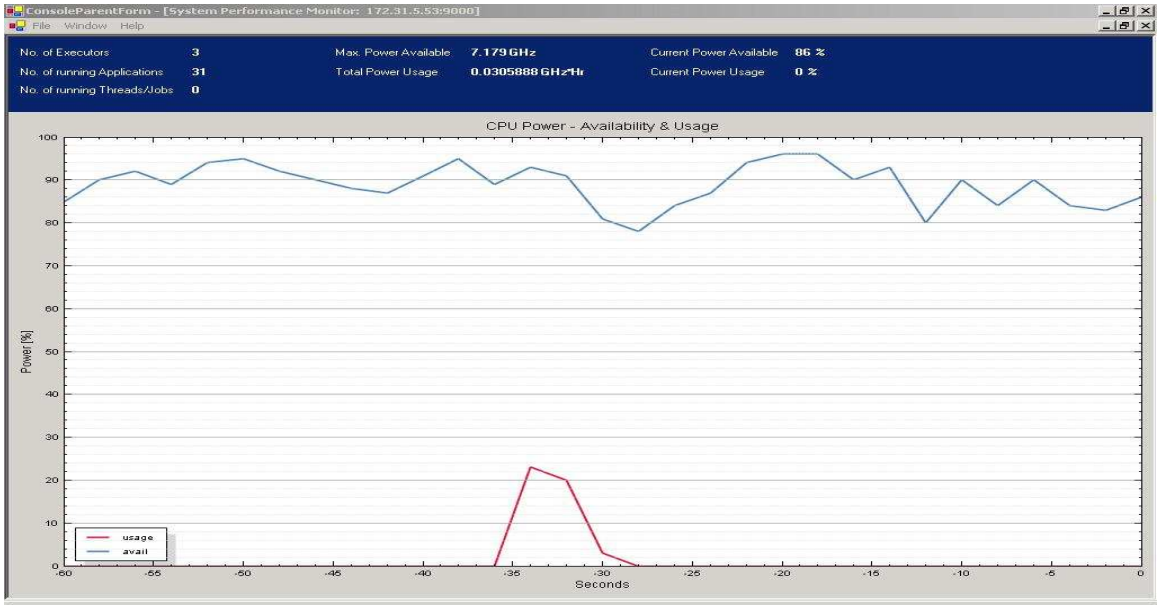


Figure 5.8: Power usages by "tiet-goq0zlzs0, grid, software" nodes

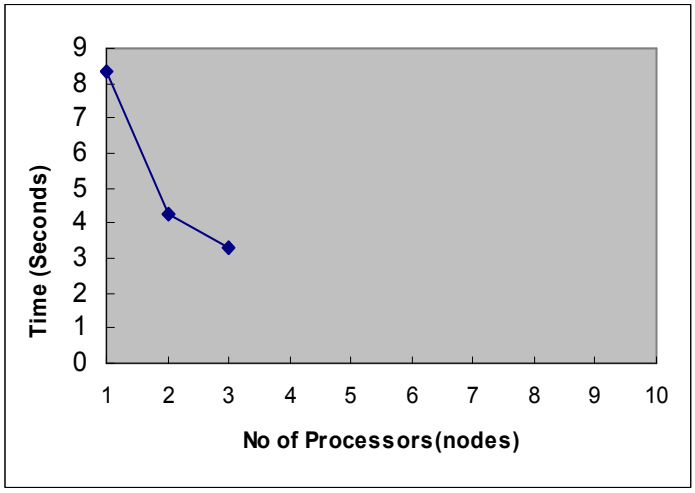


Figure 5.9: Performance evaluation of heterogeneous processors grid environment

We can analyze from table 5.2 and table 5.4, figure 5.5 and figure 5.9, the heterogeneous processors and RAM very much affect the grid application performance.

5.4.3 Testbed for Heterogeneous Operating System in Grid Environment and Results

Run application on heterogeneous operating systems and rest of the components are homogeneous. For this experiment we used two nodes "cex" and "Ram123" and both were working on Windows XP platform, and third node was "tiet-goq0zl1zs0" and working on Windows 2000 platform. Rest of the configuration was same as shown below:

Node Name	Processor	RAM/Hard Disk	Operating System	Grid Middleware
tiet-goq0zl1zs0	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Windows 2000	Alchemi
cex	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Windows XP	Alchemi
Ram123	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Windows XP	Alchemi

Table 5.5: Heterogeneous Operating System in grid environment

Results: We started this experiment by executing Prime Number Generator application on each node separately and all of the results presented in table represent the average of

three separate runs. Then we combined two Windows XP nodes and executed the same application. Then we combined one Windows 2000 nodes and one Windows XP node and executed the same application. Finally, we combined all three nodes. The results are shown in table 5.6:

No. of Node	Name of Node(s)	Average Execution Time in Seconds
1	cex / Ram123	3.3281250
1	tiet-goq0zlzs0	4.4843750
2	cex , Ram123	2.3281250
2	cex, tiet-goq0zlzs0	3.2784320
3	tiet-goq0zlzs0, cex, Ram123	2.2881250

Table 5.6: Results of Heterogeneous Operating System in grid environment

When we run an application on single Windows XP node then it took less time as compare to single Windows 2000 node. Results show, performance of grid improved with Windows XP. A power usage by single Windows XP node to complete an application is approximately 86% as shown in figure 5.10.

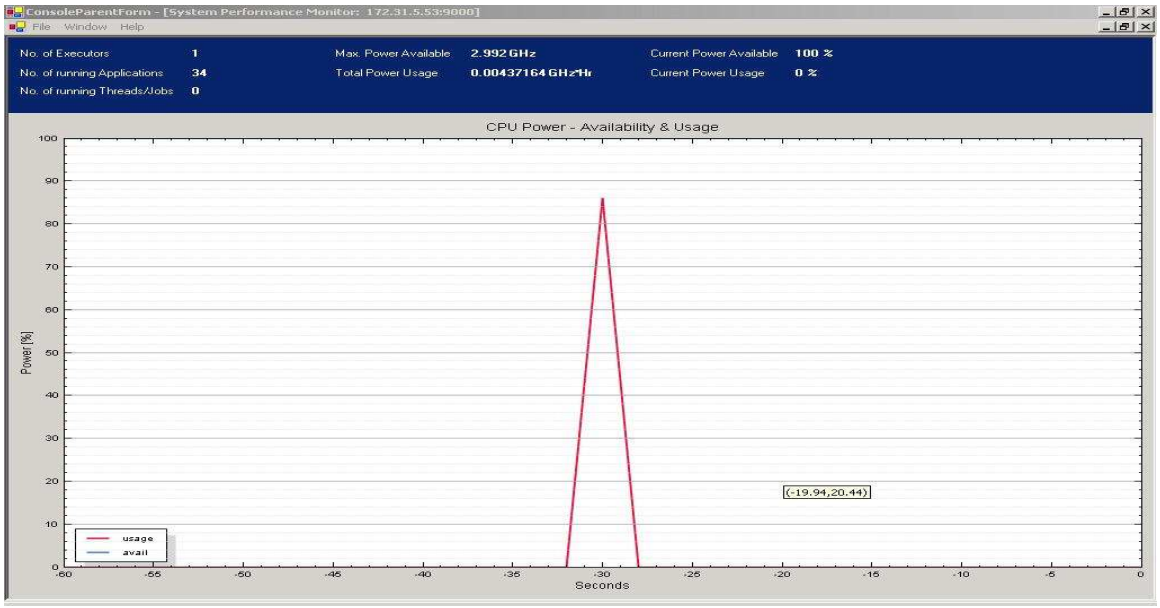


Figure 5.10: Power usages by "cex/Ram123" node

When we run an application on two Windows XP nodes (cex and Ram123) the power usages is approximately half as compare to the power usages by single node to complete

same application. A power usage by two nodes to complete an application is approximately 30% as shown in figure 5.11.

Figure 5.11: Power usages by "cex, Ram123" nodes

When we run an application on one Windows XP node (cex or Ram123) and on one Windows 2000 node then power usages is approximately 50%. But time taken by these nodes to complete application is more as compare to homogeneous platform nodes. So the results show the performance of the application degraded by introducing the heterogeneous Operating System in grid environment.

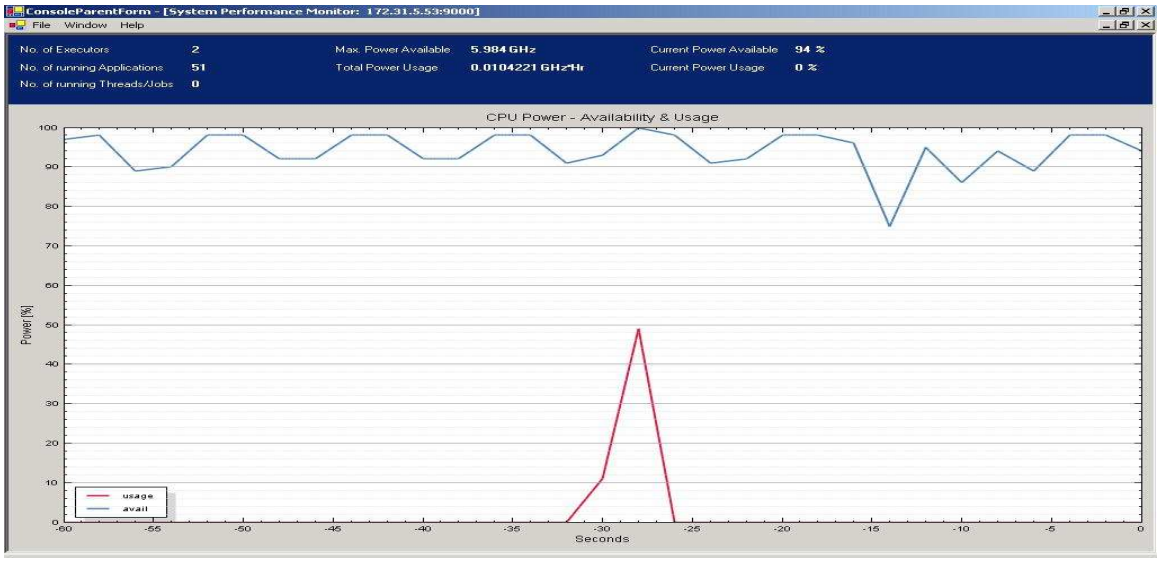


Figure 5.12: Power usages by "cex, tiet-goq0zlzs0" nodes

When we run an application on heterogeneous platforms means two Windows XP nodes (cex or Ram123) and on one Windows 2000 node then power usages is approximately 20%. But time taken by these nodes to complete application is more as compare to homogeneous platform nodes. So the results show the performance of the application degraded by introducing the heterogeneous Operating System in grid environment.

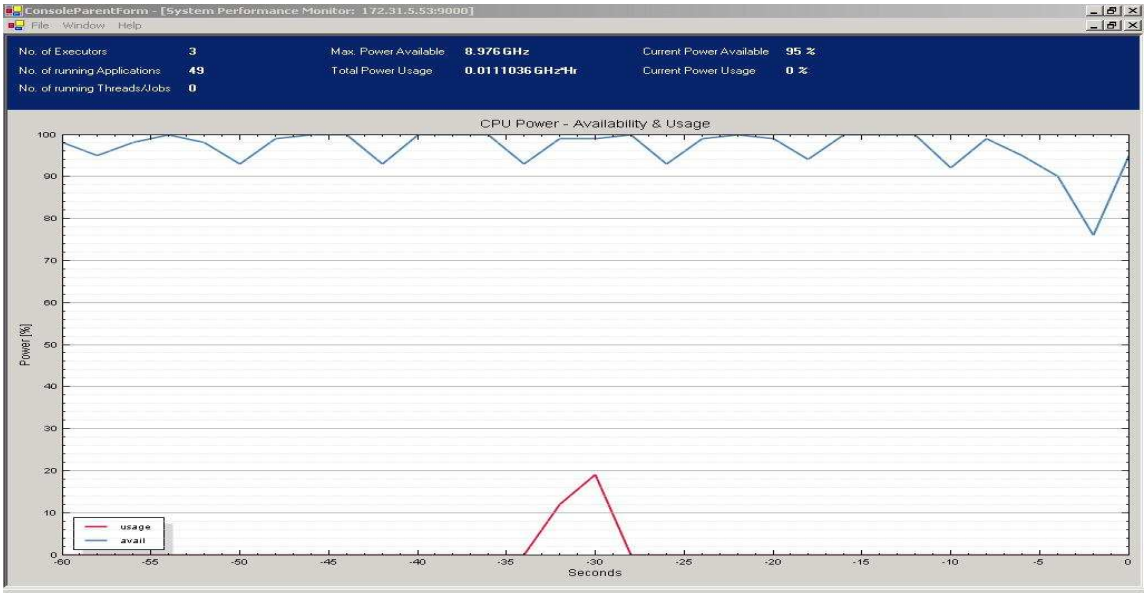


Figure 5.13: Power usages by "cex, Ram123, tiet-goq0zlzs0" nodes

We concluded the above result into the following graph in figure 5.14. Figure shows windows XP based grid environment is for more effective than Windows 2000 based grid and due to heterogeneity in operating system performance of the application also degraded.

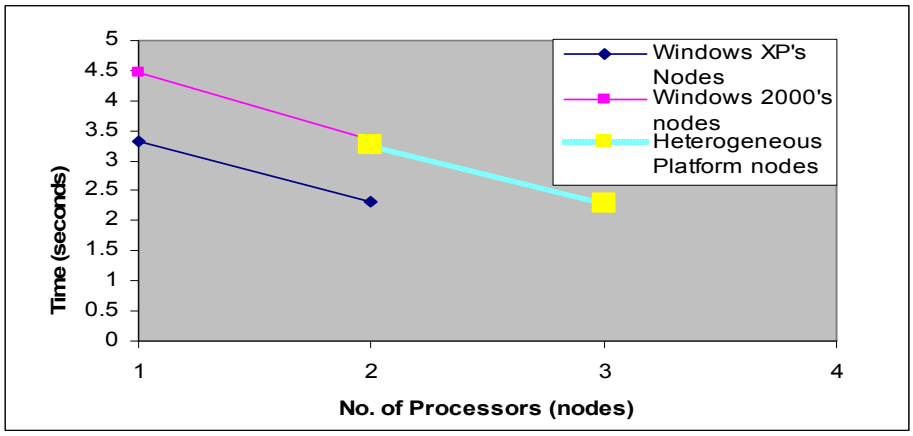


Figure 5.14: Performance in heterogeneous platforms

Testbed for Globus Middlewares in Grid Environment and Results

Now we are going to perform same experiment as above on GT4 based grid environment. We have three GT4 based nodes research4.tiet.ac.in, research5.tiet.ac.in and research6.tiet.ac.in and running on Red Hat Linux 9 platforms. Following is the configuration of these nodes:

Node Name	Processor	RAM/Hard Disk	Operating System	Grid Middleware
gt4.tiet.ac.in	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Red Hat Linux 9	GT4
gt5.tiet.ac.in	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Red Hat Linux 9	GT4
gt6.tiet.ac.in	Intel Pentium 4 3.00 GHz	512 MB/ 80 GB	Red Hat Linux 9	GT4

Table 5.7: Configurations for Globus Toolkit

Results: We started experiment by executing same Prime Number Generator application on each node separately and all of the results presented in table 5.8 represent the average of three separate runs. Then we added one more homogeneous node and executed the same application, again the results are average of three separate runs. Again one more homogeneous node added and executed the same application and results are average of three separate runs. The results are shown in following:

No. of Node	Name of Node(s)	Average Execution Time in Seconds
1	gt4.tiet.ac.in/ gt4.tiet.ac.in/ gt5.tiet.ac.in	20
2	gt4.tiet.ac.in, gt5.tiet.ac.in/ gt4.tiet.ac.in, gt6.tiet.ac.in/ gt5.tiet.ac.in, gt6.tiet.ac.in	12
3	gt4.tiet.ac.in, gt5.tiet.ac.in, gt6.tiet.ac.in	7

Table 5.8: Results of GT4 based homogeneous grid environment

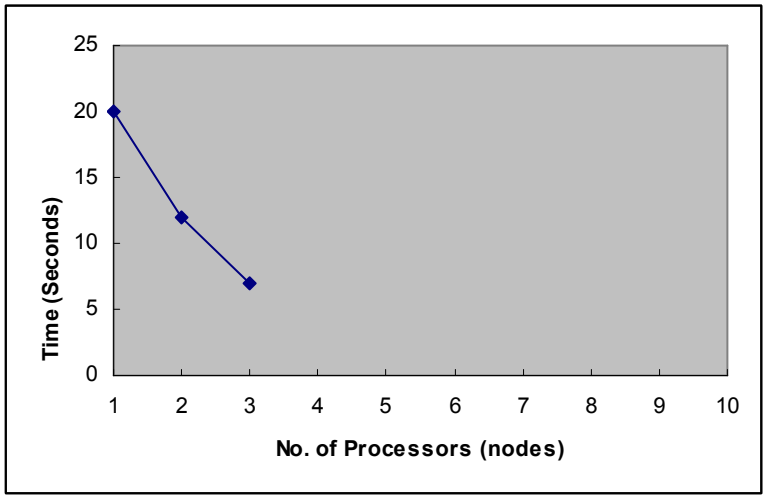


Figure 5.15: Performance evaluation on GT4

Figure 5.15 shows the time taken to complete prime number generator application on single node is approximately 20 seconds and when we added more nodes to this grid environment then performance increased.

But if we compare results of Gt4 and Alchemi based heterogeneous grid environment, then we can conclude that Alchemi based grid environment is more efficient than GT4 based grid environment.

Quantified Conclusion

We concluded all above results in figure 5.16.

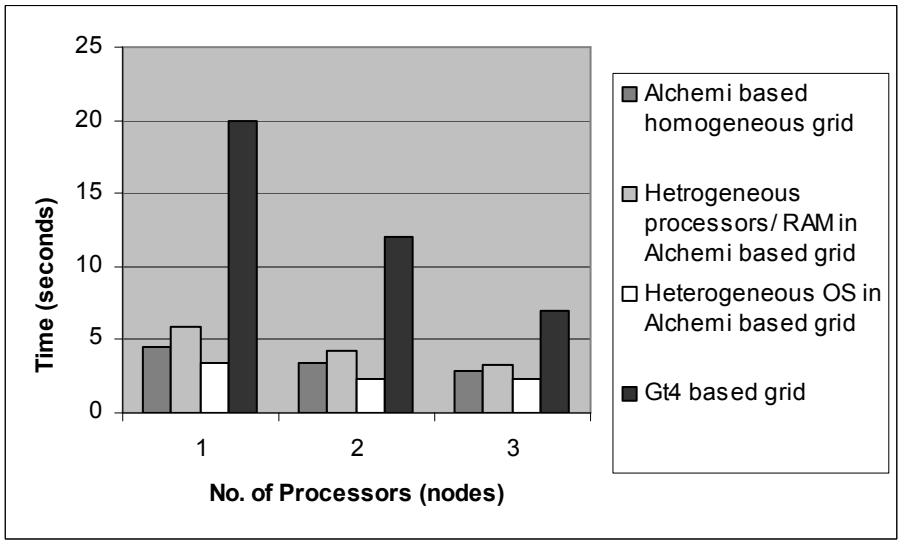


Figure 5.16: Performance Evaluation

Now we can quantify the impacts of resource heterogeneity on application performance. Form figure 5.16, when we introduced the heterogeneous processors and RAM in our grid environment then it took 16.66% more time to complete the given application as compare to homogeneous resources. When we run same application on two Windows XP nodes in grid environment then it took 30.74% less time to complete as compare to two Windows 2000 based grid system. From statistics, when we combined two Windows XP and one Windows 2000 node then application took 44.29% time more as compare to three homogeneous Windows XP based nodes. Finally when we run application with same logic (prime number generator) on three nodes based on GT4 and Red Hat Linux 9 platform then it took 183.33% more time to complete application as compare to three Alchemi based nodes.

5.6 Summary

In this chapter we have developed experimental framework for examining different factors that affects the application performance. We have used prime number generator application and run it on homogeneous as well as heterogeneous resources. On the basis on results gathered we quantified the impacts of resource heterogeneity.

In next chapter we are going to conclude our thesis work and suggestion for future directions.

This chapter discusses the conclusions of the work presented by this thesis. The chapter ends with a discussion of the future direction this work will take.

6.1 Conclusion

Through this thesis, we have described multiple aspects of Grid Computing and introduced numerous concepts which illustrate its broad capabilities. Grid Computing is definitely a promising tendency to solve high demanding applications and all kinds of problems. Objective of the grid environment is to achieve high performance computing by optimal usage of geographically distributed and heterogeneous resources.

But grid application performance remains a challenge in this heterogeneous grid environment. In the Grid environments, applications face heterogeneity in resources and dynamic load on the resources as other applications are sharing the resources.

There are number of factors, which can affect the grid application performance like resource heterogeneity, dynamic load on resources, task runtime prediction uncertainty, task-to-resource ratio and resource sharing in the Grid environment. In this thesis we focused on resource heterogeneity and tried to quantify the impacts of resource heterogeneity on grid application performance. We categorized resource heterogeneity in to main three categories as hardware heterogeneity, software heterogeneity and policy heterogeneity.

For quantification of these factors, we had defined an experimental framework including application characterization, GridBus broker (optional), and the heterogeneous computing environment to run the application. To allow more controllable and repeatable experiments, we realized this framework with a simulator. We quantified the affects of resource heterogeneity on prime number generator grid application by using different hardware configurations and software configurations.

In our experiment the heterogeneous processors and RAM in our grid environment took 16.66% more time to complete the given application as compare to homogeneous

resources. Two Windows XP nodes took 30.74% less time to complete an application as compare to two Windows 2000 based grid system. Heterogeneous operating systems nodes took 44.29% time more as compare to homogeneous Windows XP based nodes. Finally the application (prime number generator) on three GT4 and Red Hat Linux 9 platform nodes took 183.33% more time to complete application as compare to three Alchemi based nodes.

6.2 Future Directions

The work performed in this thesis can be used as one of the major parameter while developing the grid middlewares. This not only improves the performance of grid application but also make it more powerful, reliable and capable of handing more complex and large problems in heterogeneous environment.

A further extension to this work would be use to enhance load balancing and resource management techniques.

This work can be used to improve the quality of the performance monitoring tools like Network Weather System (NWS).

References

[1] Ian Foster, "What is the Grid? A Three Point Checklist", Argonne National Laboratory & University of Chicago, July 20, 2002.

[2] F. Berman, G. Fox, and T. Hey, "The Grid: past, present, future, in Grid Computing -- Making the Global Infrastructure a Reality." 2002, John Wiley & Sons, Ltd. p. 9-49.

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

[4] The Metacomputer: One from Many, NCSA (1995) http://archive.ncsa.uiuc.edu/Cyberia/MetaComp/MetaHome.html

[5] J. Joseph, M. Ernest, and C. Fellenstein, "Evolution of grid architecture and grid adoption models".

[6] Frederico Buchholz Maciel, "Resource Management in OGSA", http://forge.gridforum.org/projects/cmm-wg/, March 1, 2005.

[7] Allan Snavely, Greg Chun, Henri Casanova, San Diego, Rob F. Van der Wijngaart, Michael A. Frumkin, "Benchmarks for Grid Computing: A Review of Ongoing Efforts and Future Directions".

[8] http://www.planet-lab.org/

[9] http://www.teragrid.org/

[10] http://www.openpbs.org/

[11] Li, W., et al., "The Encyclopedia of Life Project: Grid Software and Deployment". Journal of New Generation Computing on Grid Systems for Life Sciences, 2004.

[12] Shirts, M.R. and V.S. Pande, "Screen Savers of the World", Unite! Science, 2000. 290(5498): p. 1903-1904.

[13] Jean-Christophe Durand, "Grid Computing a Conceptual and Practical Study", November 8, 2004.

[14] Oracle Application Server 10g Grid Computing An Oracle White Paper September 2003.

[15] Sean J. Norman: Grid computing: A grid application deployment system, MS Thesis, The University of Western Ontario London, Ontario, Canada. 2005.

[16] Allan Snavely, Greg Chun, Henri Casanova, Rob F. Van der Wijngaart, Michael A. Frumkin, "Benchmarks for Grid Computing: A Review of Ongoing Efforts and Future Directions".

[17] http://setiathome.berkeley.edu/

[18] Richard Huang: Scheduling Compute Intensive Application in Volatile, Shared Resource (Grid) Environment, MS Thesis, The University of California, San Diego. 2005.

[19] Gregor von Laszewski, Jarek Gawor, Carlos J. Pena, and Ian Foster, "InfoGram: A Grid Service that Support Both Information Queries and Job Execution". High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 24-26, 2002.

[20] Yang, L., J. Schopf, and I. Foster. "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments". Proceedings of Supercomputing Conference. 2003. Phoenix, AZ.

[21] P. Lindner, N. Currle-Linde, M. M. Resch, E. Gabriel. "Distributed Application Management in Heterogeneous GRIDS", The Web and the GRID: from e-science to e-business, Euroweb 2002

[22] Jack Dongarra and Alexey Lartovetsky, "An Overview of Heterogeneous High Performance and Grid computing".

[23] Hsun-Chang Chang, Kuan-Ching Li, Yaw-Ling Lin, Chao-Tung Yang, Hsiao-Hsi Wang, Liang-Teh Lee, "Performance Issues of Grid Computing Based on Different Architecture Cluster Computing Platforms," aina, pp. 321-324, 19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 2 (INA,, USW,, WAMIS,, and IPv6 papers), 2005.

[24] http://en.wikipedia.org/wiki/Windows_2000.

[25] Ivan Bowman, Saheem Siddiqi, Meyer C. Tanuan, “Concrete Architecture of the Linux Kernel”, University of Waterloo, 12-Feb-98.

[26] Prof. Richard Sinnott Dr. John Watt, Tutorial 1: Introduction to Globus Toolkit.

[27] Rishu Jaidka, Neeraj Jayaswal, Kenny Thomas Lucas, “Performance Analysis of Grid based application using MPICH G2 and Grid Application Framework for Java”. National Conference on RAFIT-2005, 2nd-3rd March, 2005.

[28] N1 Grid Engine 6 User’s Guide, Sun Microsystems, Inc. 4150 Network Circle Santa Clara, CA 95054 U.S.A. Part No: 817–6117–10, June 2004.

[29] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, “Peer-to-Peer Grid Computing and a .NET-based Alchemi - Framework”. The University of Melbourne, Australia

[30] Krishna Nadiminti, Akshay Luther, Rajkumar Buyya, Alchemi: A .NET-based Enterprise Grid System and Framework, User Guide for Alchemi 1.0 December 2005.

[31] R. Buyya, Economic-based Distributed Resource Management and Scheduling for Grid Computing, PhD Thesis, Monash University, Melbourne, Australia, April 2002.

[32] M. Litzkow, M. Livny, M. Mutka, “Condor---a hunter of idle workstations, in: Proceedings of the Eighth International Conference of Distributed Computing Systems”, June 1988.

[33] Rich Wolski1, Graziano Obertelli1, Matthew Allen1, Daniel Nurmi1, and John Brevik1, “Predicting Grid Resource Performance On-line”, University of California.

[34] Xian-He Sun, and Ming Wu, "GHS: A Performance Prediction and Task Scheduling System for Grid Computing". Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS’05). 2005.

[35] Fredrik Vraalsen, Ruth A. Aydt, Celso L. Mendes, and Daniel A. Reed. Performance contracts: Predicting and monitoring grid application behavior. Proceedings of GRID, 2001.

[36] K. Nadiminti, S. Venugopal, H. Gibbins, TianChi Ma and R. Buyya, The Gridbus Grid Service Broker and Scheduler (v.2.4) User Guide, Technical Report, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia.

These are the steps for installing various Alchemi Components. For every component we are giving its installation, configuration and operations are defined.

1. Alchemi Manager

Installation

The Alchemi Manager can be installed in two modes

As a normal Windows desktop application

As a windows service. (Supported only on Windows NT/2000/XP/2003)

To install the manager as a windows application, use the Manager Setup installer. For service mode installation uses the Manager Service Setup. The configuration steps are the same for both modes. In case of the service-mode, the "Alchemi Manager Service" installed and configured to run automatically on Windows start-up. After installation, the standard Windows service control manager can be used to control the service. Alternatively the Alchemi ManagerServiceController program can be used. The Manager Service controller is a graphical interface, which is exactly similar to the normal Manager application.

Install the Manager via the Manager installer. Use the sa password noted previously to install the database during the installation.

Configuration & Operation

The Manager can be run from the desktop or Start -> Programs -> Alchemi -> Manager -> Alchemi Manager. The database configuration settings used during installation automatically appear when the Manager is first started.

Click the "Start" button to start the Manager.

When closed, the Manager is minimised to the system tray.

2. Cross Platform Manager

Installation

Install the XPManager web service via the Cross Platform Manager installer.

Configuration

If the XPManager is installed on a different machine that the Manager, or if the default port of the Manager is changed, the web service's configuration must be modified. The XPManager is configured via the ASP.NET Web.config file located in the installation directory (wwwroot\Alchemi\CrossPlatformManager by default):

```
<appSettings>
<add key="ManagerUri" value="tcp://localhost:9000/Alchemi_Node" />
</appSettings>
```

Operation

The XPManager web service URL is of the format

http://[host_name]/[installation_path]

The default is therefore

http://[host_name]/Alchemi/CrossPlatformManager

The web service interfaces with the Manager. The Manager must therefore be running and started for the web service to work.

3. Executor

Installation

The Alchemi Executor can be installed in two modes

As a normal Windows desktop application

As a windows service. (Supported only on Windows NT/2000/XP/2003)

To install the executor as a windows application, use the Executor Setup installer. For service mode installation uses the Executor Service Setup. The configuration steps are the same for both modes. In case of the service-mode, the “Alchemi Executor Service” installed and configured to run automatically on Windows start-up. After installation, the standard Windows service control manager can be used to control the service. Alternatively the Alchemi ExecutorServiceController program can be used. The Executor

service controller is a graphical interface, which looks very similar to the normal Executor application.

Install the Executor via the Executor installer and follow the on-screen instructions.

Configuration & Operation

The Executor can be run from the desktop or Start -> Programs -> Alchemi -> Executor -> Alchemi Executor.

The Executor is configured from the application itself.

You need to configure 2 aspects of the Executor:

The host and port of the Manager to connect to.

Dedicated / non-dedicated execution. A non-dedicated Executor executes grid threads on a voluntary basis (it requests threads to execute from the Manager), while a dedicated Executor is always executing grid threads (it is directly provided grid threads to execute by the Manager). A non-dedicated Executor works behind firewalls.

Click the "Connect" button to connect the Executor to the Manager.

Installation of GT4 on Linux Appendix B

Our installation of GT4 is on Red Hat Linux 9. Globus Toolkit is the well known open source middleware for establishing Grid environment.

Installation Process

Step 1: Downloading GT4 from the web site.

Step 2: Preparing System effectively before Installation

Before starting the installation, prepare the system effectively as given below

Check out the prerequisites and platform compatibility. They are the most important.

Create a user named 'globus' in the local machine.

Set the basic environment variable GLOBUS_LOCATION to the directory selected for installing GT4.

Step3: Installing Prerequisites of GT4

Be sure to have the following software installed and configured on the Globus installation destination machine.

gcc, gcc-c++

make

tar

libxml2

Ant 1.6.5

J2SDk1.4.2.10

Postgres 8.1.2

Sudo 1.6.8 p12

GNU pkg

Step 4 Installation of GLOBUS

Set the variable GLOBUS_LOCATION=/usr/local/globus-4.0.1 and making the owner of this directory globus. Before getting into configuring of globus it is required that all paths are set correctly.

export JAVA_HOME /usr/local/J2SDK

export ANT_HOME /usr/local/ant

export GLOBUS_LOCATION /usr/local/globus-4.0.1

export PGDATA /usr/local/pgsql/data

After downloading the tarball, extract it using one of the following:

```
# tar -zxf gt4.0.0-all-source-installer.tar.gz
```

NOTE: You will need a "globus" user and group that will be used to perform administrative tasks such as starting and stopping the container, deploying services, etc. Set this user up on the installation system before continuing. The following actions will be performed by this user.

Next, create the Globus destination directory with the following commands. These must be done as root.

```
# mkdir /usr/local/globus-4.0.0

# chown globus:globus /usr/local/globus-4.0.0
```

Also, change the source extracted source directory and its contents to be owned by "globus". Note that this directory must be accessible by user "globus". i.e., having it in root would make it inaccessible by globus since /root can only be read by user "root".

```
# chown globus:globus gt4.0.0-all-source-installer -R
```

In the directory where the Globus source was extracted, perform the following as the "globus" user.

```
# cd gt4.0.0-all-source-installer
# su globus
globus$ export GLOBUS_LOCATION=/usr/local/globus-4.0.0
globus$ ./configure --prefix=$GLOBUS_LOCATION --with-flavor=gcc32dbg
```

The output of the configure command should be similar to the following. Note that we do not need a java compiler or ant since we won't be installing portions of Globus that require these.

[globus@myhostgt4.0.0-all-source-installer]$./configure-- prefix=$GLOBUS_LOCATION

We are now ready to compile Globus. Depending on how fast the install machine is, this can take over an hour. Feel free to have a quick nap while waiting.

```
globus$ make
globus$ make install
```

Step 5 Setting up Security

Globus enforces secure grid communication by using public key cryptography. Public key cryptography is used to digitally sign information. Each user on a Globus system is assigned a "certificate" that is used to sign information. SimpleCA is a package that provides a simplified certification authority for the purpose of issuing credentials to Globus Toolkit users and services.

Each certificate contains the following information:

A subject name, which identifies the person or object that the certificate represents.
The public key belonging to the subject.
The identity of a *Certificate Authority (CA)* that has signed the certificate to certify that the public key and the identity both belong to the subject.
The digital signature of the named CA.

Create Users

Before continuing, be sure to have 1) a system globus user (for administrative tasks, deploying services etc) 2) a regular user that will be used to run the client applications.

Simple-CA Setup

As the globus user, run the following. You only need to do this once per grid.

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

Installation of Gridbus Broker Appendix C

The following are the steps of installation of Gridbus broker version 2.4:

1. Requirements

Broker side (i.e. on the machine running the broker)

Java Virtual Machine 1.4 or higher

Valid grid certificates properly set up (if using remote Globus nodes)

By default the certificates are places in the <USER_HOME>/.globus directory Where <USER_HOME> is the user's home directory.

For a user "belle" on a UNIX machine this would be: /home/belle/.globus

For a user "belle" on a Windows NT/2000/XP machine this would be: C:\Documents and Settings\belle\.globus.

Additionally, some ports on the local node should be configured to be open so that the jobs can connect back to the broker. Please refer to the Globus documentation for more details.

Remote Grid node side

For a compute resource:

Middleware installation which is one of:

Globus 2.4

Globus 3.2 (with the pre-WS globus-gatekeeper and gridftp services running)

Globus 4.0

Alchemi 1.0 (Cross-platform manager)

Unicore Gateway 4.1 (experimental support within the broker)

Condor 6.6.9

Open PBS 2.3

SGE Sun N1 Grid Engine 6

2. Installation process

Installing the broker is a simple process. The broker is distributed as a .tar.gz (and a .zip) archive that can be downloaded from

http://www.gridbus.org/broker/2.4/gridbusbroker2.4.tar.gz.

http://www.gridbus.org/broker/2.4/gridbusbroker2.4.zip.

The installation just involves unzipping the files to any directory and optionally setting the PATH environment variable to include the broker executable script (gbb.sh or gbb.bat depending on your OS). Following are the steps to be followed:

Unzip the archive to the directory where you want to install the broker. In case of Windows, you can use WinZip (if you download the .zip file) or WinRar (for the .tar.gz)

In case of *nix, runs the command:

$ tar –zxvf gridbusbroker.2.4.tar.gz

Set the GBB_HOME variable to the directory where you have installed the broker.

Additionally, it is recommended to have the directory gridbus-broker2.4/bin added tothe system PATH variable.

For example, for a Bash shell:

$ export PATH=$PATH:<broker-install-directory>/bin

Set the permissions for the gbb.sh executable:

$ chmod 755 gbb.sh

Test the installation by running the broker from a shell:

$ ./gbb.sh –test

If you see a message confirming that the configuration is ok, congratulations! You have successfully installed the Gridbus broker on your machine.

Amarjit Manjotra, Inderveer Chana, **"Job Scheduling and Execution in Grid Environment"**, ICT-06, JMIT, Radaur, Haryana, India, February 09-11. **[Published in Conference Proceedings].**