Performance Evaluation using Nimrod/G: A Grid Resource Broker

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

Master of Engineering
in
Software Engineering

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

Submitted By
Vikas Mangla
(Roll No 8043125)

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

May 2006

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Performance Evaluation Using Nimrod/G: A Grid Resource Broker"** in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering at Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Dr. (Mrs.) Seema Bawa.

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

(Vikas Mangla)

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

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

Countersigned by

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

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

Acknowledgement

No sufficiently complex enterprise can be accomplished alone, and graduate research is no exception. I wish to express my deep gratitude to Dr. Seema Bawa, Professor and Head, Computer Science and Engineering Department for providing individual guidance and support throughout the Thesis work.

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

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

I would also like to acknowledge the TIET Grid group for all the valuable discussions that helped to clear all the problems and doubts, which I faced in 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.

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

Vikas Mangla

(8043125)

Abstract

Grid computing is the next generation IT infrastructure that promises to transform the way organizations and individuals compute, communicate and collaborate. It offers untapped processing cycles from networks of computers spanning vast geographical boundaries. Grid resource broker is responsible for resource discovery, resource selection, binding of software, data, and hardware resources, initiating computations, adapting to the changes in Grid resources, and presenting the Grid to the user as a single, unified resource. In general, a brokering system of Grid computing environments aims at delivering better performance. Desirable performance goals of Grid brokering include: maximizing system throughput, maximizing resource utilization, minimizing the execution time and fulfilling economical constraints.

In this thesis work, we have simulated the environment of Nimrod/G Resource Broker: an Economic based Grid Scheduler, which manages all operations associated with remote execution including resource discovery, trading, scheduling based on economic principles and a user defined quality of service requirement. The Nimrod-G resource broker is implemented by the services provide by Globus. Performance evaluation has been done in order to demonstrate how the performance is affected if by varying the number of jobs submitted to Nimrod/g or by varying the number of computing resources available in Grid environment.

Table of Contents

List of Figures

Organization of Thesis

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

The second chapter of the thesis is focused on Grid Resource Brokering. In it we discuss what brokering is, what are the components of a Grid resource broker, how brokering is done and what are the challenges involved in Grid environment.

The third chapter gives in depth details of the Nimrod/g: A Grid resource Broker. It tells what is the architecture of Nimrod-g, How it Schedule the resources and finally how an experiment is submitted to Nimrod/G for execution.

In the fourth chapter problem has been formulated.

The fifth Chapter explains the installation of the Nimrod-G, and then running the sample experiments on the Broker. Finally a performance Evaluation has been done on Nimrod/G Environment.

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

Chapter1

Introduction

In today's complex world of high speed computing, computers have become extremely powerful and 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. This gives us the idea of distributed computing, which further evolved into Grid computing [11].

Grid computing is the next generation IT infrastructure that promises to transform the way organizations and individuals compute, communicate and collaborate. It offers untapped processing cycles from networks of computers spanning vast geographical boundaries. Sharing in a Grid is not just a simple sharing of files but of hardware, software, data, and other resources [1]. Thus a complex yet secure sharing is at the heart of the Grid. It needs to be clearly defined as to what resources are going to be shared by the users, who are the users who are allowed to share these resources and under what conditions this sharing takes place by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

Grid focus on ensembles of distributed heterogeneous resources used as a platform for high performance computing. Grid services utilize the available computational resources so that tasks are run on whatever machine currently has available capacity. A Grid also allows a single large computation to be spread across several machines, each of which is executing some portion of the computation [1].

A Grid can be viewed as a seamless, integrated computational and collaborative environment and a high level view of activities. As shown in Figure 1.1, users interact with Grid resource broker for solving problem, which in turn performs resource

discovery, scheduling, and processing application jobs on the distributed Grid resources.

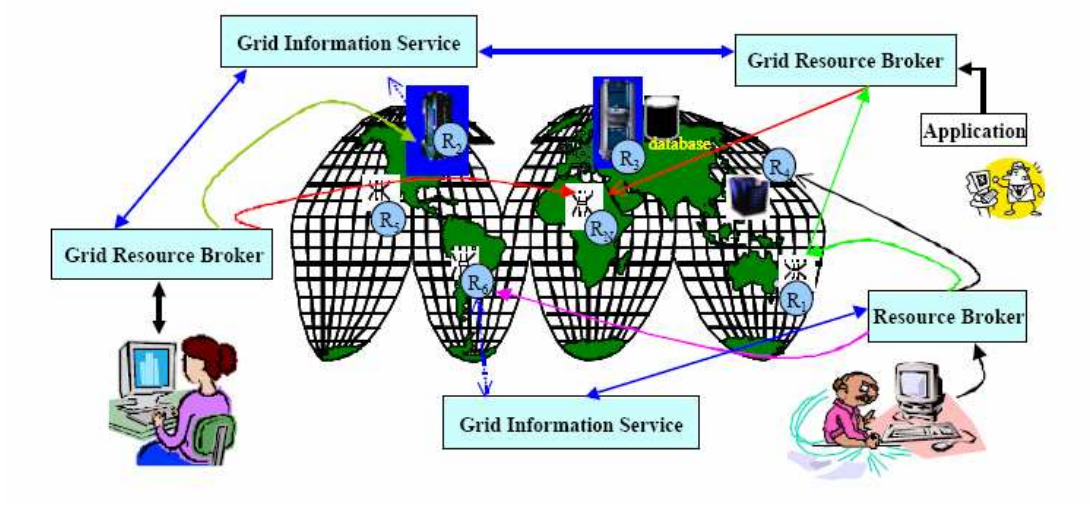

Figure 1.1 A high-level view of the Grid and interaction between its entities [12].

1.1 What is Grid?

Since Grid computing is still emerging, the concept of Grid computing itself is evolving. The definition given by Foster is widely accepted and frequently referred.

According to Foster,

"Grid computing strives to aggregate diverse, heterogeneous, geographically distributed and multiple-domain-spanning resources to provide a platform for transparent, secure, coordinated, and high-performance resource-sharing and problem solving [11]."

The resources that Grid computing is attempting to integrate are various. They include supercomputers, workstations, databases, storages, networks, software, special instruments, advanced display devices, and even people.

A Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [4].

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 [4].

1.2 Grid Types

Grids can be classified on the basis of two parameters

- Grid functionality
- Grid topology

1.2.1 Functional classification

According to the distinct targeted application realms, Grid systems can be classified into two categories [22]. But there are actually no hard boundaries between these Grid categories. Real Grids may be a combination of two of these types. The two categories of Grid systems are described below.

- Computational Grid
- Data Grid

Computational Grid

A computational Grid is a system that aims at achieving higher aggregate computational power than any single constituent machine. A high throughput Grid aims to increase the completion rate of a stream of jobs through utilizing available idle computing cycles as many as possible [22].

Data Grid

A data Grid is responsible for housing and providing access to data across multiple organizations. Users are not concerned with where this data is located as long as they have access to the data. For example, you may have two universities doing life science research, each with unique data. A data Grid would allow them to share their data, manage the data, and manage security issues [22].

1.2.2 Topology Classification

Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as hierarchy spanning the world. Grids can be classified into three categories according to the topology of Grid [22].

- IntraGrid
- ExtraGrid
- InterGrid

The relationship between the three Grid topologies is illustrated in Figure 1.2

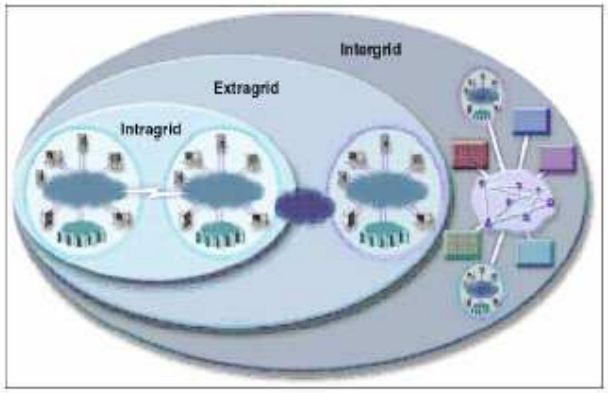

Figure. 1.2 Grid Types [2]

IntraGrid

A typical IntraGrid topology exists within a single organization. The single organization could be made up of a number of computers that share a common security domain, which are connected by a private high-speed local network. The primary characteristics of an IntraGrid are a single administrative domain and bandwidth guarantee on the private network. Within an IntraGrid, it is easier to design

the scheduling system, since an IntraGrid provides a relatively static set of computing resources and communication capability between machines [22].

ExtraGrid

An ExtraGrid couples two or more IntraGrid. The ExtraGrid typically involves more than one administrative domain, and the level of management complexity increases. The primary characteristics of an ExtraGrid are dispersed security, multiple domains, and remote/WAN connectivity. Within an ExtraGrid, the resources become more dynamic. A business would benefit from an ExtraGrid if there were a business initiative of integrating with external trusted business partners [22].

InterGrid

An InterGrid has an analogy with the Internet. It is the most complicated form of Grid topology. The primary characteristics of an InterGrid include dispersed security, multiple domains and WAN connectivity. A business may deem an InterGrid necessary if there is a need for a collaborative computing community, or simplified end-to-end processes with the organizations that will use the InterGrid [22].

1.3 Grid Architecture

The architecture of the Grid is often described in terms of "layers", each providing a specific function. In general, the higher layers are focused on the user (user-centric, in the jargon), whereas the lower layers are more focused on computers and networks (hardware-centric) [8].

Fabric Layer

For all the physical infrastructure of the Grid, including computers and the communication network. It is made up of the actual resources that are part of the Grid, such as computers, storage systems, electronic data catalogues, and even sensors such as telescopes or other instruments, which can be connected directly to the network.

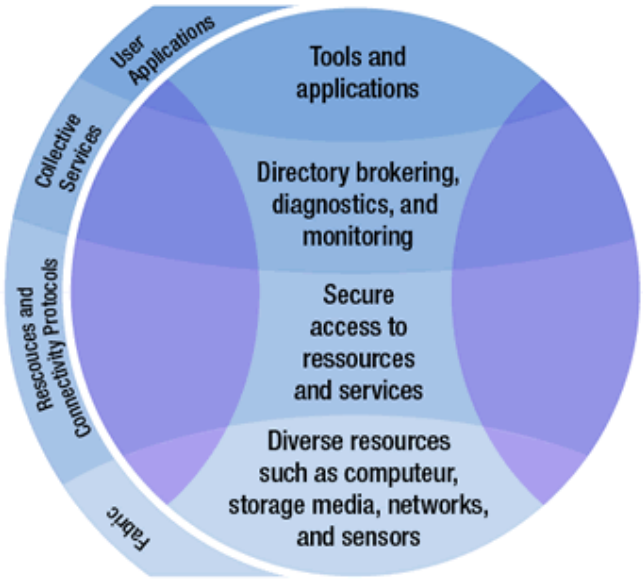

Figure 1.3 Layered Grid Architecture [8]

Resource and connectivity protocols

Resource and connectivity protocols handle all "Grid specific" network transactions between different computers and other resources on the Grid. Grids have to be able to recognize messages that are relevant to them, and filter out the rest. This is done with communication protocols, which let the resources speak to each other, enabling exchange of data, and authentication protocols, which provide secure mechanisms for verifying the identity of both users and resources [8].

Collective services

The collective services are also based on protocols: information protocols, which obtain information about the structure and state of the resources on the Grid, and management protocols, which negotiate access to resources in a uniform way. The services include:

- Keeping directories of available resources updated at all times,
- Brokering resources
- Monitoring and diagnosing problems on the Grid
- Replicating key data so that multiple copies are available at different locations
- Providing membership/policy services for keeping track on the Grid of who is allowed to do what, when [8].

Applications Layer

The highest layer of the structure is the application layer, which includes all different user applications (science, engineering, and business, financial), portals and development toolkits supporting the applications. This is the layer that users of the Grid will "see" [8].

1.4 Ideas Behind the Grid

Of course, there are many big ideas behind the Grid. And of course, some of them have been around long before the name Grid appeared. Nevertheless, if you look at where the software engineers and developers who are building the Grid are spending their time and effort, then there are four big areas [9].

- Resource Sharing
- Security
- Load Balancing
- Open Standards

Resource Sharing

The First Big Idea behind the Grid is sharing of resources: You enter the Grid to use remote resources, which allows you to do things that you cannot do with the computer you own, or the computer center this is more than simple file exchange: it is direct access to remote software, computers and data. It can even give you access and control of remote sensors, telescopes and other devices that do not belong to you [9].

Security

The Second Big Idea behind the Grid security. The Grid needs an efficient way to keep track of all this information [9]: who is authorized to use the Grid, and which resources on the Grid are they authorized to use? Who authenticates that a given user is who he says he is? What are the usage policies of the different resources?

Load Balancing

Load Balancing is one of the major concerns in Grid Environment. It focuses on the efficient utilization of the resources in the Grid. There should not be a case one resource is over utilized and another is not utilized at all [9].

Open Standards

The next Big Idea behind the Grid is open standards. The idea is to convince the community of software engineers currently developing the Grid, including those from major IT companies, to set common standards for the Grid up-front, so that applications made to run on one Grid will run on all others [9].

1.5 Benefits of Grid Computing

Some of the benefits of Grid Computing are listed below [1]:

- Provides users around the world with dynamic and adaptive access to unparalleled levels of computing.
- Improved methods for collaborative work.
- Unprecedented Price-to-Performance ratio.
- With the infrastructure provided by the Grid, scientists are able to perform complex tasks, integrate their work and collaborate remotely.
- Grids can lead to savings in processing time.
- Efficient, effective, and economic utilization of available resources.
- Increased availability and reliability of resources.
- Shared access (by multiple users) to large amounts of data.
- Seamless and secure access to large number of geographically distributed resources.
- Reduction in average job response time may occur but an overhead of limited network bandwidth and latency exists [1].

In this chapter, we have an introduction about Grid computing, what are the types of Grids, what are the big ideas behind the Grid and what are the benefits offered by Grids to the end users.

Chapter 2

Grid Brokering and Scheduling

The Grid resource broker acts as a mediator between the user and Grid resources using middleware services. It is responsible for resource discovery, resource selection, binding of software, data, and hardware resources, initiating computations, adapting to the changes in Grid resources, and presenting the Grid to the user as a single, unified resource.

Grid brokering is defined as the process of making scheduling decisions involving resources over multiple administrative domains. Grid brokering is a software framework with which the scheduler collects resource state information, selects appropriate resources, predicts the potential performance for each candidate schedule, and determines the best schedule for the applications to be executed on a Grid system subject to some performance goals.

In general, a brokering system of Grid computing environments aims at delivering better performance. Desirable performance goals of Grid scheduling include: maximizing system throughput, maximizing resource utilization, minimizing the execution time and fulfilling economical constraints.

2.1 Challenges for Grid Brokering

Although a Grid also falls into the category of distributed parallel computing environments, it has a lot of unique characteristics, which make the scheduling in Grid environments highly difficult. An adequate Grid brokering system should overcome these challenges to leverage the promising potential of Grid systems, providing high-performance services [22].

The grand challenges imposed by Grid systems are examined in detail in the following:

- **Resource Heterogeneity**

A Grid mainly has two categories of resources: networks and computational resources. Heterogeneity exists in both of the two categories of resources. First, networks used to interconnect these computational resources may differ significantly in terms of their bandwidth and communication protocols. A wide-area Grid may have to utilize the best-effort services provided by the Internet [22]. Second, computational resources are usually heterogeneous in that these resources may have different hardware, such as instruction set, computer architectures, number of processor, physical memory size, CPU speed, and so on, and also different software, such as different operating systems, file systems, cluster management software, and so on. The heterogeneity results in differing capability of processing jobs. Resources with different capacity could not be considered uniformly. An adequate brokering system should address the heterogeneity and further leverage different computing power of diverse resources.

- **Site Autonomy**

Typically a Grid may comprise multiple administrative domains. Each domain shares a common security and management policy. Each domain usually authorizes a group of users to use the resources in the domain. Thus applications from non-authorized users should not be eligible to run on the resources in some specific domains.

Further more, a site is an autonomous computational entity. A shared site will result in many problems. It usually has its own brokering policy, which complicates the prediction of a job on the site [22]. A single overall Performance goal is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performance goal.

Local priority is another important issue. Certain classes of jobs have higher priority only on certain specific resources. For example, it can be expected that local jobs will be assigned higher priorities such that local jobs will be better served on the local resources.

- **Resource Non-dedication**

Because of non-dedication of resources, resource usage contention is a major issue. Competition may exist in both computational resources and interconnection networks. Due to the non-dedication of resources, a resource may join multiple Grids simultaneously. The workloads from both local users and other Grids share the resource concurrently. The underlying interconnection network is shared as well [22]. One consequence of contention is that behavior and performance can vary over time. (Contention-free) at the guaranteed level. Brokers must be able to consider the effects of contention and predict the available resource capabilities.

- **Application Diversity**

The problem arises because the Grid applications are from a wide range of users, each having its own special requirements. For example, some applications may require sequential execution, some applications may consist of a set of independent jobs, and others may consist of a set of dependent jobs. In this context, building a general-purpose scheduling system seems extremely difficult. An adequate brokering system should be able to handle a variety of applications [22].

- **Dynamic Behavior**

In traditional parallel computing environments, such as a cluster, the pool of resources is assumed to be fixed or stable. In a Grid environment, dynamics exists in both the networks and computational resources. First, a network shared by many parties cannot provide guaranteed bandwidth. This is particularly true when wide-area networks such as the Internet are involved. Second, both the avail-ability and capability of computational resources will exhibit dynamic behavior. On one hand new resources may join the Grid, and on the other hand, some resources may become unavailable due do problems such as network failure. The capability of resources may vary overtime due to the contention among many parties who share the resources [22]. An adequate broker should adapt to such dynamic behavior. After a new resource joins the Grid, the broker should be able to detect it automatically and leverage the new resource in the later scheduling decision-making. When a computational resource becomes unavailable resulting from an unexceptional failure, mechanisms, such as check pointing or rescheduling, should be taken to guarantee the reliability of Grid systems.

These challenges pose significant obstacles on the problem of designing an efficient and effective brokering system for Grid environments. Some problems resulting from the above are not solved successfully yet and still open research issues. As a result, brokering frame work must be developed for

So these concerns [22] put the following requirements on any Grid Broker

- Adaptive to the dynamic environment.
- Adaptive to the varying performance metrics upon the course of application execution.
- Performance predictions over time.
- Coarse and fine-tuning the component parameters.

2.2 Grid Broker Components

The resource broker mainly consists of the following components:

2.2.1 Job Control Agent:

It is a persistent control engine responsible for shepherding a job through the system. It coordinates with schedule adviser for schedule generation, handles actual creation of jobs, maintenance of job status, interacting with clients/users, schedule adviser, and dispatcher [16].

2.2.2 Schedule Adviser:

It is responsible for resource discovery (using the Grid explorer), resource selection and job assignment (schedule generation) to ensure that the user requirements are met.

2.2.3 Grid Explorer:

It is responsible for resource discovery by interacting with the Grid-information server and identifying the list of authorized machines, and keeping track of resource status information.

2.2.4 Trade Manager:

This works under the direction of resource selection algorithm (schedule advisor) to identify resource access costs. It interacts with trade servers and negotiates for access to resources at low costs.

2.2.5 Deployment Agent:

It is responsible for activating task execution on the selected resource as per the scheduler's instruction and periodically updates the status of task execution to Job Control Agent [16].

2.3 Grid Brokering Procedure

A user goes through three stages to schedule a job when it involves multiple sites. Phase one is resource discovery, in which the user makes a list of potential resources to use. Phase two involves gathering information about those resources and choosing a best set to use. In phase three the user then runs the job [19][20].

2.3.1 Phase 1: Resource Discovery

Resource discovery involves the user selecting a set of resources to investigate in more detail in phase two, information gathering. At the beginning of this phase, the potential set of resources is the empty set, and at the end of this phase, the potential set of resources is some set that has passed a minimal feasibility requirement. Most users do this in three steps: authorization filtering, job requirement knowledge, and filtering to meet the minimal job requirements [19][20].

Authorization Filtering

It is generally assumed that a user will know which resources he or she has access to in terms of basic services. At the end of this step the user will have a list of machines or resources to which he or she has access.

- **Application requirement definition**

In order to proceed in resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources (see Step 3). The set of possible job requirements can be very broad, and vary significantly between jobs. It may include static details, such as the operating system or hardware for which a binary of the code is available, or that the code is best suited to a specific architecture [19].

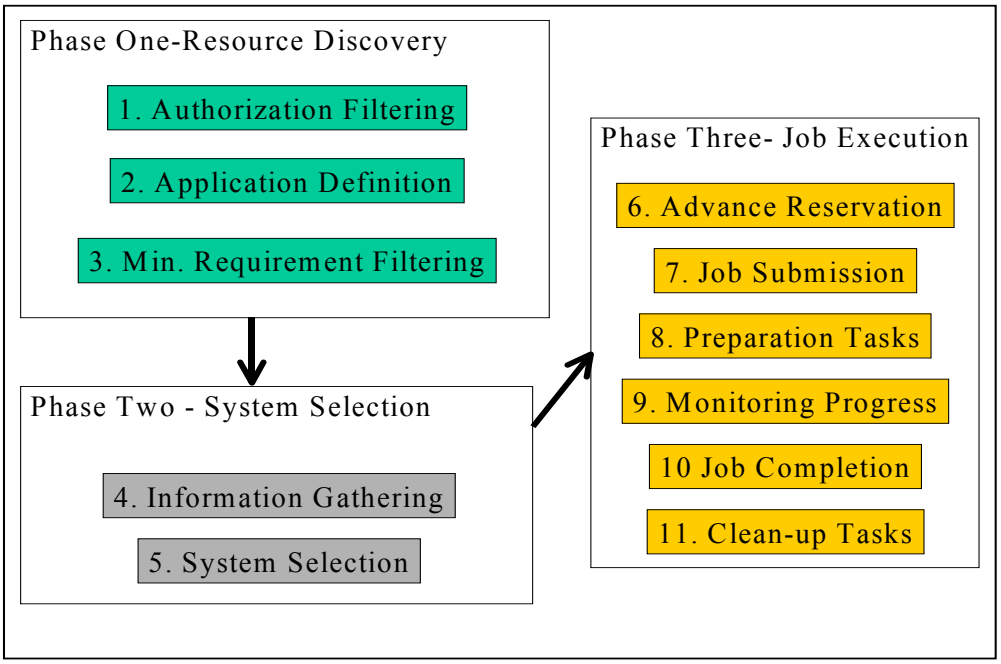

Fig2.1 Grid Brokering Procedure [3]

Dynamic details are also possible, for example a minimum RAM requirement, connectivity needed, /tmp space needed, etc. This may include any information about the job that should be specified to make sure that the job could be matched to a set of resources.

- **Minimal requirement filtering**

Given a set of resources to which a user has access and the minimal set of requirements the job has, the third step in the resource discovery phase is to filter out the resources that do not meet the minimal job requirements. The user generally does this step by going through the list of resources and eliminating the ones that do not

meet the job requirements as much as they're known. It could also be combined with the gathering of more detailed information about each resource (step 4) (and in fact this is how most proposed systems go about the process). However, when being done by hand, if a user can eliminate an inappropriate resource it is done at this stage to simplify the information gathering in the next phase [20].

2.3.2 Phase 2: System Selection

Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job. This is generally done in two steps: gathering information and making a decision [19][20].

- **Gathering information (query)**

In order to make the best possible resource match, a user needs to gather dynamic information about the resources in question. Depending on the application and resource in question, different information may be needed. Take for instance the simple case of finding the best single resource for a job to run on. A user might want to know the load on the various machines queue lengths if the machine has queues. In addition, physical characteristics and software requirements play a role – is the compiler you need on the machine, is the disk big enough for the data, etc. Then there are location/connectivity issues – is the machine close enough to the data store? All of these issues are multiplied in the case of multiple resources. Making an advance reservation may or may not be a part of this step [20].

- **Select the system(s) to run on**

Given the information gathered by the previous step, a decision of which resource (or set of resources) should the user submits a job is made in the step. This can and will be done in a variety of ways. Note that this does not address the situation of speculative execution – when a job is submitted to multiple resources, and when one begins to run the other submissions is cancelled. This is only the selection of a resource (or set of resources).

2.3.3 Phase 3: Run job

The third phase of scheduling is running a job. This involves a number of steps, few of which have been defined in a uniform way between resources [19][20]. They include:

- **(Optional) Make an advance reservation**

It may be the case that to make the best use of a given system, part or all of the resources will have to be reserved in advance. Depending on the resource, this can be easy or hard to do, may be done with mechanical means as opposed to human means, and the reservations may or may not expire with or without cost.

- **Submit job to resources**

Once resources are chosen the application must be submitted to the resources. This may be as easy as running a single command or as complicated as running a series of scripts, and may or may not include setup or staging.

- **Preparation Tasks**

The Preparation stage may involve setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application.

- **Monitor progress (maybe go back to 4)**

Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing.

- **Find out Job is done**

When the job is finished, the user needs to be notified.

- **Completion tasks**

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, break down the environment, remove temporary settings, etc [19].

This chapter gives us overview of Grid resource brokering. It explores the broker components, brokering procedure and certain challenges, a Grid broker need to overcome.

Chapter3

Nimrod/G: A Grid Resource Broker

3.1 Introduction

Nimrod-G is a tool for automated modeling and execution of parameter sweep applications (parameter studies) over global computational Grids [15][17]. It provides a simple declarative parametric modeling language for expressing parametric experiments. A domain expert can easily create a plan for a parametric experiment and use the Nimrod-G system to deploy jobs on distributed resources for execution. It uses novel resource management and scheduling algorithms based on economic principles. Specifically, it supports user-defined deadline and budget constraints for schedule optimizations and manages supply and demand of resources in the Grid using a set of resource trading services [16]. Nimrod-G provides a persistent and programmable *task-farming engine* (TFE) that enables "plugging" of user-defined schedulers and customized applications or problem solving environments (e.g., Active Sheets) in place of default components. The task-farming engine is a coordination point for processes performing resource trading, scheduling, data and executable staging, remote execution, and result collation. The Nimrod-G project builds on the early work [5][7] that focused on creating *tools* that help domain experts to compose their legacy serial applications for parameter studies and run them on computational clusters and manually managed Grids. The Nimrod-G system automates the allocation of resources and application scheduling on the Grid using economic principles in order to provide some measurable quality-of-service to the end user. That is, the focus of this thesis is within an intersection area of Grid architectures, economic principles, and scheduling optimizations, which is essential for pushing the Grid into the mainstream computing. [3]

3.2 Architecture

Nimrod-G has been developed as a Grid resource broker based on the GRACE framework. It leverages services provided by Grid middleware systems such as Globus, Legion, and the GRACE trading mechanisms. The middleware systems provide a set of low-level protocols for secure and uniform access to remote resources; and services for accessing resources information and storage management. The modular and layered architecture of Nimrod-G is shown in Figure 3.1. The Nimrod-G architecture follows hourglass design model that allows its implementation on top of different middleware systems and enables the usage of its services by multiple clients/applications.

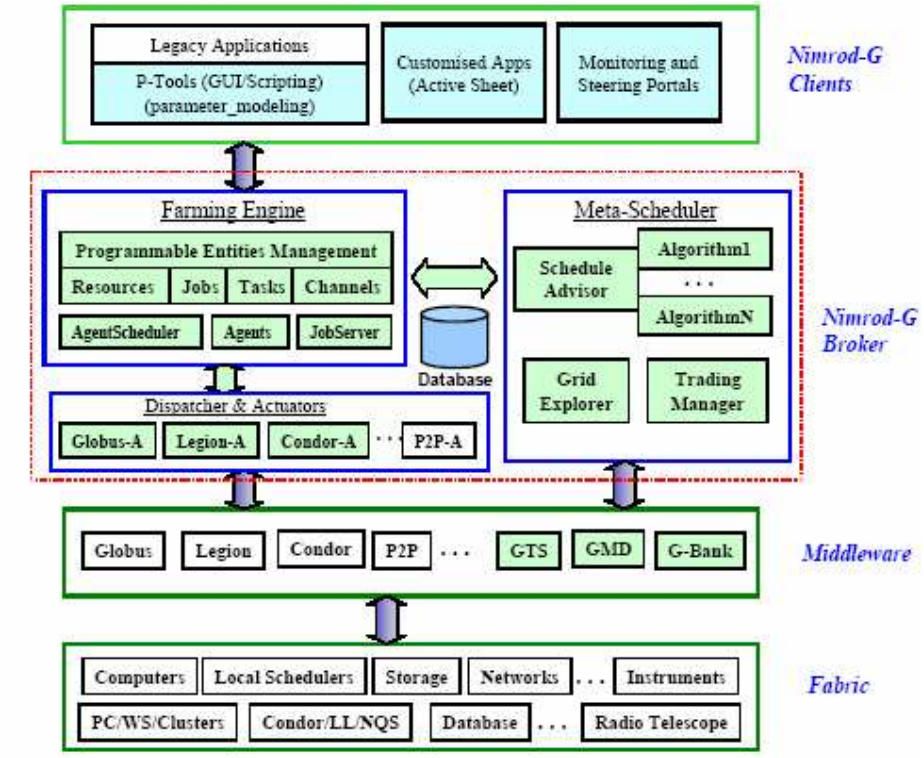

Fig3.1 Layered Architecture of Nimrod/G [14]

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

- Nimrod-G Clients, which can be:
    - Tools for creating parameter sweep applications.
    - Steering and control monitors, and

- Customized end user applications (e.g., Active Sheets) [19].

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

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

3.2.1 Nimrod-G Clients

- **Tools for Creating Parameter Sweep Applications**

Nimrod supports GUI tools and declarative programming language that assist in creation of parameter sweep applications [7]. They allow the user to:

a) Parameterize input files,

b) Prepare a plan file containing the commands that define parameters and their values,

c) Generate a run file, which converts the generic plan file to a detailed list of jobs, and

d) Control and monitor execution of the jobs.

The application execution environment handles online creation of input files and command line arguments through parameter substitution.

- **Steering and Control Monitors**

These components act as a user-interface for controlling and monitoring a Nimrod-G experiment. The user can vary constraints related to time and cost that influence the direction the scheduler takes while selecting resources. It serves as a monitoring console and lists the status of all jobs, which a user can view and control. A Nimrod-G monitoring and steering client snapshot is shown in Figure 4.3. Another feature of the

Nimrod-G client is that it is possible to run multiple instances of the same client at different locations. That means the experiment can be started on one machine,

monitored on another machine by the same or different user, and the experiment can be controlled from yet another location. We have used this feature to monitor and control an experiment from Monash University and Pittsburgh Supercomputing Center at Carnegie-Mellon University simultaneously during HPDC-2000 research demonstrations.

- **Customized End User Applications**

Specialized applications can be developed to create jobs at runtime and add jobs to the Nimrod-G Engine for processing on the Grid. These applications can use the Nimrod-G job management services (APIs and protocols described in [21]) for adding and managing jobs. One such application is Active Sheets [6], an extended Microsoft Excel spreadsheet that submits cell functions as jobs to the Nimrod-G broker for parallel execution on the Grid. Another example is the Nimrod/O system, a tool that uses non-linear optimization algorithms to facilitate automatic optimal design.

3.2.2 The Nimrod-G Grid Resource Broker

The Nimrod-G Resource broker is responsible for determining the specific requirements that an experiment places on the Grid and performing resource discovery, scheduling, dispatching jobs to remote Grid nodes, starting and managing job execution, and gathering results back to the home node. The sub-modules of our resource broker are, the task farming engine; the scheduler that consists of a Grid explorer for resource discovery, a schedule advisor backed with scheduling algorithms, and a resource trading manager; a dispatcher and an actuator for deploying agents on Grid resources; and agents for managing execution of Nimrod-G jobs on Grid resources [14]. The interaction between components of the Nimrod-G runtime machinery and Grid services during runtime is shown in Figure 3.2.

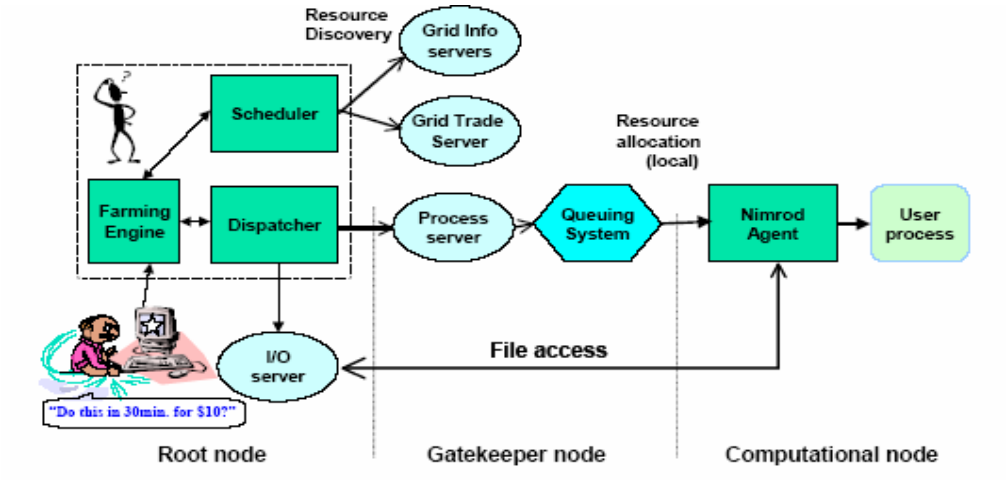

Fig 3.2 The Flow actions in Nimrod/G runtime environment [14]

The machine on which the broker runs is called the *root node*, the machine (e.g., a cluster master node) that acts as a front-end to a Grid resource and forwards the user jobs to queuing system or forks them for execution is called the *gatekeeper node*, and the machine (e.g., cluster worker node) that executes the user job is called the *computational node*.

- **The Task Farming Engine (TFE)**

The Nimrod-G task-farming engine is a persistent and programmable job control agent that manages and controls an experiment. The farming engine is responsible for managing the execution of parameterized application jobs. It takes care of the actual creation of jobs, the maintenance of job status, and providing a means for interaction between the clients, the schedule advisor, and the dispatcher. The scheduler and dispatcher respectively interact with the TFE to map jobs to resources and deploy on them. That is, the TFE manages the experiment under the direction of schedule advisors, and then instructs the dispatcher to deploy an application job for execution on the selected resource.

The TFE maintains the state of an entire experiment and ensures that it is recorded in persistent storage. This helps in keeping track of the experiment progress (e.g., status of jobs execution and resources status) and allows the experiment to be restarted if the root node fails without the need for execution of jobs that are already executed. The TFE exposes interfaces for job, resource, and task management along with the job-to-resource mapping APIs and protocols described in [21]. The developers of scheduling

algorithms can use these interfaces to implement their own schedulers rapidly by taking advantage of Nimrod-G TFE and dispatcher capability without concern for the complexity of low-level remote execution mechanisms.

The programmable capability of the task-farming engine enables "plugging" of user-defined schedulers and customized clients or problem solving environments (e.g., Active Sheets [6]) in place of the default components. The task-farming engine is a coordination point for processes performing resource trading, scheduling, data and executable staging, remote execution, and result collation [14].

- **The Scheduler**

The scheduler is responsible for resource discovery, resource trading, resource selection, and job assignment. The resource discovery algorithm interacts with an information service (the MDS in Globus), identifies the list of authorized and available machines, trades for resource access cost, and keeps track of resource status information. The resource selection algorithm is responsible for selecting those resources that meet the deadline and budget constraints along with optimization requirements. Nimrod-G incorporates three different algorithms for deadline and budget constrained scheduling [18].

- **The Dispatcher and Actuators**

The dispatcher triggers appropriate actuators—depending on the type of middleware running on resources—to deploy agents on Grid resources and assign one of the resource-mapped jobs for execution. Even though the schedule advisor creates a schedule for the entire duration based on user requirements, the dispatcher deploys jobs on resources periodically, depending on the load and the number of free CPUs available. When the dispatcher decides to deploy computation, it triggers appropriate actuator depending on middleware service. For example, a Globus-specific actuator is required for Globus resources, and a Legion-specific actuator is required for Legion resources [14].

- **Agents**

Nimrod-G agents are deployed on Grid resources dynamically at runtime depending on the scheduler's instructions. The agent is submitted as a job to the resource process

server (e.g., GRAM gatekeeper for a resource running Globus), which then submits to the local resource manager (fork manager in case of timeshare resources and queuing system in case of space-shared resource) for starting its execution. The agent is responsible for setting up the execution environment on a given resource for a user job. It is responsible for transporting the code and data to the machine; starting the execution of the task on the assigned resource and sending results back to the TFE. Since the agent operates on the "far side" of the middleware resource management components, it needs to provide error-detection for the user's job, sending the job termination status information back to the TFE. The Nimrod-G agent also records the amount of resource consumed during job execution, such as the CPU time and wall clock time. The online measurement of the amount of resource consumed by the job during its execution helps the scheduler evaluate resource performance and change the schedule accordingly. Typically, there is only one type of agent for all mechanisms, irrespective of whether they are fork or queue nodes. However, different agents are required for different middleware systems. [3]

3.3 Experiment management in Nimrod/G

There are sequence of steps that user has to perform: First user has to create a plan file which is composed of two sections parameter section and data section, after this plan file is need to be converted to a run file, then particular experiment is added to the Nimrod/g database and further the resources are assigned on which the jobs of the submitted experiment will run. The machine on which user submits the experiment is called root node and the machines on which actually computation is performed are called computational nodes. Thus a given experiment can be conducted with one root and multiple nodes. The sequential steps for execution of an experiment using Nimrod/g are:

3.3.1 Creating a plan file:

Nimrod/G uses a simple declarative language to describe the experiments. This description is usually written as a simple script file called a 'plan file'. There are two sections in this plan file; the first one describes the parameters that your experiment has; the second one describes the task that Nimrod/G needs to execute to complete a

your experiment. The two subsections below will explain the 'parameter' and 'task' parts separately:

3.3.1.1 Parameter Section:

Parameters are used to specify the range of values for a given experiment from which the various jobs will be created [23]. There is a simple syntax that needs to be followed in order to have a valid parameter line in the plan file:

parameter <name> <type> [<domain>]

Name: This is the parameter name. Every parameter has to have a unique name. The main reason for this to distinguish between different parameters

Type: This part of the parameter tells Nimrod/G what type of parameter this is. There are three types and they are float, integer and text

Domain: This is an optional part of the parameter line. Well, optional in some cases, where in other cases it is required. There are following domains available in this descriptive language:

- **Single value** - parameter has only a single value. This is the default domain, i.e. it is used if no other domain has been specified.
- **Range** - this parameter has a number of values that are within a specified range, i.e. values are generated within the lower and upper bounds for this parameter. These bounds need to be specified in the plan file. The syntax for the range domain is**:** *e.g. range from <value> to <value> step <value>*
- **Random** - this domain of parameter has a number of randomly generated points between specified lower and upper boundaries. The syntax for this domain is *random from <value> to <value> e.g. random from 1.0 to 15.0 points 6.* In this example there are going to be 6 randomly generated values between 1.0 and 15.0 and those values will be used for different jobs in your experiment.
- **select anyof** - With this parameter any combination of values in the value list can be used for the experiments. We usually select all of them but it is not required to be that way as it can be seen in the plan file above. In the example above only 1, 3 and 5 will be considered as parameter values for experiment jobs. If the default selection is not made, all of the values will be used. The

syntax for this domain is *select anyof <value_list> [default <value_list>]* where value_list consist of number of values separated with commas (**,**).

- **select oneof** - Similar to the one above except that only one value can be select for use from this list. If the default value is not specified the first value in the value_list is going to be used. The syntax for this domain is select oneof <value_list> [default <value>]

3.3.1.2 Task section:

This part of the plan file describes the process of executing a single instance of your experiment. There is a number of operations that can be performed in a task including copying files to and from remote nodes; executing an experiment executable; substituting placeholders in the input files with the real values; telling Nimrod/G what to do in case the job fails. As in with parameters, there is a syntax [23] that needs to be followed in order to create a valid task that can be executed by Nimrod/G. Below is a more thorough description of the task operations. Before we go and explain them we have to mention that with some operations (copy and execute) we need to tell Nimrod/G where is the operation performed or where the file locations are while copying. This is done by using 'node:' or 'root:' with the operation. 'node:' tells Nimrod/G that whatever we are trying to do it should be done on the node where the job is to be run and with 'root:' we are telling Nimrod/G that the location is the machine where Nimrod/G is running [23]. Default one is 'root:' so in our practice we almost never use it (as you can see in the example plan file). Now onto the operations:

- **copy** - this operation is used to copy both input and output files. Input files are copied to the computational node where the output files are copied from that node to the root machine. Syntax for this operations is as follows: *copy<source><destination>*

  In the case where you want to copy input files to the computational node from the root machine you would have something like this *copy input_file node:*. This means that the 'input_file' will be copied from root machine to the node and the file name is going stay the same. Other way to do this is to have *copy input_file node:input* where the input_file would be renamed to 'input' This means that the 'input_file' will be copied from root machine to the node and the file name is going to be the same.

- **execute** - This is the task operation that executes some executable. The syntax is *node:execute <user_command>*.

3.3.2 Adding/Creating Experiments

This section of the thesis shows how to add and create an experiment to the database with the plan file as a starting point. There are few commands that need to be executed in order to get everything done. These will be explained shortly but before we did that we just want to quickly explain what needs to be done to have an experiment ready for executing. Besides plan files Nimrod/G has something that we call a 'run file'. Run file is an expanded version of a plan file where all possible combinations of parameters are created and listed in that file. Besides this it is pretty much the same as the plan file. The reason we have done things this is way is because run files can get quite big especially when you might have few thousand combinations of parameters, i.e. few thousand jobs in your experiments. In such cases it would be very difficult for a user to write down the run file manually.

Once the run file is created, you can add it to the database. "Adding" an experiment to the database is just simply adding a single entry to one of the experiment tables saying that we have defined an experiment with a particular name. Whatever that name is will be the name of both plan and run file for that experiment. Note that in order to run your experiment all you need is one or more resources assigned to it.

To convert plan file to run file, execute following Nimrod/G command: *nimrod-generate<planfile_name>*
where, for example, 'planfile_name' would be named 'demo.pln' for an experiment called *demo.* Now, our Generator will find the location of the plan file using its name. For every experiment there is a rule which says that all the relevant files for the experiment called 'exp1' will be stored in the directory *$HOME/.nimrod/experiments/exp1/* and plan file's full path is *$HOME/.nimrod/experiments/exp1/exp1.pln*. The run file created by Generator will have the following path *$HOME/.nimrod/experiments/exp1/exp1.run*

Once you have a run file ready you can add the experiment to the database by running

nimrod portalapi addrun <exp_name> [<exp_type>]

The 'exp_name' option is self-explanatory and it is required. On the other hand 'exp_type' need a bit of elaboration. At the moment there are three different experiment types that Nimrod/G handles: G, O and A [23]. Nimrod/G will behave differently depending on which one you have created. The default value, one that is used if you omit this option, is G. This basically tell Nimrod/G that the experiment is of Nimrod/G type and this is what most of you will be using. 'O' type tells Nimrod/G that the experiment is a Nimrod/O experiments where 'A' signifies ActiveSheets experiment.

3.3.3 Adding/Removing Resource to/from Experiments:

In order to run a experiment you have to have resources that will run jobs that belong to that experiment. This is done by assigning resources to experiments. In this section we assume that the resource is already inserted into the database but it is not assigned to the experiment in questions yet. It only requires you to remember the experiment name and the resource details of a resource you want to add. Once you remembered both of these run the following command to add the resource

nimrod portalapi addserver <exp_name> <resource_details> <res_type>

Removing resources is the same as adding a resource to the experiment except that instead of 'addserver' you will have 'removeserver'. Everything else is the same.

3.3.4 Monitoring experiment execution

This section explains how to start the experiment and then how to monitor its progress. As Nimrod/G uses the Database so that we have Experiment persistence, pausing experiment will not push us backwards as all completed jobs will stay complete. This is how you should start your experiments

nimrod portalapi startexp <exp_name>

As we said before it is quite simple and efficient way to start the experiment. This will start all experiment related modules including agent and jobs schedulers.

Stopping/Pausing your experiment is not much different from starting, except that there are two ways to do this. The first one is to pause the experiment but let all currently running jobs to complete [23]. The second one pauses the experiment but all

currently running jobs will be terminated and their status will be reset to 'ready' (waiting). The syntax for the first case is : *nimrod portalapi stopexp <exp_name>* where the second one has the following syntax *nimrod portalapi shutdownexp <exp_name>*.

Both of these two commands only update NimrodEnfExperiment table entry giving instructions to Nimrod/G modules on how to proceed. This means that some time after you've stopped/shut your experiment you will still see some activity until all modules have completed their shutdown procedures. How long this takes really depends on how much is going on at the time of the shutdown and can take two or more minutes.

Once you have your experiment running all you can do is wait for it to finish, you can occasionally check the experiments progress. So what can you do to see if everything is working fine? The first thing always checks is if jobs are getting started on all the resources. Other things that we can do when monitoring the progress is whether jobs are failing on any machines and if yes is it always the same machine. The section below shows commands that will help you with the monitoring:

NimrodEnfJob - This table [23] holds almost all of the job related information. Data stored here includes where the job is scheduled (if scheduled); what agent is running a particular job; how much cpu and wall time has been consumed by a job; which experiment a job belongs to; job's parameter set; time when job was started and when job was finished; state of the job i.e. whether job is running or waiting to be ran etc.

NimrodEnfExperiment - This table has experiment data: experiment name, path to the directory where experiment files are stored; state of the experiment etc [23].

This chapter gives the details of Nimrod/G broker. It describes what is its Architecture, how the experiment management is done in its environment, and gives us a basic roadmap to run an experiment in the Nimrod/G environment.

Chapter 4

Problem statement

Computational Grids that couple geographically distributed resources such as PCs, workstations, supercomputers, processors, clusters, and scientific instruments, have emerged as a next generation computing platform for solving large-scale problems in science, engineering, and commerce. However resource management and scheduling in these environments continue to be a tedious task. In Grid environment, scheduling is a software framework with which the scheduler collects resource state information, selects appropriate resources, predicts the potential performance for each candidate schedule, and determines the best schedule for the applications to be executed on a Grid system subject to some performance goals.

A Grid scheduling system need to be there for scheduling jobs on resources distributed with varying quality of service requirements. Nimrod-G Resource Broker: an Economic based Grid Scheduler manages all operations associated with remote execution including resource discovery, trading, scheduling based on economic principles and a user defined quality of service requirement. The Nimrod-G resource broker is implemented by leveraging the services provide by Globus and it makes use of deadline and budget based scheduling algorithms like cost optimization, time optimization and cost-time optimization.

The task to simulate the Nimrod/G environment running the parametric sweep applications and doing the performance evaluation has been taken. To demonstrate how the performance is affected by varying the number of jobs submitted to the Nmrod/g Resource Broker or by varying the number of computing resources in Grid environment.

Chapter 5

Implementation Details and Experimental Results

5.1 Nimrod/G 2.0 Installation Steps:

There are number of sequential steps need to be followed for the installation of Nimrod/g. We have to install the PGSQL database, to set the various environment variables, to create the database structure, to make a user of the database, to make a directory where all the applications will be stored, to create a executable file for a particular application and finally execute the run file generated from plan file.

5.1.1 To set the Nimrod/G Environment variables:

There are number of things that you need to define in your environment prior to Nimrod/G installation. Following is the list of environment variables that you need to set in your environment:

- **NIMROD_INSTALL** - This should point to the location where you want Nimrod/G 2.0 installed. We have set the value of NIMROD_INSTALL as:
  NIMROD_INSTALL=/opt/nimrod-3.0.1
  In order to have nimrod executable in path, we also added *$NIMROD_INSTALL/bin* to the *PATH* environment variable
- **NIMROD_DATABASE** - This tells Nimrod/G what interface it should use to communicate with Nimrod/G database. We have set the value *pgsql-bin* for this environment variable as we are using PGSQL database. So we have set the value as:
  NIMROD_DATABASE=pgsql-bin
- **PSQL_LOCATION** - This should point to the location where you have PostgreSQL installed. As PostgreSQL 8.3.1 was already installed so we set the value of this environment variable as:
  PSQL_LOCATION=/usr/local/pgsql/bin
  We have also added to PATH environment variable

- **X.509_USER_PROXY** - As Nimrod/G also uses Globus Toolkit to submit jobs to remote resource; environment variables such as *GLOBUS_LOCATION* need to be set so that globus command line tools can function properly. We have set the value of this variable as:
  X.509_USER_PROXY=/home/vikas/.nimrod/globus_proxy

5.1.3 Installation of Nimrod/G

- Download Nimrod/G 2.0 Binary package
- Untar the package to location /opt.
- Change the directory to the /opt
- Run: *./configure --prefix=$NIMROD_INSTALL*
  NIMROD_INSTALL environment variable should point to the location where Nimrod/G is to be installed.
- Run: *make install*
  This command should be run as a super user if you want to allow all users to see and use Nimrod/G. In other case it can be run by a unprivileged user as long as that user has permission rights to write to *$NIMROD_INSTALL* directory
- Then we gave the command run 'nimrod'. To verify the successful installation of the nimrod. It prints a message telling the version number of the release downloaded:
  Nimrod/G version 3.0.0 - June 2005

5.1.3 Account set up to use nimrod

The Postgresql8.1.2 version was already installed and configured. We use this installed database for executing application on Nimrod/g. The various steps we performed are:

- Create a postgres user named vikas who will manage the experiments using command
  createuser vikas
- Edit the file postgresql.conf and set default_with_oids=on
- Then create nimrod database as vikas user using command *nimrod dbcreate*It create all the tables required for nimrod experiment management

5.1.4 Setting up of Nimrod/G File Structure

Nimrod/G stores most of the experiment related data in the Nimrod/G database. The rest is stored inside a number of files that can be found in a specific location on your file system. To make it easier to locate and use those files Nimrod/G team has set a standard way of creating file/directory structure for every experiment a user may define. The nimrod file structure setup consists of the following actions:

- Every user needs one file. *nimrodrc* and one subdirectory *.nimrod* created in the home directory, i.e. *$HOME/.nimrodrc* and *$HOME/.nimrod* respectively. As login vikas, we run the following commands:

  mkdir /home/vikas/.nimrod

  Inside *$HOME/.nimrod* directory few files and directories may be found including nimrod log files (.log extension) as well as *'experiments'* directory.
- Creating experiments directory inside the .nimrod directory

  '*Experiments*' directory contains a variable number of subdirectories, where each one of those subdirectories is related to one of the experiments user may have had defined. As login Vikas in the directory /home/vikas/.nimrod we give command:

  mkdir experiments
- Then create an experiment called *'even'* all the files related to that experiment will be located at *$HOME/.nimrod/experiments/even* directory. This directory will include the run file and plan file for this experiment Same goes for every other experiment you may have had created.

  These 'experiment directories' are used by Nimrod/G to store and retrieve all the files that are related to those experiments. If Nimrod/G is told to get the results back to the machine where Nimrod/G is running all the result files will be copied to the experiment directory, unless some other location is specified. Input files that may be required by your experiment, should also be placed into that experiment directory. Last thing that can be found in experiment directory are the logs (.log extension) Nimrod/G creates. Only logs related to one particular experiment are going to be placed in that experiment's directory.

5.2 Running the Experiments Using Nimrod-g

Nimrod-g requires the various steps to run the job:

5.2.1 Adding/removing resources to nimrod database:

Resources are needed to be added so that a particular application can use those resources for the execution. The resources are added in the following format:

nimrod resource add <resource_type> <resource_details>

This is the minimum requirement for a command that adds a resource. To elaborate a bit on a format we will explain what the parts are:

nimrod - This is a shell script. Every single Nimrod/G command that is executed requires this to proceed.

resource - This argument basically tells Nimrod/G that the command is a resource command.

add – This is for adding resources to nimrod database so that later on an experiment can use those resources already added

resource_type - This argument tells Nimrod/G what type of resource is being added. Many types of resources can be like globus, fork, pbs, condor etc. So correct type depending upon the resource to be added is need to be mentioned. We have used two type of resources:

- **`Gt4`** - Resources with GT4 Toolkit installed.
- **fork** - This is just a regular UNIX fork. There can be only one fork resource and it is the same machine where you are running Nimrod/G.

resource_details - This argument tells about the host details. In case of globus resources syntax is *hostname/jobmanager-type*

In the case where you might wish to remove the resource completely from the Database, you can do that by executing the following command:
nimrod resource remove <resource_type> <resource_details>
This is same as the add resource. The only difference is use remove instead of add. Once the command is executed that resource will be completely removed from the database. Please note that this command will fail if that resource is still assigned to any of the experiments. This means that you have to remove this resource from all experiments it is used in, before it can be removed from the Database.

The following figure illustrates the same:

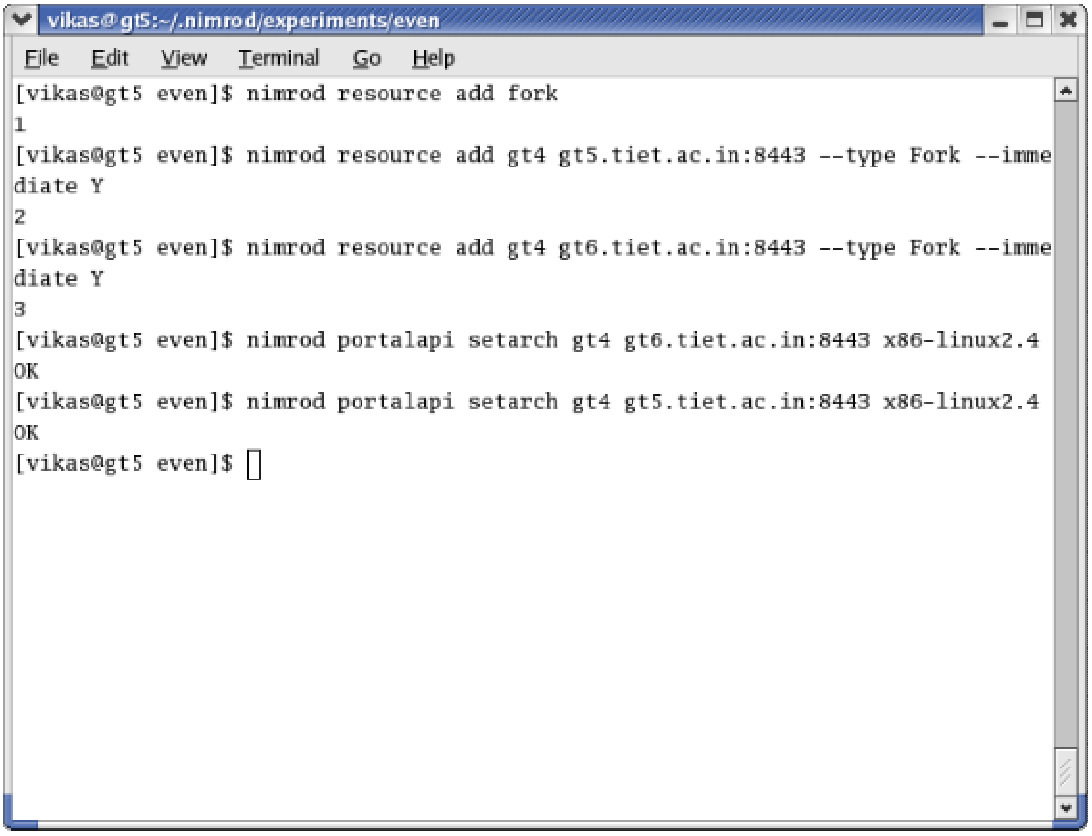

Fig 4.1 Adding resources to nimrod database

After the resources are added they can be checked by querying the Nimrod/G database as shown in the following figure:

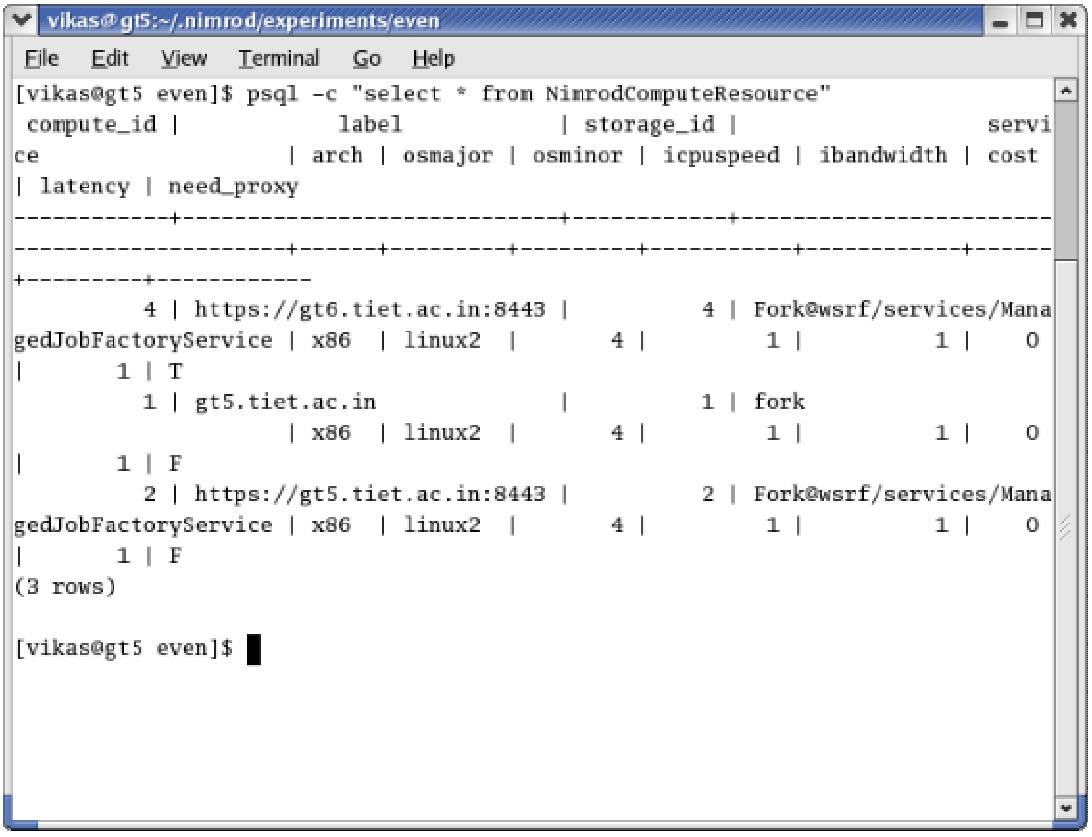

Fig 4.2 Querying Resources Added to Nimrod Database

5.2.2 Creating an experiment

It is done in three steps:

1. Creating an executable file: we have created an executable file for the application in C language as shown in the following figure:

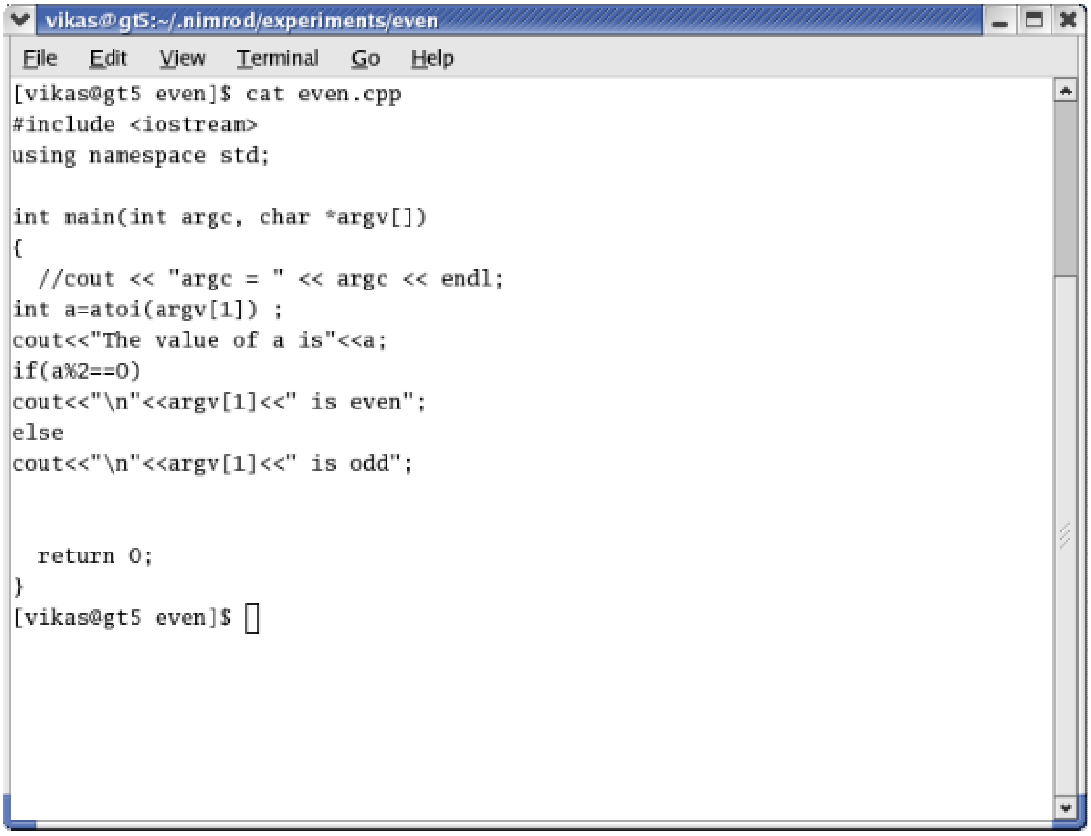

Figure4.3 Executable File for Even Generator Application

2. Creating a plan file: After creating the executable file we created a plan file as shown in the following figure:

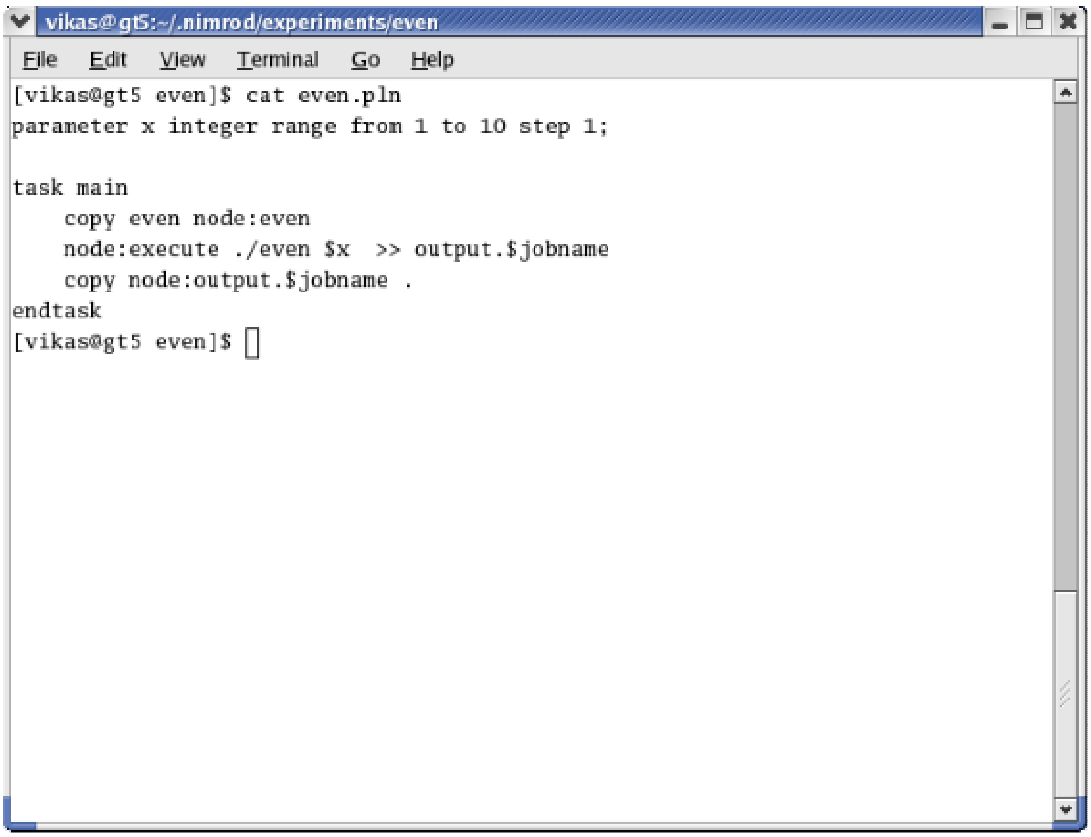

Figure4.4 Plan File for Even Generator Application

3. Creating a run file: After creating the plan file we created the run file from plan file as shown in the figure:

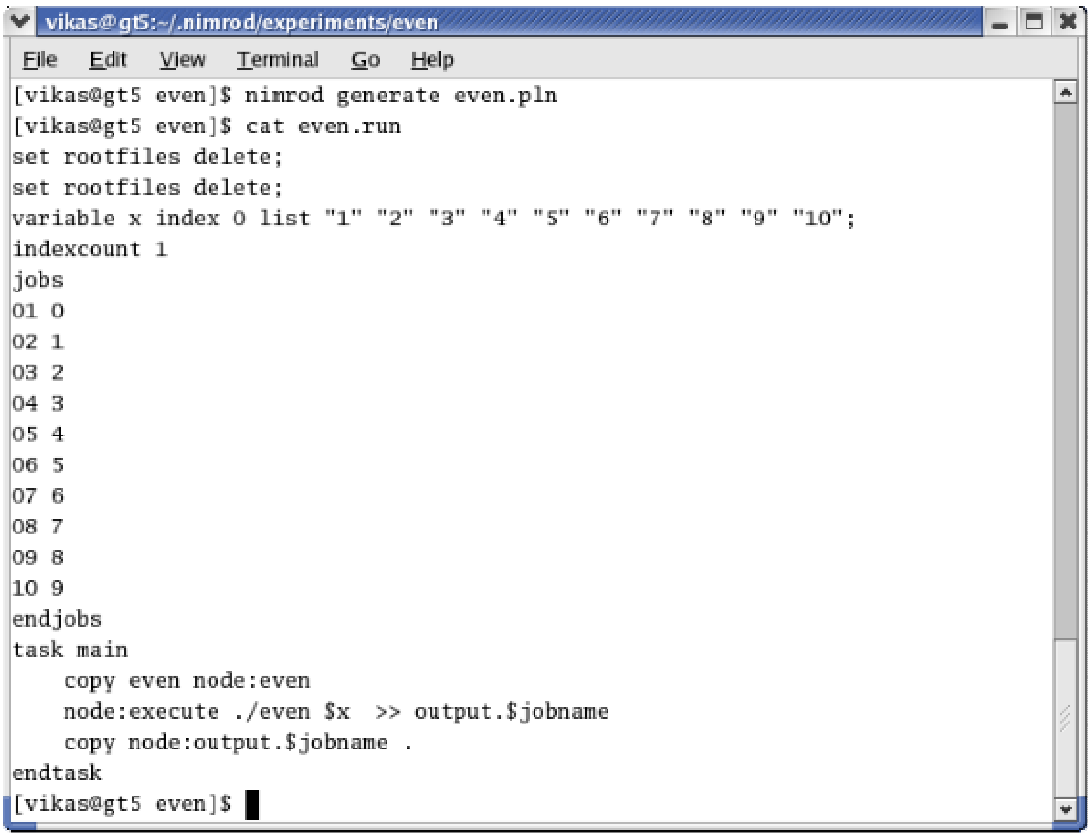

Figure4.5 Run File for Even Generator Application

5.2.3 Assigning resources to the experiment:

What we have done above was merely adding a resource to the database. If things are left like this and the resource we added is not assigned to an experiment it will not be used or even considered for use by that experiment. What this means is that we have to add/assign that resource to an experiment we want to run. The resources are added as shown in the following figure:

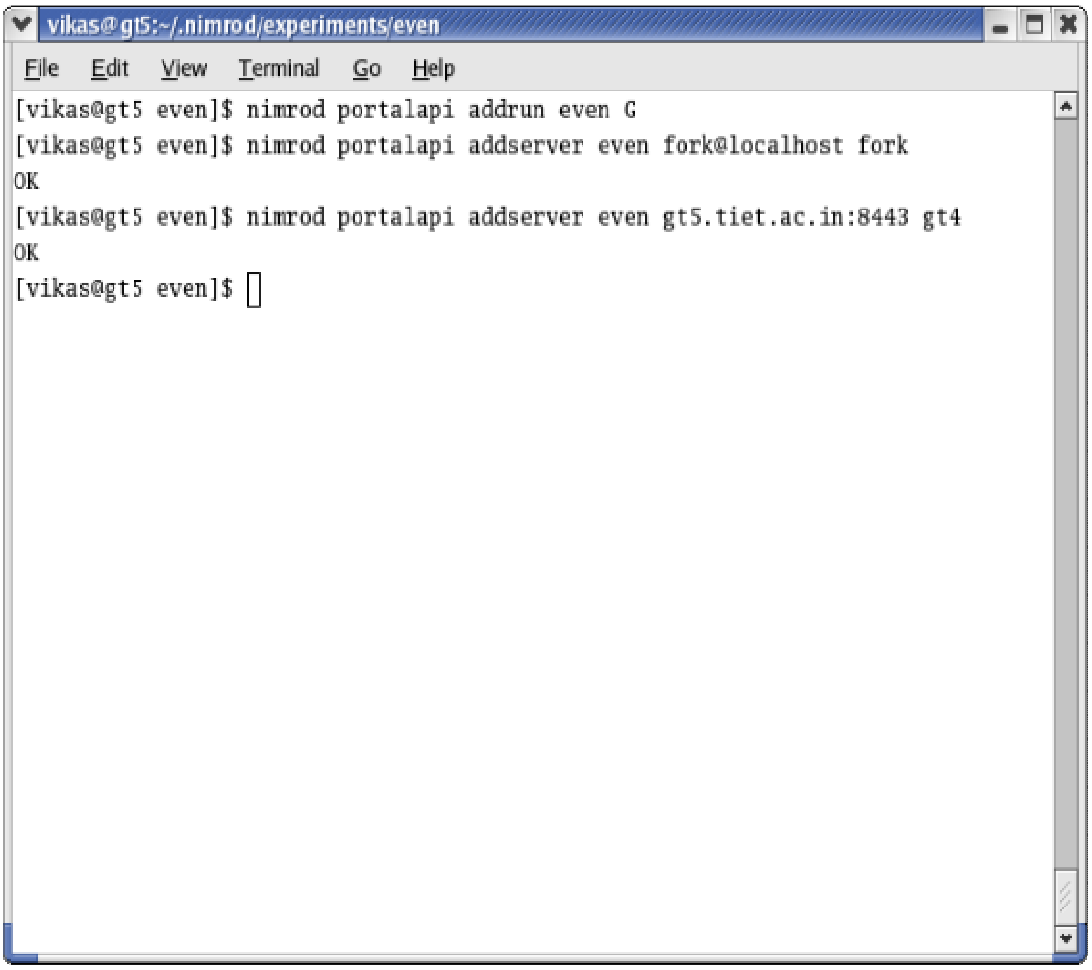

Figure4.6 Resources are added to Even Generator Application

5.2.4 Starting the experiment:

After adding the resources to even generator application we started the experiment as shown in the following diagram:

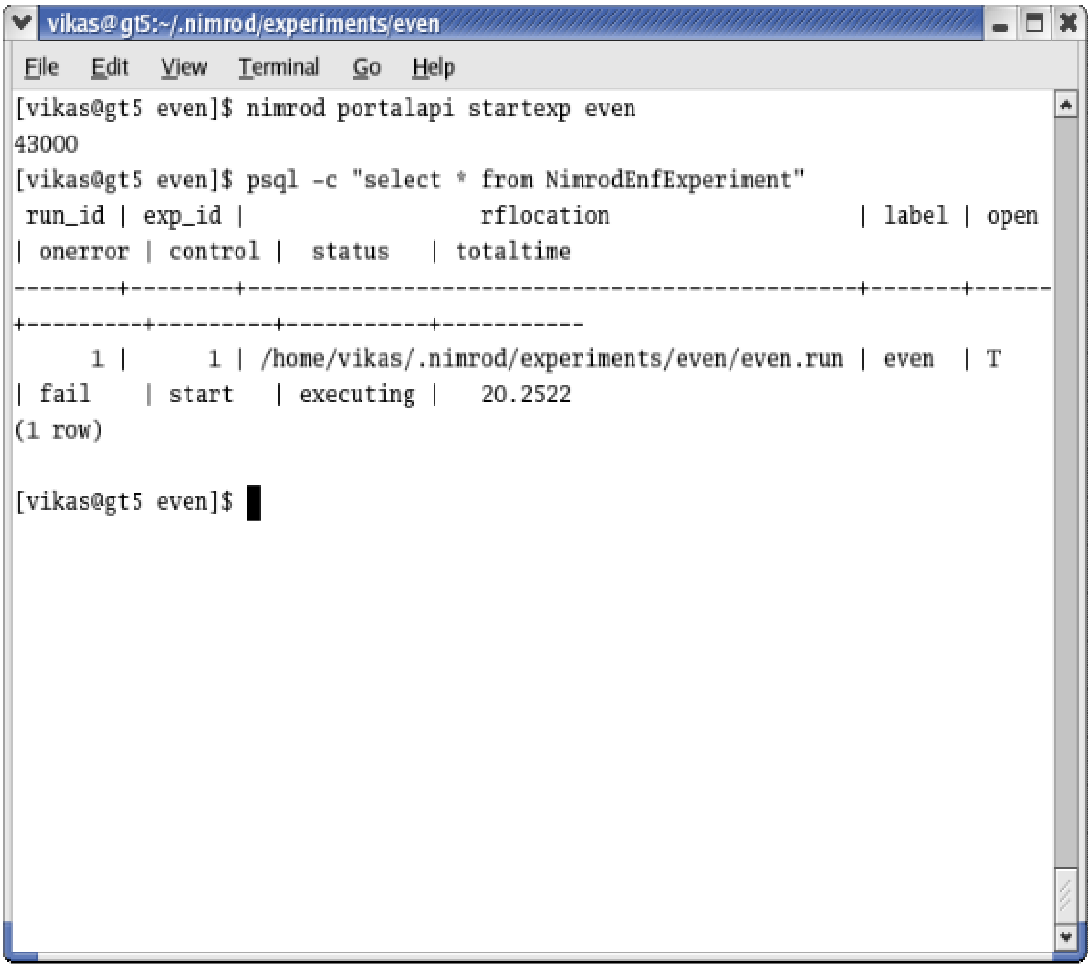

Figure4.7 Starting Even Generator Application

5.2.5 Monitoring the experiment:

Mean while the experiment is executing we can monitor that what is the status of job (ready, executing or done), what is the start time, finish time, cpu time and wait time of a particular job, on which resource a particular job has been scheduled, etc. It is shown in the following diagrams:

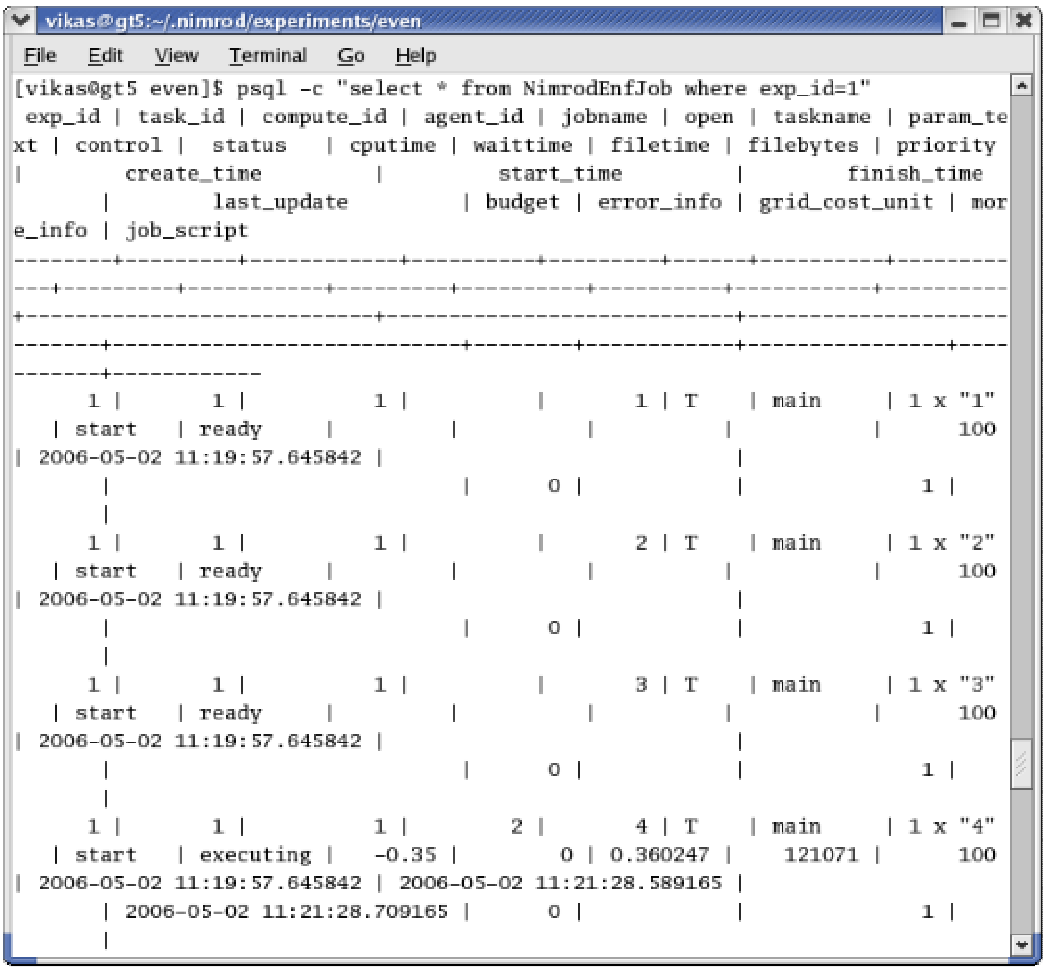

Figure4.8 Monitoring the experiment (a)

```
[vikas@gt5 even]$ psql -c "select * from NimrodEnfJob where exp_id=1"
 exp_id | task_id | compute_id | agent_id | jobname | open | taskname | param_text | control | status | cputime | waittime |
filetime | filebytes | priority |        create_time         |         start_time         |        finish_time         |
   last_update         | budget | error_info | grid_cost_unit | more_info | job_script
--------+---------+------------+----------+---------+------+----------+------------+---------+--------+---------+----------+-
---------+-----------+----------+----------------------------+----------------------------+----------------------------+-----
----------------------+--------+------------+----------------+-----------+------------
      1 |       1 |          2 |        5 |       1 | T    | main     | 1 x "1"    | start   | done   |    0.04 |  8.68184 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:37.38372  | 2006-05-02 11:21:47.455087 | 2006
-05-02 11:21:47.455087 |      0 |            |              1 |           |
      1 |       1 |          2 |        3 |       2 | T    | main     | 1 x "2"    | start   | done   |    0.05 |  8.00877 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:38.999801 | 2006-05-02 11:21:49.074732 | 2006
-05-02 11:21:49.074732 |      0 |            |              1 |           |
      1 |       1 |          1 |        2 |       3 | T    | main     | 1 x "3"    | start   | done   |    0.06 |  7.82832 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:38.907829 | 2006-05-02 11:21:48.97906  | 2006
-05-02 11:21:48.97906  |      0 |            |              1 |           |
      1 |       1 |          1 |        2 |       4 | T    | main     | 1 x "4"    | start   | done   |    0.04 |  8.19732 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:28.589165 | 2006-05-02 11:21:38.639102 | 2006
-05-02 11:21:38.639102 |      0 |            |              1 |           |
      1 |       1 |          1 |        2 |       5 | T    | main     | 1 x "5"    | start   | done   |    0.06 |  8.45248 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:18.179435 | 2006-05-02 11:21:28.357925 | 2006
-05-02 11:21:28.357925 |      0 |            |              1 |           |
      1 |       1 |          2 |        6 |       6 | T    | main     | 1 x "6"    | start   | done   |    0.03 |  8.69653 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:36.773218 | 2006-05-02 11:21:46.864783 | 2006
-05-02 11:21:46.864783 |      0 |            |              1 |           |
      1 |       1 |          2 |        1 |       7 | T    | main     | 1 x "7"    | start   | done   |    0.05 |  8.35146 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:29.815626 | 2006-05-02 11:21:39.868662 | 2006
-05-02 11:21:39.868662 |      0 |            |              1 |           |
      1 |       1 |          2 |        3 |       8 | T    | main     | 1 x "8"    | start   | done   |    0.04 |   8.3092 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:28.625355 | 2006-05-02 11:21:38.760634 | 2006
-05-02 11:21:38.760634 |      0 |            |              1 |           |
      1 |       1 |          2 |        1 |       9 | T    | main     | 1 x "9"    | start   | done   |    0.05 |  8.68831 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:19.115354 | 2006-05-02 11:21:29.296623 | 2006
-05-02 11:21:29.296623 |      0 |            |              1 |           |
      1 |       1 |          2 |        3 |      10 | T    | main     | 1 x "10"   | start   | done   |    0.05 |  8.60149 |
       0 |         0 |      100 | 2006-05-02 11:19:57.645842 | 2006-05-02 11:21:18.255677 | 2006-05-02 11:21:28.406969 | 2006
-05-02 11:21:28.406969 |      0 |            |              1 |           |
(10 rows)
```

Figure4.9 Monitoring the experiment (b)

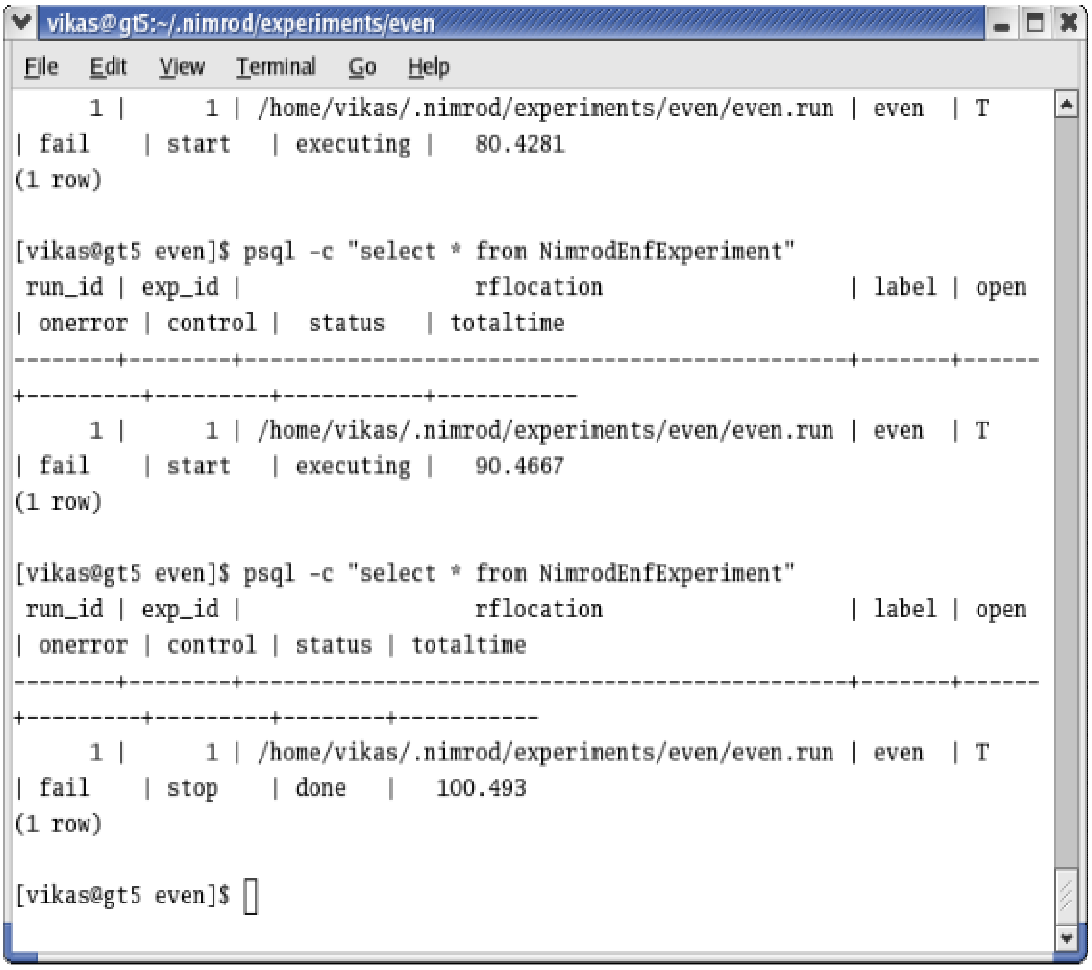

Figure4.10 Monitoring the experiment (c)

5.2.6 Viewing the output:

Meanwhile the output files are created for the corresponding jobs and can be viewed as shown in the following figure:

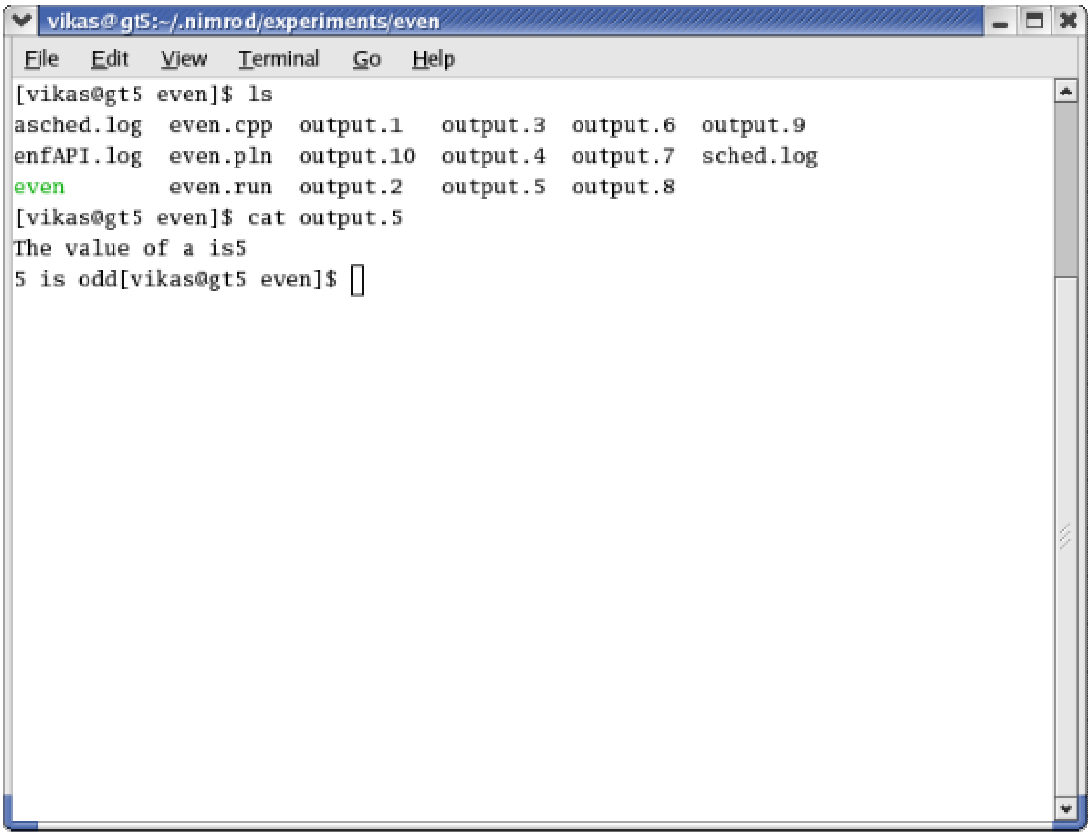

Figure4.11 Viewing the Output of Even Generator Application

5.3 Performance Evaluation

After simulating the environment of Nimrod/g, and run the applications on the environment, we evaluated the performance varying two factors. First is to vary the number of jobs submitted, and second is to vary the number of computing resources The performance evaluation is demonstrated as shown in the following diagram:

5.3.1 Testbed

The testbed is a cluster consisting of three machines (Pentium IV, 3 GHz desktop machines with 512 MB physical memory running Red Hat Linux 9).

Machine Name	IP Address
gt4.tiet.ac.in	172.31.5.106
gt5.tiet.ac.in	172.31.5.104
gt6.tiet.ac.in	172.31.5.197

Out of these machines gt5.tiet.ac.in has Nimrod-G broker Installed on it.

5.3.2 Test Application & Methodology

The test application is the prime generator. The algorithm used allows the computation of the prime numbers specified till the maximum number. The application utilizes the Nimrod specifications.

The test was performed for a range of workloads (generating prime numbers till 10, 20, 30, 40 numbers), each with one to three machines enabled. Execution time was measured as the elapsed clock time for the test program to complete on the Owner node.

5.5.3 Results

Figure shows a bar diagram between number of processors and time (in seconds taken by jobs to complete execution) with varying numbers of jobs.

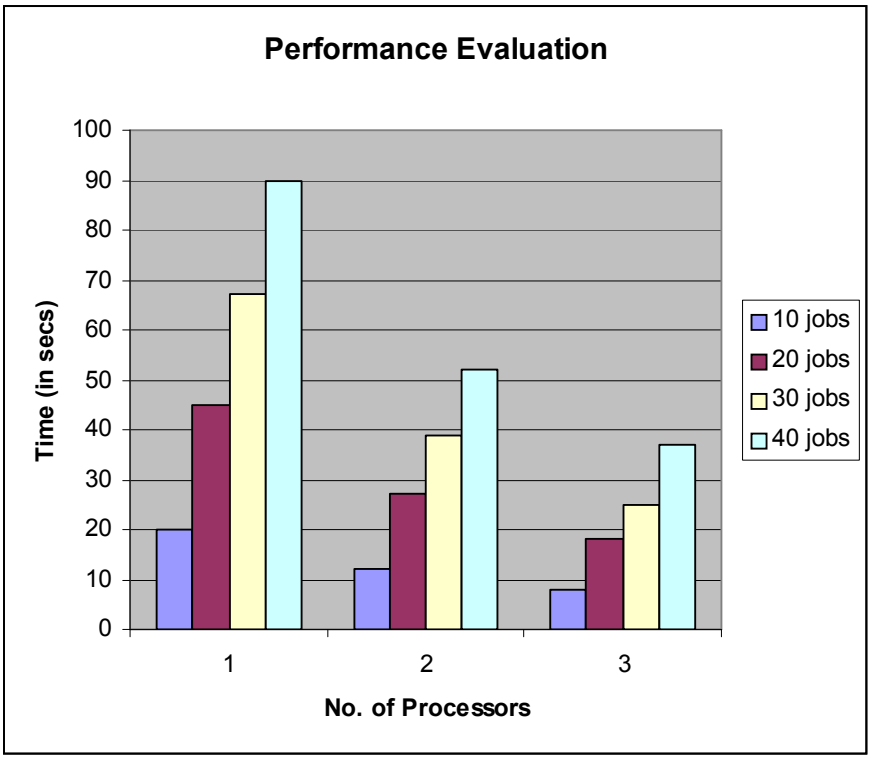

Figure 5.1 Performance Evaluation Results

The performance varies significantly, as the number of jobs increases. The time to generate the prime numbers till 40 number is 39 secs when running on three processors as compared to 90 secs when the jobs are run on single processor.

Chapter 6

Conclusion and Scope for Future Work

6.1 Conclusion

This thesis gives the introduction to Grid Computing, a detailed Grid resource brokering procedure with certain challenges, discusses how to simulate the environment of Nimrod/G Grid resource broker, and finally demonstrates the performance evaluation by varying the number of jobs and number of computing resources.

In particular we can conclude that:

- The evolution of Grid computing from distributed computing leads to effective utilization of idle resources

- The Grid resource management is key to effective utilization of Grid power

- Nimrod/g is an economy based Grid broker which does the complete management of resources by first discovering the resources, trade the resources, scheduling those resources on the submitted jobs, and finally gives back the results with meeting the performance goals

- We have simulated the Nimrod/G environment successfully, run the applications and demonstrated the performance evaluation

6.2 Scope of Future Work

We have simulated the environment of Nimrod/g by leveraging the services provided by Globus. The performance has been evaluated by varying the number of jobs and number of computing resources. This performance evaluation is limited to three-node Grid environment and to a very small experiment having one parameter only. In future we can further expand cluster to the departmental and enterprise level. We can also run the complex experiments having more than one parameters requiring large computations.

Further the nimrod/g provides the various algorithms for the scheduling purpose like Cost Optimization, Time optimization, etc. We can assign the costs to our resources and can do the evaluation by specifying a particular algorithm.

References

[1] B. Bansal, "*Design and Development of Grid Portal*", thesis submitted at Computer Science and Engineering Deptt., TIET Patiala, May 2005

[2] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Girard, R. Strachowski, S. Yu. "*Enabling applications for Grid Computing with Globus*. Redbook", IBM Corp.

[3] Bill Nitzberg1 and Jennifer M. Schopf2, "*Current Activities in the Scheduling and Resource Management"*, Area of the Global Grid Forum www.mcs.anl.gov/~schopf/Pubs/lcns02.pdf

[4] C. Kesselman. I. Foster, "*Computational Grids. In The Grid: Blueprint for a New Computing Infrastructure"*, chapter 2. Morgan-kaufman edition, 1999.

[5] 18 D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sosic, R. Sutherst, N. White, "*The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing*", Australian Computer Science Conference (ACSC 97), Macquarie University, Sydney, Feb 1997.

[6] D. Abramson, P. Roe, L. Kotler, and D. Mather, "*ActiveSheets: Super-Computing with Spreadsheets",* High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference, April 2001.

[7] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "*Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations"*, Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing, Virginia, August 1995, IEEE CS Press, USA, 1995.

[8] Franccois Grey, Matti Heikkurinen, Rosy Mondardini, "*Grid Cafe: A Palace for Everybody to Learn About Grid*", http://Gridcafe.web.cern.ch/Gridcafe/Gridwork/architecure.html

[9] Franccois Grey, Robindra Prabhu, Rosy Mondardini "*Grid Cafe: A Palace for Everybody to Learn About Grid"*, http://Gridcafe.web.cern.ch/Gridcafe/challenges/challanges.html

[10] Jospeh Joshy, Fellenstine Craig, "*Grid Computing*", Pearson's Education India

[11] Jean-Christophe Durand, "*Grid Computing: A Conceptual and Practical Study*", http://www.hec.unil.ch/cms_inforge/Durand.pdf

[12] I. Foster and C. Kesselman, editors. "*The Grid: Blueprint for a Future Computing Infrastructure",* Morgan Kaufmann Publishers, 1998.

[13] Mark Baker, Rajkumar Buyya, and Domenico Laforenza, "*Grids and Grid technologies for wide-area distributed computing*", www.Gridbus.org/papers/Gridtech.pdf

[14] Rajkumar Buyya, "*The Nimrod-G Resource Broker: An Economic based Grid Scheduler*", http://www.buyya.com/thesis/Gridbroker.pdf

[15] R. Buyya, D. Abramson, and J. Giddy, "*Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid"*, The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), May 2000, Beijing, China, IEEE Computer Society Press, USA.

[16] R. Buyya, D. Abramson, and J. Giddy, "*An Economy Driven Resource Management Architecture for Global Computational Power Grids*", Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), June 26-29, 2000, LasVegas, USA, CSREA Press, USA, 2000.
www.csse.monash.edu.au/~davida/papers/GridEconomy.pdf

[17] R. Buyya, D. Abramson, and J. Giddy, "*A Case for Economy Grid Architecture for Service-Oriented Grid Computing"*, Proceedings of the International Parallel and Distributed Processing Symposium:10th IEEE International Heterogeneous Computing Workshop (HCW 2001), April 23, 2001, SanFrancisco, California, USA, IEEE CS Press, USA, 2001.

[18] R. Buyya, J. Giddy, D. Abramson, "*An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications"*, Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000), Kluwer Academic Press, August 1, 2000, Pittsburgh, USA.

[19] Schopf, J. M., "*Super Scheduler Steps/Framework*", Global Grid Forum Scheduling Area Working Document SchedWD 8.5, http://www-unix.mcs.anl.gov/~schopf/ggf-sched/WD/schedwd.8.5.pdf

[20] Schopf, J. M., "*A General Architecture for Scheduling on the Grid*", submitted to special issue of the Journal of Parallel Distributed Computing on Grid Computing, 2002. http://www-unix.mcs.anl.gov/~schopf/Pubs/sched.arch.jpdc.pdf

[21] TurboLinux, "*Using Application Programming Interface*", Chapter 9, EnFuzion Manual, 2002. http://www.csse.monash.edu.au/cluster/enFuzion/api.htm

[22] Yanmin ZHU, "*A Survey of Grid Scheduling*", Department of Computer Science Hong Kong University of Science and Technology ihome.ust.hk/~zhuym/research/publications/QualifyingExam/QualifyingSurvey.pdf

[23] www.csse.monash.edu.au/~nimrod/nimrodg/ng.html

Paper Accepted/Communicated/Published

1 Vikas Mangla, Meena Gupta, Seema Bawa **"Resource Scheduling in Grid Enviornment"**, National Conference on Computer Science and Information Technology (NCCSIT-2006), Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, India, March 18-19, 2006. **[Published in Conference Proceedings].**

2 Vikas Mangla, Seema Bawa **"Resource Scheduling in Grid Enviornment"**, the third International Conference on Ubiquitous Intelligence and Computing (UIC-06) organized by Huazhong University of science and Technology (HUST), Co-Sponsored by HUST, NSFC, 863, ChinaGrid, IFIP in Cooperation with the IEEE Computer Society Wuhan and Three Gorges, China, September 3-6, 2006 **[Communicated]**