Resource Provisioning in Alchemi .Net

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

Master of Engineering
in
Software Engineering

submitted by:
Parvinder Singh
(Roll No. 8053115)

Under the supervision of:
Ms. Inderveer Chana
Senior Lecturer
CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

MAY 2007

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Resource Provisioning in Alchemi .Net"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Inderveer Chana.

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

(Parvinder Singh)

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)
Supervisor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr.(Mrs) Seema Bawa)
Professor &Head
Computer Science & Engineering Department
Thapar University,
Patiala.

(Dr. R.K. Sharma)
Dean
Academic Affairs
Thapar University,
Patiala

Acknowledgement

I wish to express my deep gratitude to Ms. Inderveer Chana, Senior Lecturer, Computer Science & Engineering Department for providing her uncanny guidance and support throughout the preparation of the thesis report.

I am thankful to Dr. (Mrs.) Seema Bawa, Head, and Ms. Shivani Goel, P.G. Coordinator, Computer Science & Engineering Department, for the motivation and inspiration that triggered me for the thesis work.

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

I would also like to thank all the staff members, T.I.E.T. Grid Group and all my friends especially Yugma and Manmeet who were always there at the need of the hour and provided all the help and facilities, which I required for the completion of the thesis.

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.

Parvinder Singh
(8053115)

Abstract

Grid systems and applications aim to integrate, virtualize, and manage resources and services within distributed, heterogeneous, dynamic "Virtual Organizations". It offers untapped processing cycles from networks of computers spanning vast geographical boundaries. Grid is presented to the user as a single, unified resource.

Grid Resource management is a key concern for implementing effective Grid middleware and shielding application developers from low level details. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. Grid Resource Management involves, resource discovery, information gathering about resources and selection of the best resource and finally job execution. Job execution includes a process of selecting appropriate resources, reserving those resources and finally mapping application for execution onto those resources known as Resource Provisioning.

Resource Provisioning system is a system for creating and managing multiple instances of a utility service within a shared IT (information technology) infrastructure. A service provider maintains an aggregation of resources that can be allocated to different services. Customers request access to services of particular types, and instances of these services are provisioned to meet their needs. In creating the provisioning, requests are made to resource managers (RMs) that keep track of certain types of resources, their allocations, and availabilities. The RM provides resources, which are appropriately configured to meet the functionality and performance requirements of the particular service being supported.

In this thesis work, a technique for registration, automatic deployment and on-demand provisioning of application components (activities) i.e. scavenging of idle resources that can be used to build Grid applications for Alchemi .Net has been proposed. This technique helps to implement provisioning application as a web-service that also helps to decouple Grid application from any particular middleware APIs.

Table of Contents

List of Figures

Chapter 1
Introduction

This chapter gives an introduction to grid computing, problems found in resource management and the solution, motivation behind this thesis and the organization of the thesis along with a brief idea about the contents of each of the following chapters.

1.1 What is Grid Computing?

In recent times, rapid advances in networking, hardware, and middleware technologies are facilitating the development and deployment of complex applications, such as large-scale distributed, collaborative scientific simulation, analysis of experiments in elementary particle physics, distributed mission training and virtual surgery for medical instruction. These predominantly collaborative applications are characterized by their very high demand for computing, storage and network bandwidth requirements [1] which can be fulfilled by Grid Computing.

Grid computing [1] involves using many resources (compute, data, I/O, instruments, etc.) to solve a single, large problem that could not be performed on any single resource. Grid computing requires the use of specialized middleware to mitigate the complexity of integrating of distributed resources within an Enterprise or as a public collaboration. To manage these resources a Grid Resource Management System is required which deals with the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible. Grid applications compete for resources that are very different in nature, including processors, data, scientific instruments, networks, and other services. Grid Resource Management involves many players and possibly several different layers of schedulers. At the highest level are Grid-level schedulers that may have a more general view of the resources but are very "far away" from the resources where the application will eventually run. At the lowest level is a local resource management system that manages a specific resource or set of resources.

Effective Grid computing is possible, only if the resources are scheduled well. Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites Grid scheduling involves three main phases:

- Resource discovery, which generates a list of potential resources;
- Information gathering about those resources and selection of a best set;
- And job execution, which includes selection of appropriate resources, reserving those resources and finally mapping application for execution onto those resources along with file staging and cleanup.

1.2 Motivation

The first requirement for successful sharing of resources in a Grid is an efficient discovery mechanism for locating the desired resources available in the system at the time of request, however, due to the dynamic, heterogeneous and distributed nature of the Grid environment, resource discovery is a difficult job. Grid resources are potentially very large in number and variety, individual resources are not centrally controlled, and they can enter and leave the grid systems at any time. For these reasons, resource management in large-scale grids can be very challenging. Resource management mainly deals with the problem of locating and querying information about useful Grid resources.

Existing grid infrastructure middleware, such as Globus, ICENI, and Legion, offer simplified application programming interfaces (APIs) for deploying grid applications. However, grid applications using these APIs become tightly coupled to their respective middleware infrastructure creating an impediment to interoperability, portability, maintenance and extensibility. In other words existing grid infrastructure middleware offer only the means and not the solutions for reserving and securely accessing resources. Thus, the onus of actually reserving and provisioning these different resources while also ensuring end-to-end QoS still lies on the grid applications. These low-level concerns increase the accidental complexities incurred developing complex Grid applications.

1.3 Organization of Thesis

The chapters in the thesis are organized as follows:

Chapter 2 describes in detail what grid computing is, how it evolved from distributed computing, and various types of grid systems, grid architecture, components of grid and benefits of grid.

Chapter 3 describes the resource provisioning process and various approaches and algorithms used by this process.

Chapter 4 discusses the problems found in the approaches to resource management and the possible solutions to these problems.

Chapter 5 includes the implementation details of the proposed architecture and experimental setup.

Chapter 6 includes the implementation details of the provisioning application and along with web service.

Chapter 7 summarizes the work presented in this thesis followed by the features that can be incorporated in future for the enhancement of the provisioning and overall resource management.

Chapter 2
Grid Computing

This chapter describes grid computing in detail. It explains the evolution of grid computing from distributed computing, various kinds of grids based on the kind of service they provide, different topologies of the grid depending on the number of organizations that are a part of a particular grid environment and benefits of a grid environment.

2.1 Background

Grid computing is the next major revolution in information technology after the advent of the Internet. The Grid word comes from the Power Grid (Electricity). The ancestor of the Grid is Metacomputing [2]. 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), was demonstrated at Supercomputing 1995. This project strongly influenced the subsequent Grid computing activities. 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 linked 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 goal of grid computing is to realize a persistent, secure, efficient, standards-based service infrastructure that enables coordinated sharing of autonomous and geographically distributed hardware, software, and information resources in dynamic, multi-institutional virtual organizations [3]. The sharing in this context is not primarily file exchange, as supported by the Web or peer-to-peer systems, but rather direct access to computers, software, data, services, and other resources, as required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is

allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what is called a Virtual Organization (VO). The Grid is "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources—what we refer to as virtual organizations." Ian Foster et al, 2001.

A Virtual Organization [4] can be defined as an organization providing virtual resources in a way that the users or customers do not have to know the inner goal, structure, and rules of the organization, but inside the organization a clear manageable functional and process organization exists. The management inside the organization is not in conflict with the dynamism of its elements. An ideal virtual organization would provide ideal transparent utilization of their resources.

2.2 About Grid Computing

According to Foster [4],

"Grid computing strive to aggregate diverse, heterogeneous, geographically distribute and multiple-domain-panning resource to provide a platform for transport, secure, coordinated, and high-performance resource-haring and problem solving**"**

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 .**"**

They talked about on-demand access to computing, data, and services In a subsequent article, "The Anatomy of the Grid," co-authored with Steve Tuecke in 2000, they refined the definition to address social and policy issues, stating that Grid computing is concerned with "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations."

The key concept is the ability to negotiate resource-sharing arrangements among a set of participating parties (providers and consumers) and then to use the resulting resource pool for some purpose. It all started with Distributed computing [5]. The development of Grid computing technology that allows resource sharing across multiple domains is driven by the need for large-scale high performance distributed computing. A distributed

system consists of a set of software agents that work together to implement some intended functionality. Because the agents in a distributed system do not operate in a uniform processing environment, they must communicate by protocol stacks that are intrinsically less reliable than direct code invocation and shared memory. Grid-computing principles focus on large-scale resource sharing in distributed systems in a flexible, secure, and coordinated fashion. This dynamic coordinated sharing results in innovative applications making use of high-throughput computing for dynamic problem solving.

The basic idea behind the Grid is sharing computing power. Nowadays most people have more than enough computing power on their own PC, hence sharing is unnecessary for most purposes. However back in the sixties and seventies, sharing computer power was essential. At that time, computing was dominated by huge mainframe computers, which had to be shared by whole organizations. A number of applications needed more computing power than can be offered by a single resource or organization in order to solve them within a feasible/reasonable time and cost. This promoted the exploration of logically coupling geographically distributed high-end computational resources and using them for solving large-scale problems. Such emerging infrastructure was called computational grid, and led to the popularization of a field called Grid Computing.

2.3 Grid Systems Taxonomy

Depending upon the design objectives and target applications for particular Grid environment grid systems can be classified into three categories as shown in figure 2.1. The design objective of a grid system can be any one or a combination of two or more of the following.

- improving application performance,
- improving data access, and
- enhanced services.

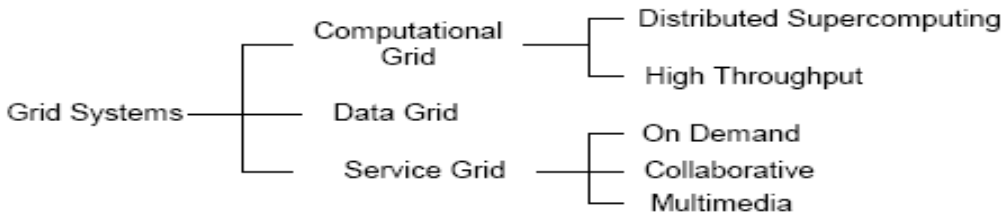

Figure 2.1 Grid Systems Taxonomy [6]

The **computational Grid** [6] category denotes systems that have a higher aggregate computational capacity available for single applications than the capacity of any constituent machine in the system. These can be further subdivided into distributed supercomputing and high throughput categories depending on how the aggregate capacity is utilized.

- A **distributed supercomputing** Grid executes the application in parallel on multiple machines to reduce the completion time of a job. Typically, applications that require a distributed supercomputer are grand challenge problems. Examples of grand challenge problems are fluid dynamics, weather modeling, nuclear simulation, molecular modeling, and complex financial analysis.
- A **high throughput Grid** increases the completion rate of a stream of jobs. Applications that involve parameter study to explore a range of possible design scenarios such as ASIC or processor design verification tests would be run on a high throughput Grid.

The **data Grid** [6] category is for systems that provide an infrastructure for synthesizing new information from data repositories such as digital libraries or data warehouses that are distributed in a wide area network. Computational Grids also need to provide data services but the major difference between a Data Grid and a computational Grid is the specialized infrastructure provided to applications for storage management and data access. In a computational Grid the applications implement their own storage management schemes rather than use Grid provided services. Typical applications for these systems would be special purpose data mining that correlates information from multiple different data sources and then processes it further. The two popular DataGrid

initiatives, European DataGrid [7] and Globus [2], are working on developing large-scale data organization, catalog, management, and access technologies.

The **Service Grid** [6] category is for systems that provide services that are not provided by any single machine. This category is further subdivided in on demand, collaborative, and multimedia Grid systems.

- A **Collaborative Grid** connects users and applications into collaborative workgroups. True collaboration will enable organizations to work on common problems regardless of their geographic locations. These systems enable real time interaction between humans and applications via a virtual workspace.
- A **Multimedia Grid** provides an infrastructure for real-time multimedia applications. This requires supporting quality of service across multiple different machines whereas a multimedia application on a single dedicated machine can be deployed without QoS.
- An **On demand Grid** facilitates access to rarely used resources for short periods of time. While many organizations and individuals would benefit from resources such as a supercomputer or specialized instrumentation equipment, their infrequent usage pattern would fail to justify the procurement of such a resource. Grid services can allow seamless use of these resources, regardless of the given domain.

2.4 Grid Topologies

There are three topologies of grid namely, Intragrid, Intergrid and Extragrid. The simplest of these is the intragrid [8], which is comprised merely of a basic set of grid services within a single organization. The complexity of the grid design is proportionate to the number of organizations that the grid is designed to support, and the geographical parameters and constraints. As more organizations join the grid, the non-functional or operational requirements for security, directory services, availability, and performance become more complex. The topological view is shown in figure 2.2.

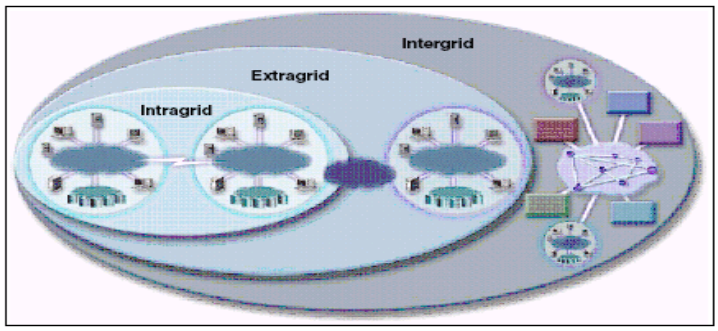

Figure 2.2. Topological view of Grids [8]

2.4.1 Intragrid

- There is a single organization.
- Partner integration is not there.
- Only one single cluster

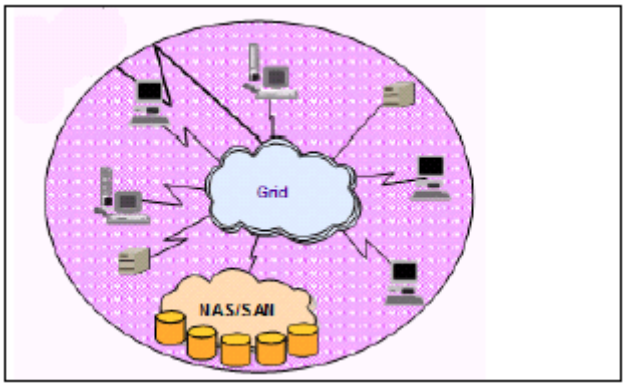

Figure 2.3. Intragrid [8]

2.4.2 Extragrid

- There are multiple organizations.
- Partner integration is there.
- Multiple clusters

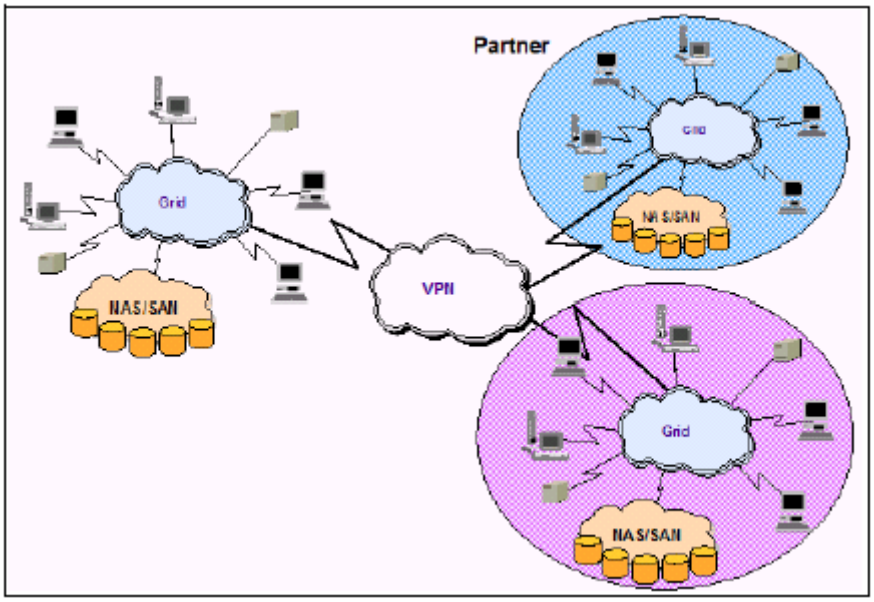

Figure 2.4. Extragrid [8]

2.4.3 Intergrid

- There are many organizations.
- There are multiple partners
- Many multiple clusters

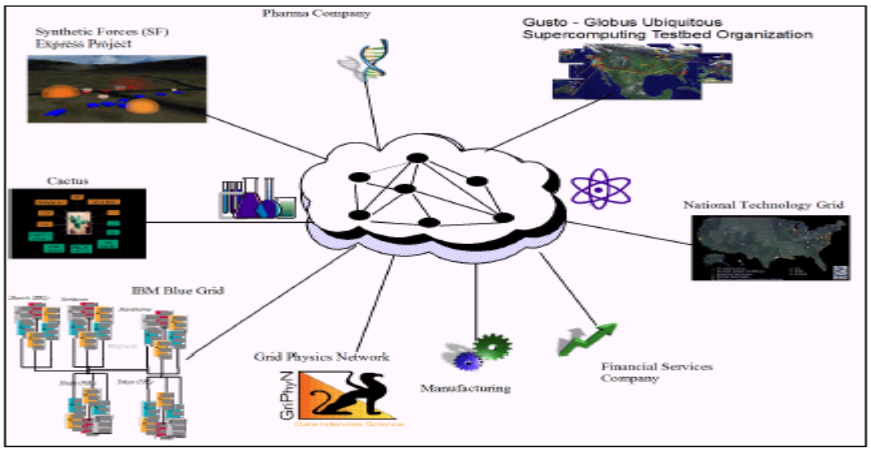

Figure 2.5 Intergrids [8]

2.5 The Grid Architecture

The Grid architecture [9] identifies the basic components of a grid system. It defines the purpose and functions of its components, while indicating how these components interact

with one another. The main focus of the grid architecture is on interoperability among resource providers and users in order to establish the sharing relationships. This interoperability, in turn, necessitates common protocols at each layer of the architectural model, which leads to the definition of a grid protocol architecture as shown in the following figure.

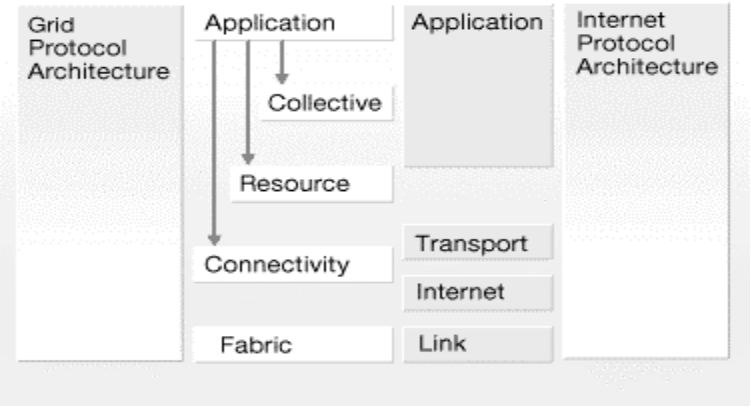

Figure 2.6 The layers of the grid Architecture and its relationship to the Internet Protocol Architecture [9]

This protocol architecture defines common mechanisms, interfaces, schema, and protocols at each layer, by which users and resources can negotiate, establish, manage, and share resources. Figure 2.6 shows the component layers of the grid architecture and the capabilities of each layer. Each layer shares the behavior of the underlying component layers. The following describes the core features of each of these component layers, starting from the bottom of the stack and moving upward.

Fabric layer—The fabric layer [9] defines the interface to local resources, which may be shared. This includes computational resources, data storage, networks, catalogs, software modules, and other system resources.

Connectivity layer—The connectivity layer [9] defines the basic communication and authentication protocols required for grid-specific networking-service transactions.

Resource layer—This 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 [9] 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.

Collective layer—While the resource layer manages an individual resource, the collective layer [9] 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.

Application layer—The application layer [9] enables the use of resources in a grid environment through various collaboration and resource access protocols.

2.6 Grid Resources

A Grid is a collection of machines, sometimes referred to as nodes, resources, members, donors, clients, hosts, engines, and many other such terms [10]. They all contribute any combination of resources to the grid as a whole. All users of the grid may use some resources, while others may have specific restrictions.

2.6.1 Computation

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are three primary ways to exploit the computation resources of a grid. The first and simplest is to use it to run an existing application on an available machine on the grid rather than locally. The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors. The third is to run an application, which needs to be executed many times, on many different machines in the grid. *Scalability* is a measure of how efficiently the multiple processors on a grid are used. If twice as many processors make an application complete in one half the time, then it is said to be perfectly scalable. However, there may be limits to scalability when applications can only be split into a limited number of separately running parts or if those parts experience some other interdependencies such as contention for resources of some kind.

2.6.2 Storage

The second most common resource used in a grid is data storage. A grid providing an integrated view of data storage is sometimes called a *data grid*. Each machine on the grid usually provides some quantity of storage for grid use, even if temporary. Storage can be memory attached to the processor or it can be *secondary storage*, using hard disk drives or other permanent storage media. Memory attached to a processor usually has very fast access but is volatile. It would best be used to cache data or to serve as temporary storage for running applications.

Secondary storage in a grid can be used in interesting ways to increase capacity, performance, sharing, and reliability of data. Many grid systems use mountable networked file systems, such as Andrew File System (AFS), Network File System (NFS), Distributed File System (DFS, or General Parallel File System (GPFS). These offer varying degrees of performance, security features, and reliability features.

Using the storage on multiple machines with a unifying file system can increase capacity. Any individual file or database can span several storage devices and machines, eliminating maximum size restrictions often imposed by file systems shipped with operating systems.

2.6.3 Communications

The rapid growth in communication capacity among machines today makes grid computing practical, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, it should not be a surprise that another important resource of a grid is data communication capacity. This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed, and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid.

2.6.4 Software and licenses

The grid may have software installed that may be too expensive to install on every grid machine. Using a grid, the jobs requiring this software are sent to the particular machines

on which this software happens to be installed. When the licensing fees are significant, this approach can save significant expenses for an organization. Some software licensing arrangements permit the software to be installed on all of the machines of a grid but may limit the number of installations that can be simultaneously used at any given instant. License management software keeps track of how many concurrent copies of the software are being used and prevents more than that number from executing at any given time. The grid job schedulers can be configured to take software licenses into account, optionally balancing them against other priorities or policies.

2.6.5 Special equipment, capacities, architectures, and policies

Platforms on the grid will often have different architectures, operating systems, devices, capacities, and equipment. Each of these items represents a different kind of resource that the grid can use as criteria for assigning jobs to machines. While some software may be available on several architectures, for example, PowerPC and x86, such software is often designed to run only on a particular type of hardware and operating system. Such attributes must be considered when assigning jobs to resources in the grid. In some cases, the administrator of a grid may create a new artificial resource type that is used by schedulers to assign work according to policy rules or other constraints. For example, some machines may be designated to only be used for medical research. These would be identified as having a medical research attribute and the scheduler could be configured to only assign jobs that require machines of the medical research *resource*.

2.7 Resource Management

Grid systems are inter connected collections of heterogeneous and geographically distributed resource harnessed together to satisfy various needs of the users. Resource Management [13] is central component of a grid system. It involves managing resources in the system. Its basic responsibility is to accept requests from users, match user requests to available resources for which the user has access and schedule the matched resources. Resource management involves security, fault tolerance along with scheduling. It is the manner in which resources are allocation, assigned, authenticated, authorized, assured, accounted, and audited. Resources include traditional resources like compute cycles,

network bandwidth, space or a storage system and also services like data transfer, simulation etc.

2.7.1 Issues in Grid Resource Management Systems (GRMS)

Resource Management in Grid systems is made complex due to various factors like site autonomy, resource heterogeneity etc. Various factors contributing to this complexity are[13]:

- **Site autonomy**: Grid resource management system should preserve site autonomy. In Grid systems resources are distributed across separate administrative domains, which results in resource heterogeneity, differences in usage, scheduling policies, security mechanisms.

- **Co-allocation:** Co-allocation is the problem of allocating resources in different sites to an application simultaneously. Different administrative domains employ different local resource managements systems like NQE, LSF etc. A grid resource management system should be able to interface and interoperate with these local resource managements.

- **Negotiation:** In grid system resources are added and removed dynamically. Different types of applications with different resource requirements are executed. Resource owners set their own resource usage policies and costs. This necessitates a need for negotiation between resource users and resource providers.

In a grid different type of applications from a wide range of domains are executed each with different resource management requirements. While some type of applications require to be scheduled as soon as possible even if it means reduced performance, some class of application need high performance. The resource management technique should allow new policies to be incorporated into it without requiring substantial changes to the existing code. In addition to the above a resource management system for the grid should address the other issues like flexibility and extensibility, global name space, security, fault tolerance and Quality of service.

2.7.2 Functions of GRMS

In Grid, Resource Management is a complex processes that involve resources over different administrative domain. The steps involved in execution of a job are[13]:

- Initially, the end user submits to the Resource Management System (RMS) the job to be executed along with some constraints like job execution deadline, the maximum cost of execution.
- In next step, Resource Management takes the job specification and from it estimates the resource requirements like the number of processors required, the execution time, and memory required.
- After estimating the resource requirements RMS discovers available resources and provision these resources for job execution.
- Finally schedule the jobs on these resources by interacting with the local resource management system.

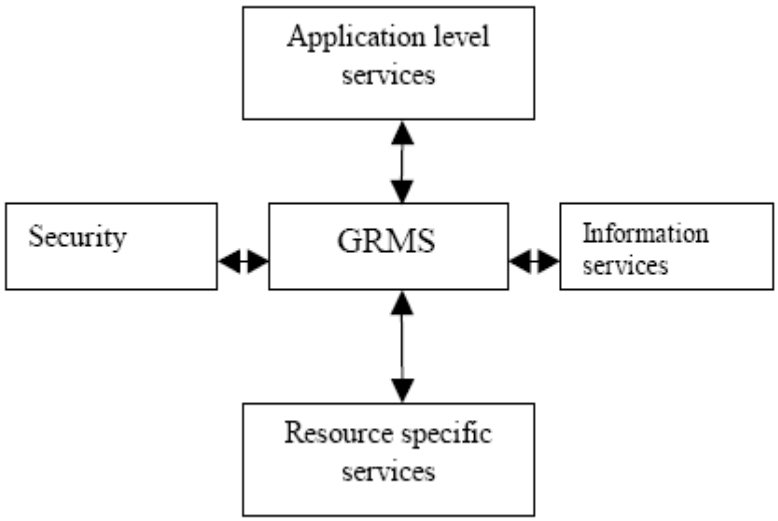

Figure 2.7. Resource Management System [13]

A RMS is also responsible for naming the resources in the system, monitoring and reporting the job, resource status and accounting for resource usage. The RMS interacts with the security system to validate user requests, the information service to obtain information about resource availability, the local system to schedule jobs on the local resource management system

This chapter described in detail what grid computing is, evolution of grid computing from distributed computing, various kinds of grids based on the kind of service they provide, different topologies of the grid depending on the number of organizations that are a part of a particular grid environment, grid resources ,benefits of grid and grid resource management system .

2.8 Benefits of Grid Computing

Grid Computing is a vast and active research area, which encompasses various aspects of distributed computing such as seamless access to, distributed data (Data Grids), and collaborative distributed environments (Service Grids). This pattern focuses on Grids for running computationally intensive applications (Computational Grids), focusing on Grids that leverage the idle capacity of commodity workstations.

Using the idle periods of available computing resources can greatly increase the amount of computing power available to users of an organization. A software infrastructure (Middleware) that allows the use of these shared computing resources must address aspects such as application deployment, distributed scheduling, provisioning, collection of execution results, fault-tolerance, and security. This section describes the benefits of a grid environment in detail.

2.8.1 Exploiting underutilized resources

In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5 percent of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides a technique for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

2.8.2 Parallel CPU capacity

The potential for massive parallel CPU capacity [10] is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others.

2.8.3 Collaboration of virtual resources

In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources. The users of the grid can be organized dynamically into a number of virtual organizations [11, 12], each with different policy requirements. These virtual organizations can then share their resources collectively as a larger grid.

2.8.4 Access to additional resources

In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity. For example, if a user needs to increase his total bandwidth to the Internet to implement a data mining search engine, the work can be split among grid machines that have independent connections to the Internet.

2.8.5 Reliability

Grid provides reliability in terms of failure at one location; the other parts of the grid are not likely to be affected. Grid management software can automatically resubmit jobs to other machines on the grid when a failure is detected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the grid. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering; thus offering much more reliability.

2.8.6 Resource Balancing

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid-enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization. This feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization.

The next chapter discusses resource provisioning from resource user and resource providers point of view respectively, issues and models of resource provisioning and the various existing approaches and algorithms used for resource provisioning.

Chapter 3

Resource Provisioning Approaches And Algorithms

The Grid infrastructure is organized as a collection of sites, where each site has a collection of resources and is administered independently of the other sites. To manage these resources a Resource Management System is required to search for required resources and reserve them for later use. This chapter describes resource provisioning and its different types along with provisioning process and steps followed in this process. Finally we give a description of various resource provisioning approaches and algorithms used for resource provisioning has been given.

3.1 Resource Provisioning

Provisioning means many different things to many different people. Consider the following definitions [14]:

Definitions:

- the act or process of providing
- the fact or state of being prepared beforehand
- a measure taken beforehand to deal with a need or contingency
- a stock of needed materials or supplies

In this context, provisioning refers to the “preparation beforehand” of IT systems “materials or supplies” required to carry out some defined activity. In general, it goes further than the initial “contingency” to the onward management lifecycle of the managed items. This could include the provisioning of purely digital services like user accounts and access privileges on systems, networks and applications. It could also include the provisioning of non-digital or “physical” resources like the requesting of CPU,Disk Space and network bandwidth etc.

In short Resource Provisioning can be defined:

"Provisioning is the automation of all the steps required to manage (setup, amend & revoke) user or system access and entitlement rights to electronic services".

3.1.1 What is a Provisioning System?

A facility or capability exists within a network to carry out provisioning actions on a given set of resources. In many (but not mandatory to all) provisioning systems, the "system" denotes the capacity to take provisioning requests and to execute them in the context of some pre-defined flow of execution. In many cases (but not mandatory to all) the provisioning system encompasses a formal work- flow to implement these pre-defined business process.

Figure 3.1 shows a high- level schematic of two provisioning systems, each responsible for managing provisioning actions for a defined set of resources.

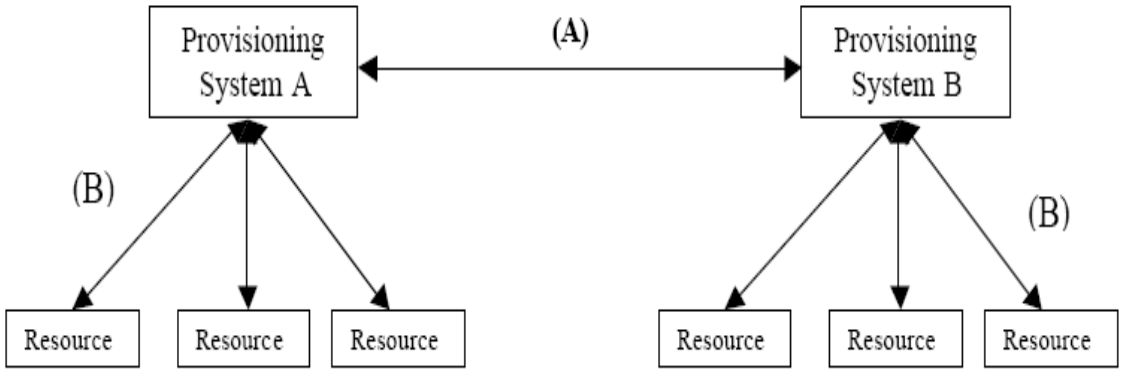

Figure 3.1 Schematic Diagram of Provisioning system [14]

3.2 Types of provisioning

Depending on infrastructure capability, types of services and business models (e.g., assumption of risk, optimization objectives) one or more of the provisioning scenarios described next may be appropriate for a specific service provisioning. The objective of provisioning is to allocate sufficient resources in order to avoid violations of service level guarantees.

Provisioning scenarios can be classified into three categories [15].

- Dedicated resource provisioning: The simplest case of provisioning is that of dedicated resources allocated to a single consumer. When a subscription request is received, resources are provisioned after considering the service specifications versus the anticipated load. In this case, the SLA system monitors for service level compliance, reporting violations when they occur. In the absence of the capability to add resources dynamically, the dedicated environments are typically over-provisioned anticipating the worst-case workload scenario.

- Per-SLA virtualized resource provisioning: In an environment in which consumed resources are virtualized (and underlying physical resources are shared in support of multiple SLAs), the provisioning system evaluates whether an additional SLA can be supported with the resources available to this virtualized system. The system may also add new resources to this shared pool based on the aggregate anticipated load of all supported SLAs. As before, the SLA system monitors for compliance and reports violations.

- Dynamic resource provisioning: In this, the most sophisticated on demand environment, resources are allocated to services, as needed. An initial set of resources is provisioned for a service, and the SLA system monitors the performance of that service. When the load changes, new resources are allocated or deallocated dynamically, in order to minimize service delivery costs while meeting SLA objectives. Dynamic provisioning can be used in conjunction with the previous two scenarios for initial resource allocation.

Another Provisioning category can be on the basis of static or dynamic i.e. Resource Provisioning can be done statically using advance reservations or dynamically using mechanisms such as Condor Glidein [16].

Static Provisioning

Using advance reservations a scheduler agrees to make certain resources available for certain timeframe for a specified user or set of users. All the jobs submitted by the user during this timeframe are scheduled on the reserved resources. There is generally

no incentive for making reservations for a single job since policy decisions will likely be made so that there is no start time advantage to making a reservation over submitting the job to a queue [16]. However for a workflow, all the tasks in the workflow can be executed using a single resource reservation, eliminating the queue wait time and so the completion time is expected to be reduced. A notice period has to be given before the reservation can become active. The scheduler uses this notice period to make sure that no currently queued jobs would get unnecessarily delayed because of this reservation. In other words, the scheduler tries to make sure that no start time advantage is obtained by making advance reservation instead of submitting the job to the queue.

Dynamic Provisioning

Resource provisioning can also be done dynamically using mechanisms that can provide temporary access to dedicated resources from a remote site without requiring extra coordination with the remote scheduler. An example of dynamic resource provisioning is Condor Glidein [16]. It submits a job to a remote site. When this jobs starts executing on the allocated resources, it starts up processes on them that allow job submission to these resources bypassing the remote scheduler. Thus it created a dedicated execution environment on the set of allocated resources controlled by a scheduler local to the user. The cost associated with dynamic provisioning for Condor Glidein is the wait time before this main job can start running.

Points to Ponder

- Resource provisioning involves large number of factors which should be kept in mind while doing Resource Provisioning:
- When using resource provisioning, an important decision to be made is how many resources should be provisioned and for how long.
- The delay incurred in obtaining the provisioning often depends on the amount of resources requested.
- The resources provisioned should be enough to execute the workflow to completion.

- A number of provisioning requests can be determined differing in the amount and duration of resources requested that are sufficient to execute a particular workflow.
- A particular request has to be selected among these that optimize the metric of interest.

3.3 Resource Provisioning in Grid Environment

Grid Computing encompasses various aspects of distributed computing such as seamless access to distributed data (Data Grids), and collaborative distributed environments (Service Grids). This pattern focuses on Grids for running computationally intensive applications (Computational Grids), focusing on Grids that leverage the idle capacity of commodity workstations. In order to encompass a large amount of resource providing machines, the middleware must be easy to deploy, without requiring extensive reconfiguration on existing systems. Security must be provided for both resource owners and users submitting applications. Machines sharing their resources need protection against malicious code. Grid users should have some guarantees regarding data confidentiality and integrity. A large number of existing parallel applications exist that could benefit from the Grid. The middleware must provide grid-enabled libraries supporting standard parallel programming models, allowing for easy application migration to the Grid environment. A user interested in executing applications submits a request via an access agent. The applications can be either regular applications consisting of a single process, or parallel applications. The scheduling service receives the execution request, checks the identity of the user who submitted the application, and uses a resource management service to discover which machines (nodes) have available resources to execute the application. These nodes, which must run the resource provision service, are called resource providers. If there are nodes with free resources, the scheduler sends the jobs to the selected nodes. Otherwise, the request waits in an execution queue. When the execution is finished, the results are returned to the user.

3.3.1 Grid Structure

The structure of the Grid can be composed of five main components. Figure 3.3 shows the interactions between these components [17].

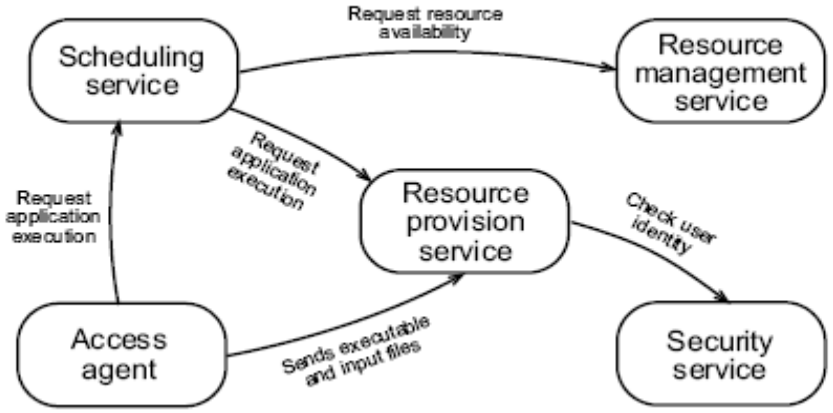

Figure 3.2 Components of Grid [17]

The access agent is the primary access point for users that interact with the Grid. It runs on each node from which application execution requests will be submitted. Besides the submission of execution requests, it allows the user to specify application requirements, monitor executions, and collect execution results.

The resource provision service runs on each machine that exports its resources to the Grid. It is responsible for servicing execution requests by retrieving application code, starting application execution, reporting errors on application execution and returning application results. This service is also responsible for managing local resource usage and providing information about resource availability.

The resource management service is responsible for monitoring the state of shared resources and responding to resource requesting, by matching the requests with resource offerings. It maintains a list of which resource providers are currently available and the state of their resources, such as processor usage and free memory. This service can also detect resource provider failures, and notifies the access agent when an execution fails due to a resource failure.

The scheduling service schedules execution of applications on available shared resources. It receives application execution requests, obtains available resources with the

resource management service, and then determines which application will execute on each resource. The security service is responsible for three major tasks: (1) protecting shared resources, so that a node sharing its resources with the Grid does not suffer the effects of a rogue application,(2) user authentication, so that application ownership can be established, enabling relations of trust and accountability, and (3) securing Grid communications, providing information confidentiality and integrity.

3.3.2 Dynamics

The behavior of a possible implementation of the Grid management when an access agent requests an application execution can be as follows.

- The access agent sends an execution request to the scheduling service, possibly with some information about the required resources.
- The scheduling service queries the resource monitoring service for nodes meeting the request requirements. A list containing the location of nodes satisfying the query is returned.
- The scheduling service determines if it is possible to execute the application in the available nodes. If the execution is possible, the request is sent to the selected resource providers. Otherwise, it queues the request for later execution.
- The resource providers download the application code from the access agent that submitted the execution request.
- The resource providers use the security service to verify the identity and permissions of the code owner.
- The resource providers execute the application. When the execution is finished, the results are sent back to the access agent.

3.3.3 Advantages of Resource provisioning

Resource provisioning implies creating a contract between the user and the resource owner specifying that a certain resource would be made available to the user for a certain timeframe. Resource provisioning for applications has several advantages over best effort execution[18].

- First and most importantly, it allows the user or workflow manager to control the scheduling and execution of the application on the provisioned resources. This in turn enables the estimation of the application performance prior to its execution.
- Second, it allows the user or workflow manager to explore the space of resources to be provisioned and optimize for performance without worrying about external factors such as the workload of the resource or the policies of the resource provider.
- Third, it enables the adaptation of the application based on the provisioned resources in order to achieve a certain level of performance.
- Fourth, it allows the execution of applications that require co-allocation of resources or have other constraints on the resource requirements.
- Last, due to the deterministic nature of the resource availability resulting from the contractual provisioning process, mechanisms for adapting to the otherwise dynamic nature of the Grid are not essential.

3.3.4 Resource Provisioning Overheads

Resource Provisioning offers many advantages but there are likely to be costs associated with resource provisioning that justify provisioning from resource owner's perspective [19]. These costs might include an element that represents the charge for the normal usage of the resource (e.g. number of service units per processor second), but it might also include an additional element that would account for factors such as the loss of utilization for the resource provider or the increased queue wait time of best effort jobs due to the provisioned resources. The problem is to identify from the user's perspective a set of resources to be provisioned in order to optimize a parameterized cost metric that includes the cost of provisioned resources and the application runtime.

The Grid is comprised of R autonomous compute resources or sites. Each site r is a cluster of Nr homogeneous processors. Each site r can be queried for the possible start times and the associated cost for provisioning n processors for d duration. The query and the response are represented as:

$$Er(n,d) = \{\langle s1,n,d,c1,f1,r\rangle, \ldots, \langle si,n,d,ci,fi,r\rangle \ldots\} \quad \ldots(1)$$

Note that this is no different than assuming the ability to query the possible start times of a task or a resource reservation requiring n processors for d duration from site r. Each tuple <si, n, d, ci, fi, r> in the response set represents the availability of n processors for d duration starting at time si with a multiplicative cost of ci and a fixed cost of fi from the site r. Each tuple is called a slot. The total cost of using the slot is defined as (n*d*ci + fi). The multiplicative cost is used to specify the usage charge that is related to the amount of resource provisioned. For example, it might represent service units per processor second or dollars per processor second. This multiplicative cost might also depend on the start time similar to the existence of express queues on some Grid sites that have a faster response time but charge a higher number of service units per processor second. The fixed cost might be used to represent the lost resource utilization or the increased response time due to the fragmentation of resources resulting from allowing fixed resource reservations. It might also be used to discourage users from provisioning too many slots and instead provision resources in bigger chunks resulting in less bookkeeping for the resource provider. While a single cost factor per slot would have also served the purpose, having two separate cost factors provides better semantic description of the model.

The set of all the slots available from the site r,denoted by Er(.), can be constructed as

$$E_r(.) = \bigcup_{1 \le n \le N_r} \bigcup_{1 \le d \le D_r} E_r(n,d)$$

However, it would overload the Grid site if every user queried for every possible value of n and d. It would be much more efficient if a single query can return the set of available slots on the site. Thus, we assume that each site r can be queried for the set of slots available on the site.Er(.) = {<s1,n1, d1, c1,f1,r>,……,

<si,ni,di,ci,fi,r>….}

We also assume that the sites only return the set of non-overlapping slots i.e. no two slots represent the same underlying resource and each slot can be provisioned independent of any other slot.

A global set of resource slots: Â is the set of all the resource slots available in the Grid. It can be constructed by querying each site as shown below:

Â=Sum of Available Resources

Resource Provisioning problem can be defined as the problem of finding a subset â of Â (or in other words, an element of the power set of Â, P (Â)) and a schedule ŝ of the application over â that such cost of provisioning is minimized and its performance is optimized.

3.4 Resource Provisioning Algorithm

In order to find the schedule ŝ, that minimizes the make span of the application over the resource plan â, finding such an optimal schedule is a NP-Hard problem [20].In the case of the GA heuristic, the GA is used to create the resource plan â, while a list scheduling algorithm is used to create the schedule ŝ.

Genetic Algorithm (GA)

In the genetic algorithm (called GA), an initial population of certain number of randomly generated unique subsets of Â has been discussed[20].For each individual in the population, a schedule of the application is created on the slots in the individual using the list scheduling algorithm in Figure .

1. Create a topologically ordered list of tasks in the application where parent tasks precede the child tasks.
2. While list not empty
3. Remove the first task from the list and schedule it on a slot that leads to the minimum finish time of the task.
4. End-While

The time complexity of the scheduling algorithm is O(|V|.|Â|.nA) where |V| is the number of tasks in the application, |Â| is the cardinality of Â and nA is the maximum number of processors in any slot in Â. The total cost of the individual is computed only based on the slots that have at least one task scheduled on them. Such a slot is called a scheduled slot. The total cost of the individual is computing using equation with the allocation cost determined by the scheduled slots and the scheduling cost by the make span of the schedule. The GA goes through a number of iterations. During each iteration, each individual is mated with another individual and creates two off springs. This is called a crossover. The first offspring inherits slots that exists only either in the first parent or the

second (similar to the XOR binary operation). The second offspring inherits slots that are either present in both the parents or in neither of them (similar to the XNOR operation). This doubles the population size. The selection of the crossover operators was motivated by the observation that they lead to a greater number of unique individuals in the population. After removing the duplicates, the total cost of each new member of the population is computed and retaining the least total cost individuals in the population reduces population to the previous size. The algorithm stops after certain number of iterations and the solution selected is the individual with the least total cost in the current population.

3.5 Related Work

A number of efforts have been made to develop automatic Grid resource management and brokerage solutions but very few of them are addressing the issue of resource management covering software components (activities) and their automatic deployment. A separation between meaning, behavior, and implementation of the Grid application components is described in [21]. The work in [22] matches a high-level application specification to an optimal combination of available components. In contrast, GLARE provides a high-level application specification in a hierarchy of activity types and provides dynamic registration and automatic deployment of software components. GLARE [23] a Grid-level activity registration, deployment and provisioning technique that provides dynamic registration, automatic deployment and on-demand provision of software components. GLARE is designed and implemented as a distributed technique that stores information about application (software) components, called activities. Pegasus [24] uses Chimera [25] and Transformation Catalog [26] for transforming an abstract workflow into concrete workflow. The transformation Catalog is used to map a logical representation of an executable (transformation) to a physical representation, which describes its functionality and accessibility. The catalog uses MySQL as a centralized backend database. Chimera Virtual Data System [27] describes and stores data derivation procedures and derived data in a central database. It provides a special language interpreter that translates user requests. This system is useful for datagrid applications, but works with a dedicated querying mechanism. Pegasus uses Globus

middleware services and automates replica selection. It does not provide automatic/on-demand deployment of software components.

GrADS [28] resource selection technique addresses the discovery and configuration of physical resources that match application requirements. It provides a declarative language using set matching techniques, which extend Condor matchmaking [29] and support both single and multiple resource matching. This system does not cover Grid application components. S. Decker et al describe in [30] Grid resource matching using semantic web technologies. This work proposes physical resource matching by using ontologies, background knowledge and rules. It highlights the need of semantic description of Grid resources and resource matching but does not address issues of performance and efficiency. Both systems do not cover software resources like Grid computational activities.

CrossGrid [31] provides a distributed component registry with peer-to-peer technology. It supports inter-registry communication for maintaining table coherency. Grimoire [32] extents UDDI [33] to provide invocable activities such as workflows or legacy programs. GridLab capability registry [34], CrossGrid component registry and MyGrid Grimoire [35] provide registries for static information of the Grid applications. UDDI [33] and Handle System [36] can be used to augment our system but they have their own limitations.UDDI is a specification for distributed web-based information registries for web services but unsuitable for legacy scientific applications. Also it does not support dynamic updates. Handle System supports a very basic querying mechanism. Furthermore, it requires domain specific naming authorities to be registered in a root naming authority which is not managed efficiently. Globus MDS provides a hierarchical aggregation technique for distributed Grid resources.

This chapter described static and dynamic resource provisioning along with its advantages and overhead involved .A Grid structure was also described for resource management and finally related work in different field has been described.

In the next chapter we will focus on the problems encountered in the existing approaches and possible solutions to these problems.

Chapter 4
Problem Statement

This chapter gives a detailed description of the problems in various Grid middleware with respect to Resource Provisioning i.e. what are the various challenges in present scenario and possible solution to these problems. Finally, Alchemi is discussed as Resource Provisioning technique.

4.1 Challenges with Existing Approaches

Grid computing [1] is an emerging paradigm that seeks to harness the power of the Internet and the sophisticated resources spread across it. It is also expected to support high-fidelity, real-time collaboration between geographically distributed Virtual Organizations (VOs), that comprise researchers, scientists, other users, and organizations.

Although existing grid infrastructure middleware seems suited to supporting next-generation grid applications, however, developing distributed grid applications using these is fraught with the following challenges [37].

Challenge 1: Tight coupling with Grid infrastructure Middleware

Grid applications are developed using one of existing grid infrastructure middleware technologies, such as Globus, Legion, ICENI, and others making them tightly coupled to the underlying middleware. But with advances and sophistication in grid middleware technologies, it is imperative for grid applications to avail of these advances and applications are seamlessly portable across different middleware without significantly affecting existing applications thereby preserving investments. This proliferation of grid middleware choices has raised the level of accidental complexity by increasing the amount of effort required to interoperate and port applications between grid middleware technologies.

Solution

An effective approach to decouple grid applications from the underlying grid middleware is to implement the grid middleware as a Web service [38]. Grid applications can then use standards-based ubiquitous web protocols such as http to access the underlying grid middleware.

Challenge 2. Accidental complexities in integrating software systems

To reduce lifecycle costs and time-to-market, application developers are attempting to assemble and deploy distributed grid applications by selecting the right set of compatible grid middleware components, which is a daunting task. The problem is further exacerbated by the existence of myriad strategies for configuring and deploying the underlying component middleware to leverage the environment advantages. Also, integrating applications using multiple middleware technologies demands multiple skill sets, which makes the task even more complicated. Application developers therefore spend non-trivial amounts of time debugging problems associated with the selection of incompatible strategies and components.

Solution:

To overcome this problem an integrated set of processes and tools are needed that can

- select and validate a suitable configuration of middleware components and
- generate optimized Web service configurations automatically.

Challenge 3: Satisfying multiple Quality of Service requirements simultaneously

Grid applications demand varying degrees and forms of QoS support from their grid middleware. For example, collaborative scientific applications involving geographically dispersed scientists, engineers, and physicists working on real-time experiments and data require the infrastructure to be efficient, predictable, scalable, secure, and fault tolerant. Owing to the complex nature of these QoS requirements, it is not feasible for a single grid infrastructure middleware to provide an end-to-end solution that addresses all these challenges.

Solution:

For this, a highly configurable, flexible, and optimized higher-level, web-application based on standards, are required to assemble and deploy on middleware tailored to the needs of the Grid application.

Challenge 4: Lack of well-defined patterns for resource reservation

Grid applications require simultaneous access to several different types of resources available from multiple resource and service providers that own them. These service providers include Internet Service Providers (ISPs), Storage Service Providers

(SSPs), Content Service Providers (CSPs), and others. For example, a distributed virtual surgery application involving geographically dispersed doctors, radiologists, medical professionals, and medical students will require high bandwidth for collaboration, large storage databases to hold patient records and radiology images, expensive display devices for precise 3-D modeling and rendering of images, virtual reality equipment for simulating surgeries, and telephony equipment to maintain multiple call sessions.

Applications that require these resources must maintain Service Level Agreements (SLAs) with each individual service provider that provide the resources and services. Moreover, today's grid applications must authenticate themselves with each service provider every time they access resources owned by the provider. Conventional grid infrastructure middleware provide only the means to securely access the resources from different service provider. However, the responsibility of reserving and accessing the resources is still the responsibility of the grid applications.

Solution:

A possible solution to address this problem is for the Grid to provide a set of generic resource reservation strategies that Grid applications can use. However, such a solution fails to serve the needs of all grid applications, each of whom might have differing end-to-end resource and QoS needs. Therefore ability is required to compose patterns-based strategies for multiple resource reservations while assuring the end-to-end QoS requirements of the Grid applications. These strategies should be deployed within the Grid and made available to the grid applications as a Web service.

Challenge 5: Provisioning and managing resources is hard

As mentioned in challenge 4, grid applications must make reservations for several different resources while ensuring that the end-to-end QoS requirements are met. Even if this problem is resolved by deploying custom resource reservation strategies within the grid middleware as outlined above, provisioning and managing multiple resources from multiple providers is a daunting task that existing grid infrastructure

middleware currently do not handle and leave it to the grid application. If it is, then a separate set of tools can synthesize the appropriate resource provisioning and management strategies composed from a library of higher-level QoS-enabled grid middleware components.

Solution:

A promising way to address the challenges in developing grid applications is to resolve the configuration, management, and deployment challenges of deploying QoS-enabled grid as Web services.

4.2 Problem Definition

In this work, Alchemi .Net Grid middleware has been studied for existing features. It allows flexible application composition by supporting an object-oriented grid application-programming model in addition to a grid job model.

The key features supported by Alchemi are [39]:

- 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); and
- 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.

However, Alchemi does not support advance reservation, Resource Provisioning and resource allocation, QoS and Service-level agreements. The objective of this thesis is to implement Resource Provisioning in Alchemi .Net using Cross Platform Manager.

Methodology:

For implementing Resource Provisioning, technique is proposed that uses web-service support provided through Cross Platform Manager. Cross Platform Manager mediates request for Resource Provisioning on behalf of Alchemi and forward it to web application for reserving resources

- The user submit request for resource reservation to Alchemi manager, which submits request to Cross-platform manger.
- Cross platform manager in turn calls web service for Resource Provisioning and search for available resources.
- Requested resources are selected and reserved for use by user for requested time.
- Application is mapped on to the resources for execution.
- After completion of execution resources are freed for use by other application.

The proposed architecture for resource provisioning and detailed functioning of implemented web service will be described in the next chapter.

This chapter describes the challenges in field of Resource Provisioning in existing middlewares and the possible solutions to these challenges. In the next chapter design of proposed technique would be discussed.

Chapter 5

Proposed solution

This chapter describes the proposed architecture for resource provisioning on Alchemi and various component along with their functions and experimental setup used to implement the solution for Resource Provisioning. Then, in next chapter, the application for Resource Provisioning developed in C# is explained and its implementation results using Web services are shown.

5.1 Resource Provisioning Architecture

The Resource Provisioning architecture proposed in this thesis provides a technique for necessary components and services. The architecture proposed is based on .Net architecture proposed in [40]. The overall architecture is shown in Figure 5.1. The architecture is based on the volunteer model and supports the Service Oriented Architecture (SOA) paradigm. The main components of resource provisioning system are Core services. Systems are implemented using .NET technology with C# language [40,41]. A resource may have complex internal policy for deciding permissions, relative priorities, and other scheduling and management procedures and local policy, non-Grid usage. Architecture provides a powerful mechanism for vitalising or abstracting a resource.

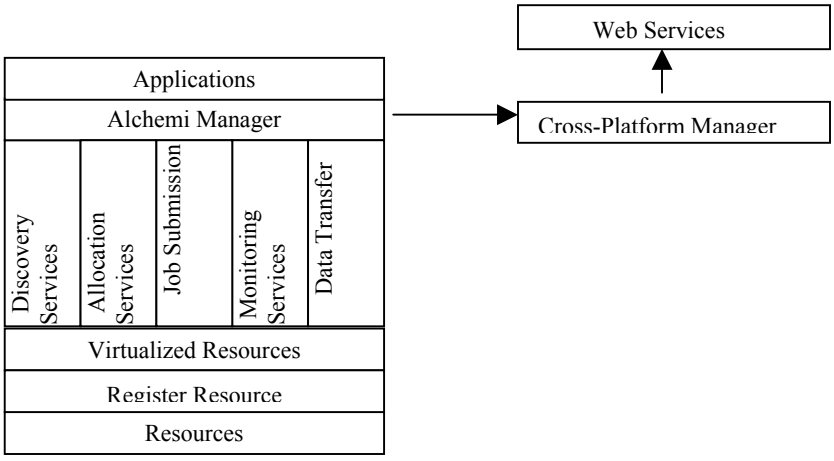

Figure 5.1 Architecture for Provisioning

5.1.1 Core services

The core services consist of the following services.

- **Resource registration service**

  The architecture uses resource registration service as resources virtualization technique. The resource registration service collects service provider's registration message and its status and information. While each resource or service may have proprietary and internal formats for expressing policy and terms of service, the use of a standardized notation renders the resource onto the Grid, making it usable by clients who are not familiar with the inner working of the service.

  This service is also designed to support the hierarchal registration mode. In this mode, a broker aggregates all information, for both itself and its workers, and forwards it to other brokers. By doing this, it helps the system in being more robustness and scalable. The operation of this service is as summarized in Figure5.2.

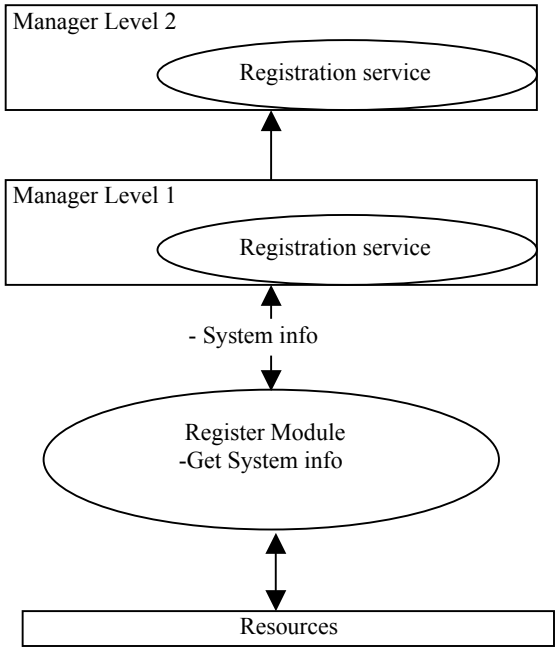

Figure 5.2 Resource registration service

- **Resource discovery and allocation service**

Resource discovery is the process of querying the distributed state of the Grid to identify those resources whose characteristics and state match to those desired by the resource consumer. For managers, resource selection is the process of choosing from set of candidate resource managers. For consumers, resource selection is the process of choosing from a set of candidate resource providers. The selection is typically driven by high-level application criteria, such as time to completion, reliability, or cost. The selection and negotiation by managers results in a modified resource environment for consumers. The resource allocation service or manager deals with assigning the jobs to resources. Allocation service will allocate the best resources shared service URL to client, and let client invoke service directly.

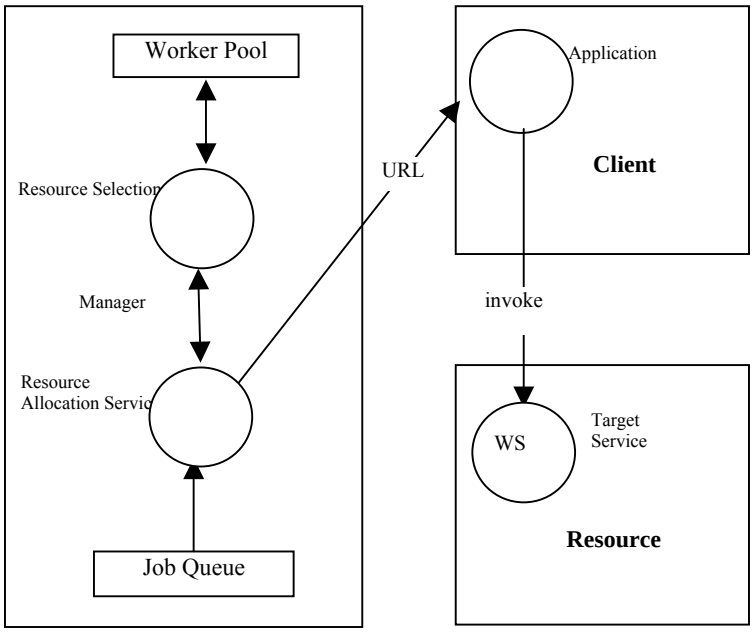

Figure 5.3 The service invocation mode

- **Job submission service:**

The user uses this service to submit jobs to broker. First the users create their application, to submit jobs. Second a web interface is used to wrap the services needed by users. Therefore, users can use this web interface to submit their jobs.

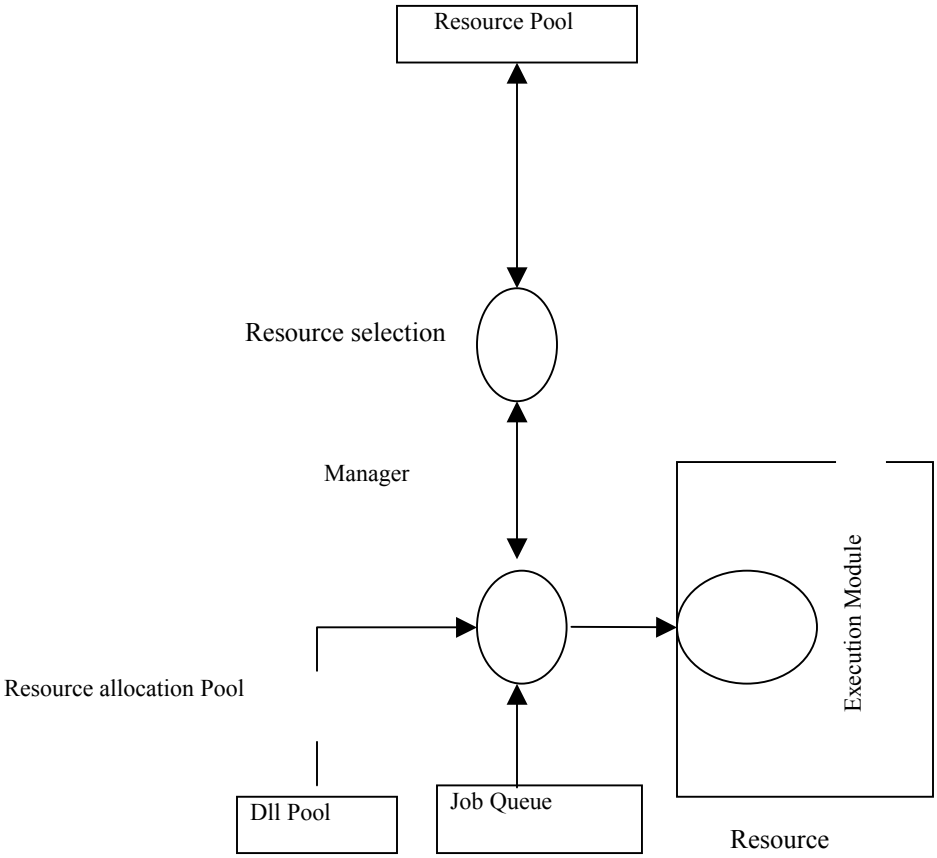

Figure 5.4 Job Submission

- **Monitoring service:**

  This service is used to monitor all status of service such as the number of resources in system, the number of jobs submitted from users, the number of jobs assigned to resources, the number of finished jobs, etc. The monitoring service can be displayed via Web interface. Monitoring can be a complicated activity, both because the environment is distributed and because resources may have been provided via intermediaries such as resource brokers. The view of the resource may have been abstracted, making it difficult for the consumer to know enough about the underlying resource set to directly monitor the environment.

- **Data transfer service:**

  This service handles data, input file, and output file transfer when user application is running. In addition, the data transfer service can be used for dynamically allocating and adjusting disk storage on the demand of users.

5.1.2. Implementation

The architecture discussed above is implemented using Alchemi middleware which uses a .Net technique. Manager in Alchemi acts as a broker between a user and a set of associated resources. It provides a single point of submission for tasks, using standard Grid resource management protocols to forward the task to one or more of the underlying resources for execution. Different components and interaction between them is shown in Figure 6.

Manager: The manager provides basic services for both clients and resource providers such as resource registration service for resource provider, resource discovery service for clients, job submission service for receiving jobs submitted by clients, and resource allocation service for allocating jobs to workers. The executor which runs on resources provides information to manager about the availability of resources and registers resource with Alchemi Manager. When request for a particular resource is received Manager checks available resources and takes appropriate actions

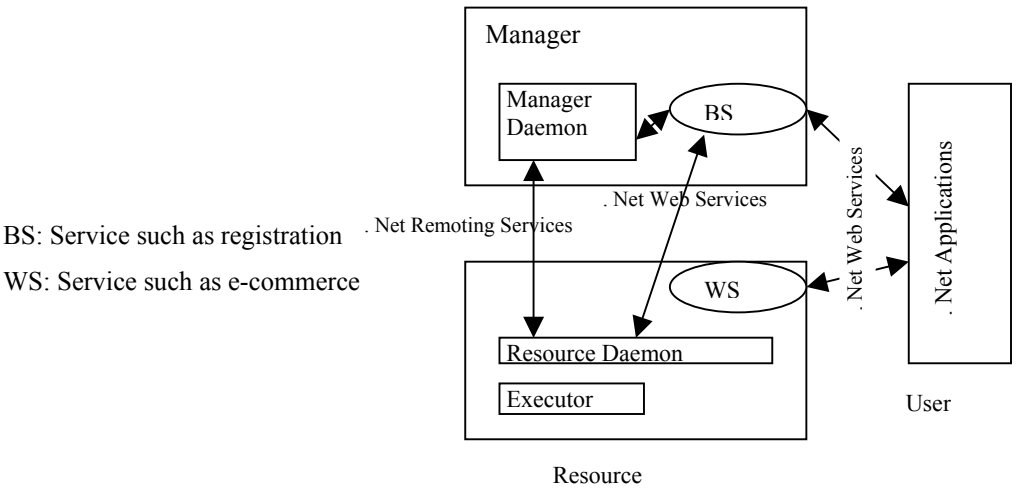

Figure 5.5 Interactions between Modules

Executor: There are two types of resources, volunteered and non-volunteered resources. In case of volunteer resource provider such as PCs, the resource executes jobs only when it is idle. For non-volunteer, such as Cluster, the manager can assign jobs to resource provider any time. Executor runs on each resource and register with Manager when resource wants to become member of Grid architecture.

User: The most important objective of users is to submit their jobs to manager. The manager then has to support this objective by providing a simple API or web interfaces to access the services.

5.2 Experimental Setup

The steps for installing various Alchemi Components are given below. It includes the installation, configuration and operations for each of the components.

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 use 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.

This chapter described the proposed architecture for Resource Provisioning and along with different scenario of resource selection and provisioning and application execution. Finally, the installation of Alchemi .Net is described which is used to run the proposed solution.

Chapter 6

Experimental Results

This chapter describes the implementation of Resource Provisioning architecture proposed in previous chapter. An application developed in C# takes as input the requires set of resource for executing an application. These resources are searched in the available set of resources on Thapar Grid and are reserved for executing an application. This application is further modified and converted into web-service which is invoked by Alchemi through Cross platform manager .The results for both are shown in the form of screen shots.

6.1 Resource Provisioning Application

As discussed in technique, for provisioning the user need to provide its resource requirements based on which availability of resource is checked .The resources are than reserved and its control is transferred to user for mapping application on the resources. In order to allocate resource on the basis of requirement of user and to find out execution-time in order to predict availability of resource and hence develop a schedule, an application is developed in C# which is further modified into web service to run on Alchemi via Cross-platform manager. The flow graph showing the execution of application is shown followed by application screenshot. The steps followed are:

- user enters the list of resources to be searched,
- entered resources are searched in the available database,
- user is asked to enter the time for which resources are required,
- resources are provisioned for the requested period and entry is made in the temporary database,
- application is executed on the resources,
- after executing ,the resources are freed,
- if request for additional resources is received than allocated resources are checked and compared with original database'
- If resources are available they are allocated else request is rejected.

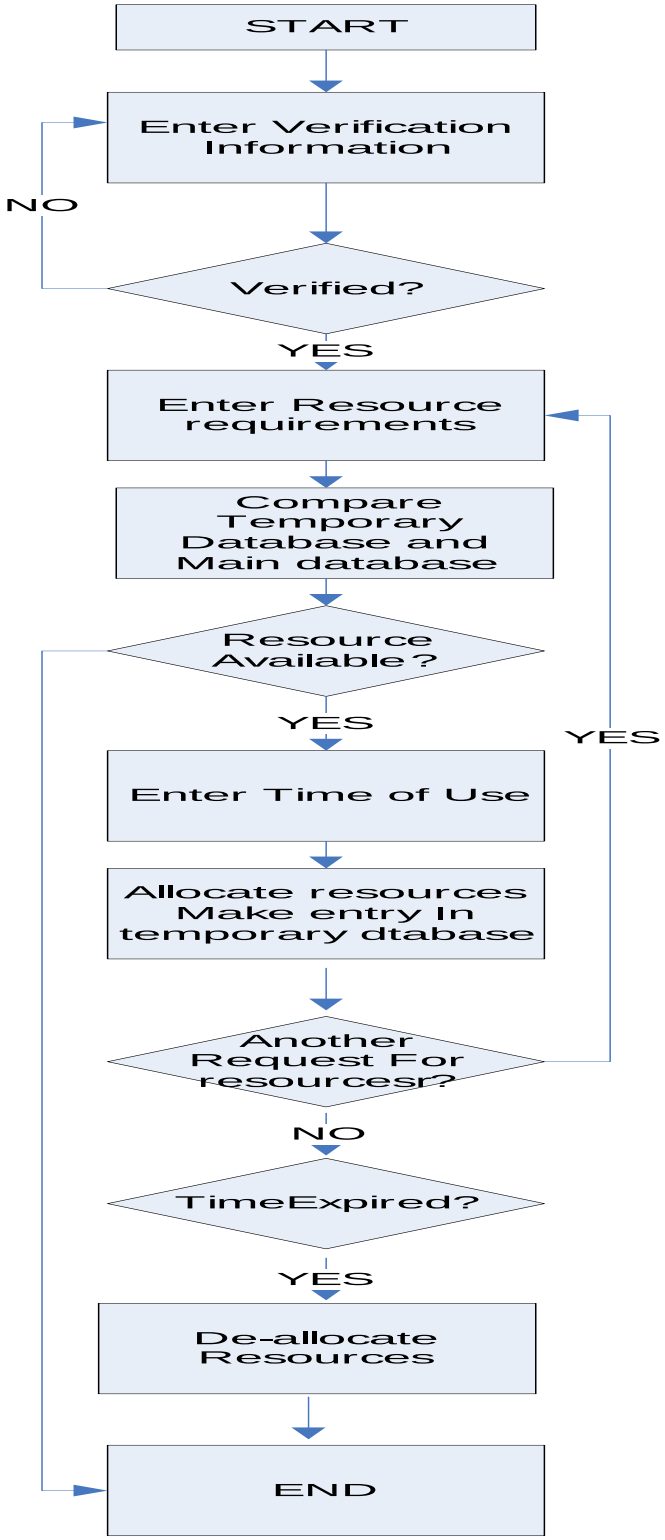

Figure 6.1 Flow Graph showing Execution of Provisioning Application

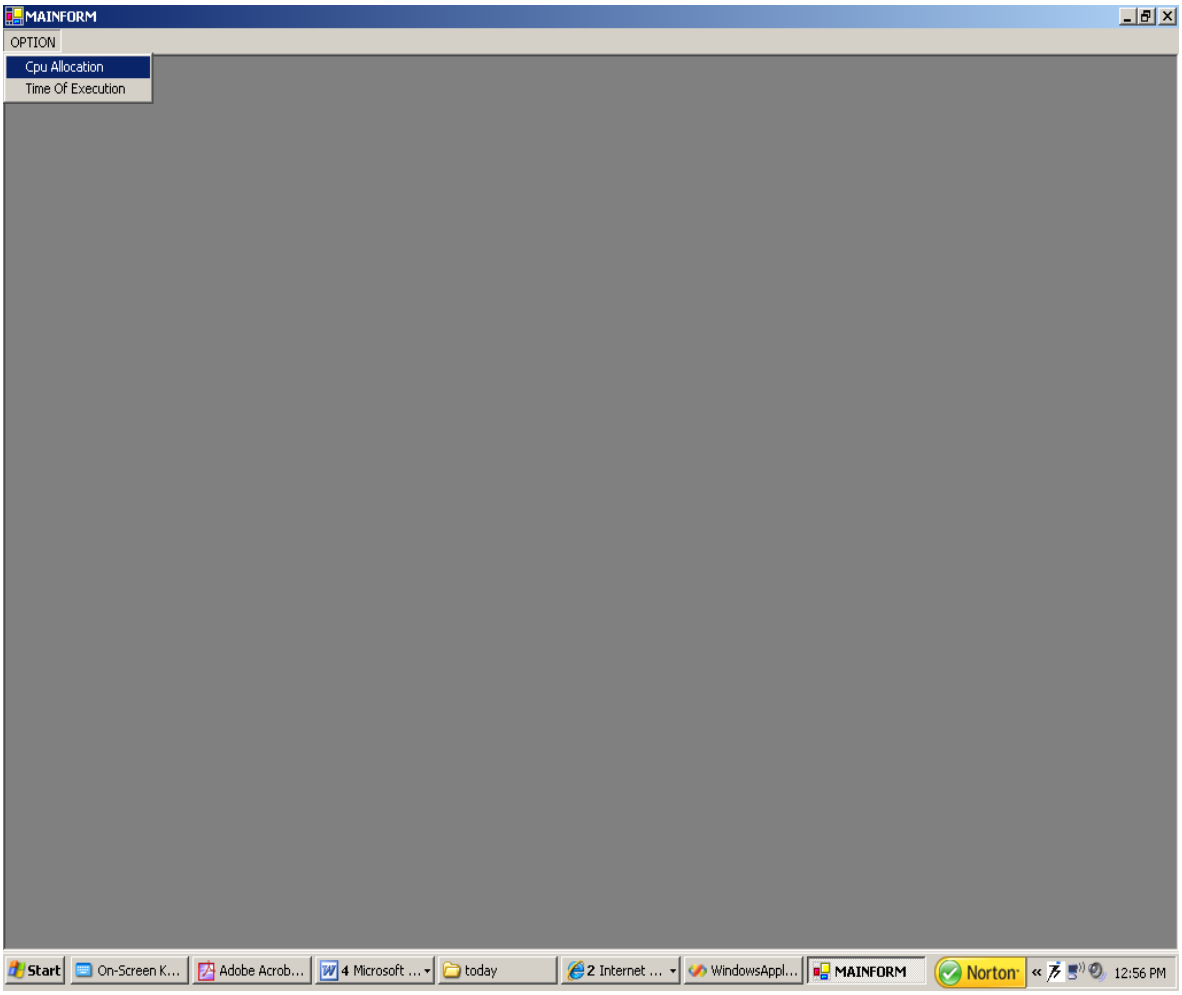

Figure 6.2 Option Window

Option Window: It provides option of either selecting provisioning application or execution time prediction application. By selecting one of the applications the user can perform either Resource Provisioning or can calculate execution time of an application.

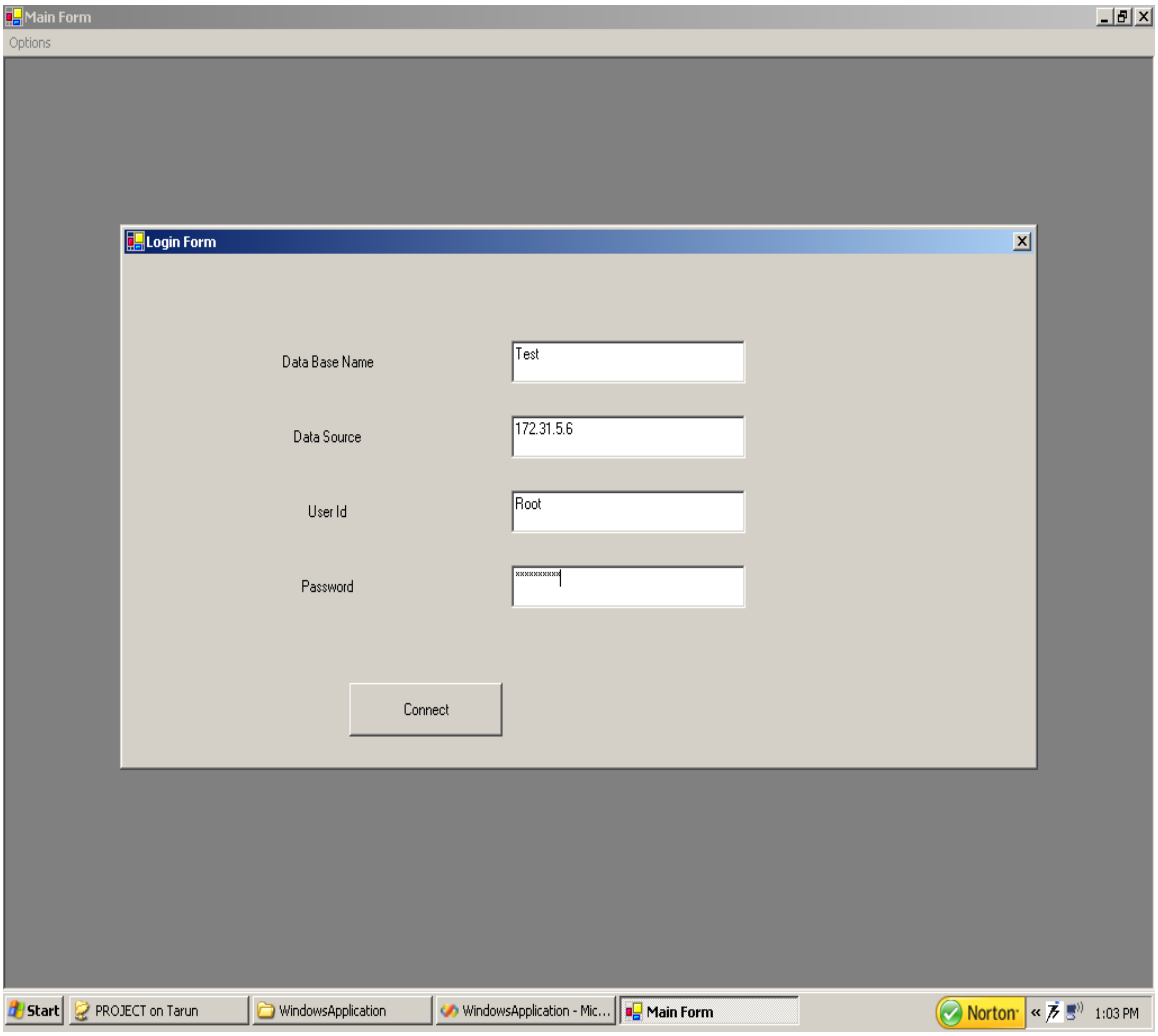

Figure 6.3. Login

The User Input: In order to use application user is required to provide following information:

- Name of the database
- IP Address of Source Computer
- User Id and Password

Based on the provided information a check is performed and user is logged in only if all information is correct.

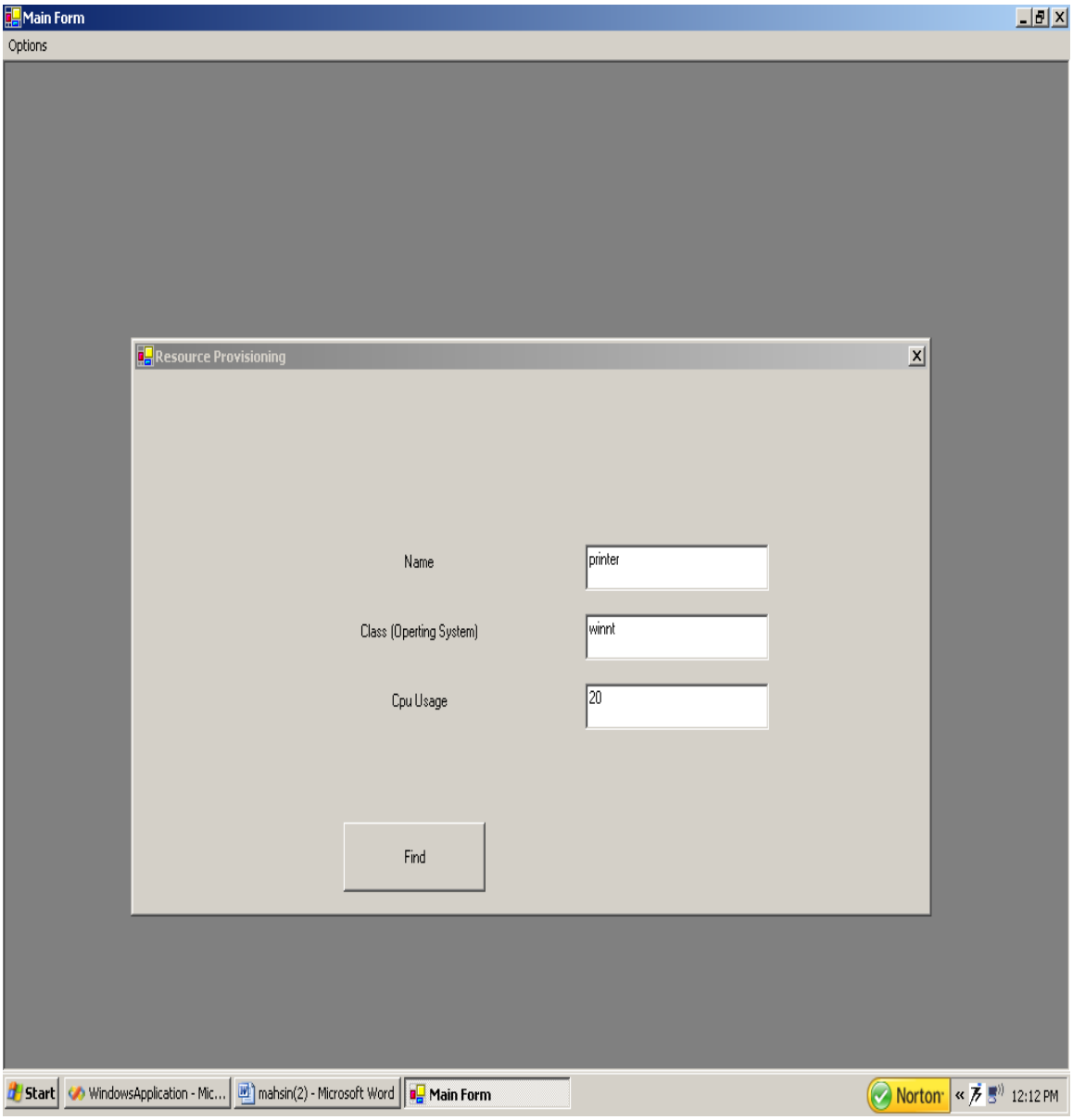

Figure 6.4 Search Resources

Search Resources: The resource user need resources to run its application The user needs to search resources from the available list of resources with resource provider. This feature asks user to enter the list of resources so that they can be matched against available resources list.

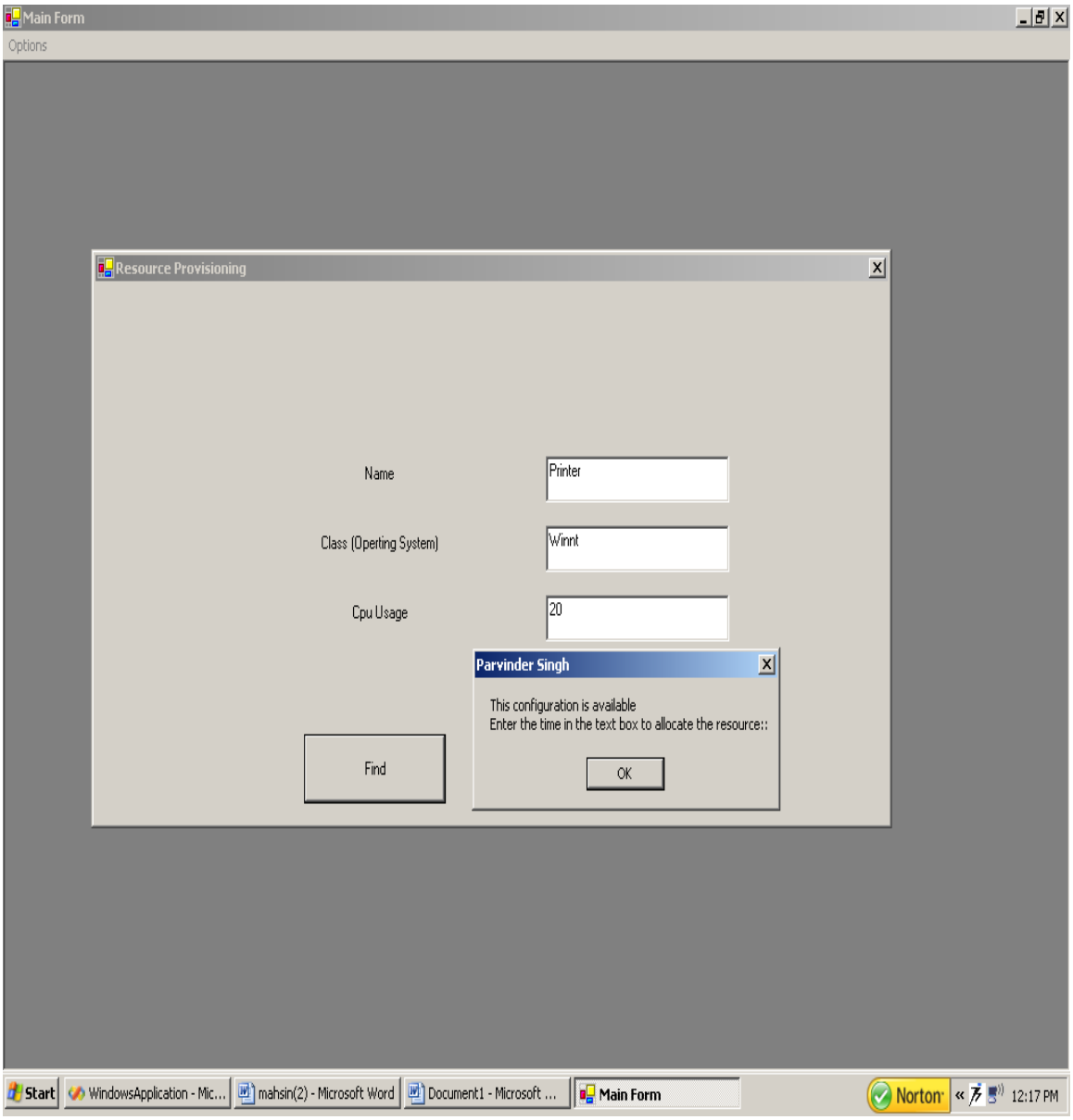

Figure 6.5 Resource Selection

Resource Selection: The entered list of resource is matched against available resources and if both data matched than user is asked to enter the time for which resources are required. The list of selected resources is made in the temporary database for further matching. Based on this information, the resources which satisfy the condition for executing actual service is provided.

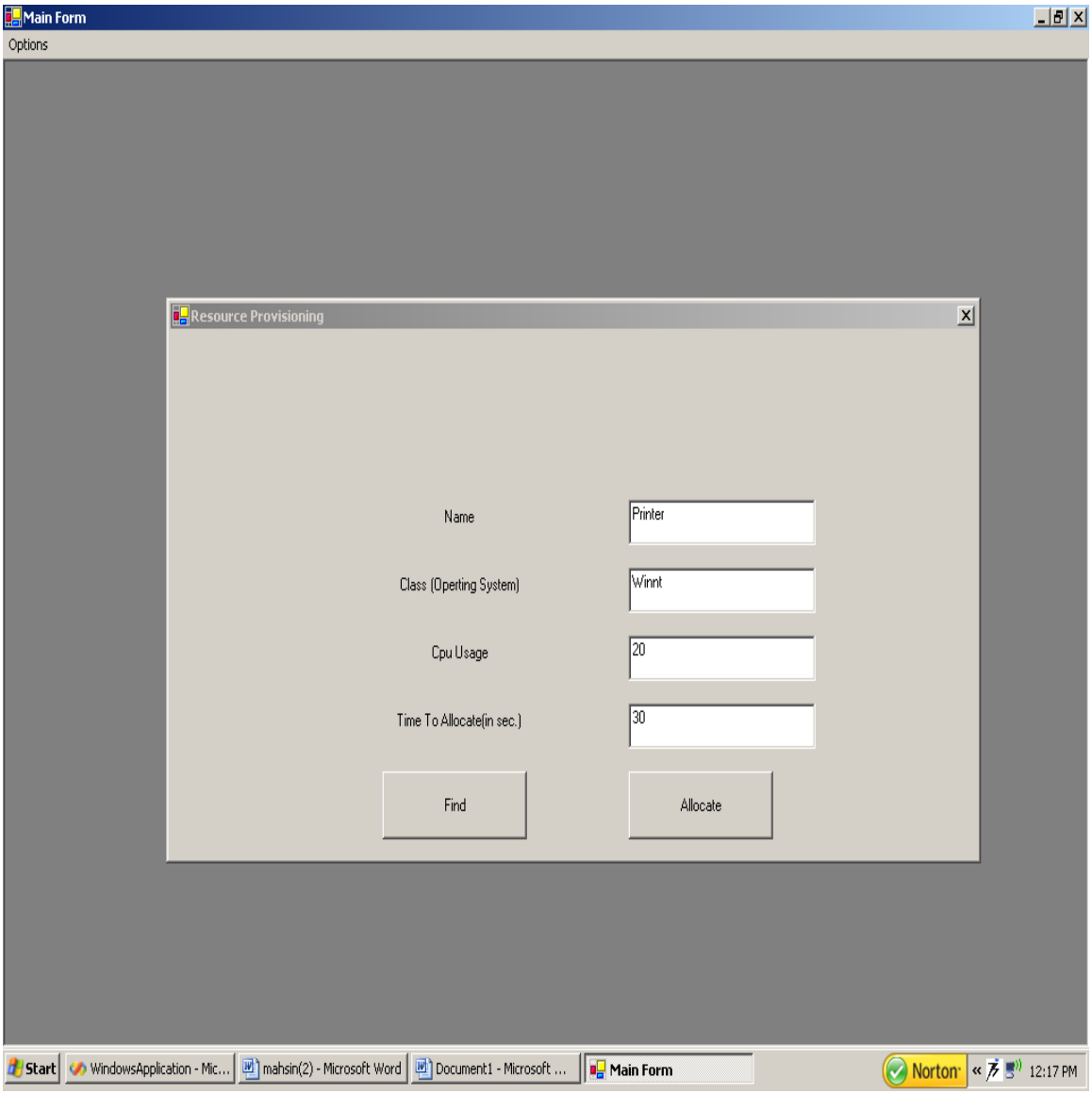

Figure 6.6 Enter Time

Time of Allocation: After search for the resources in the database is completed the user is asked to enter the time for which resources are required. The selected resources are than provisioned for the time entered by user.

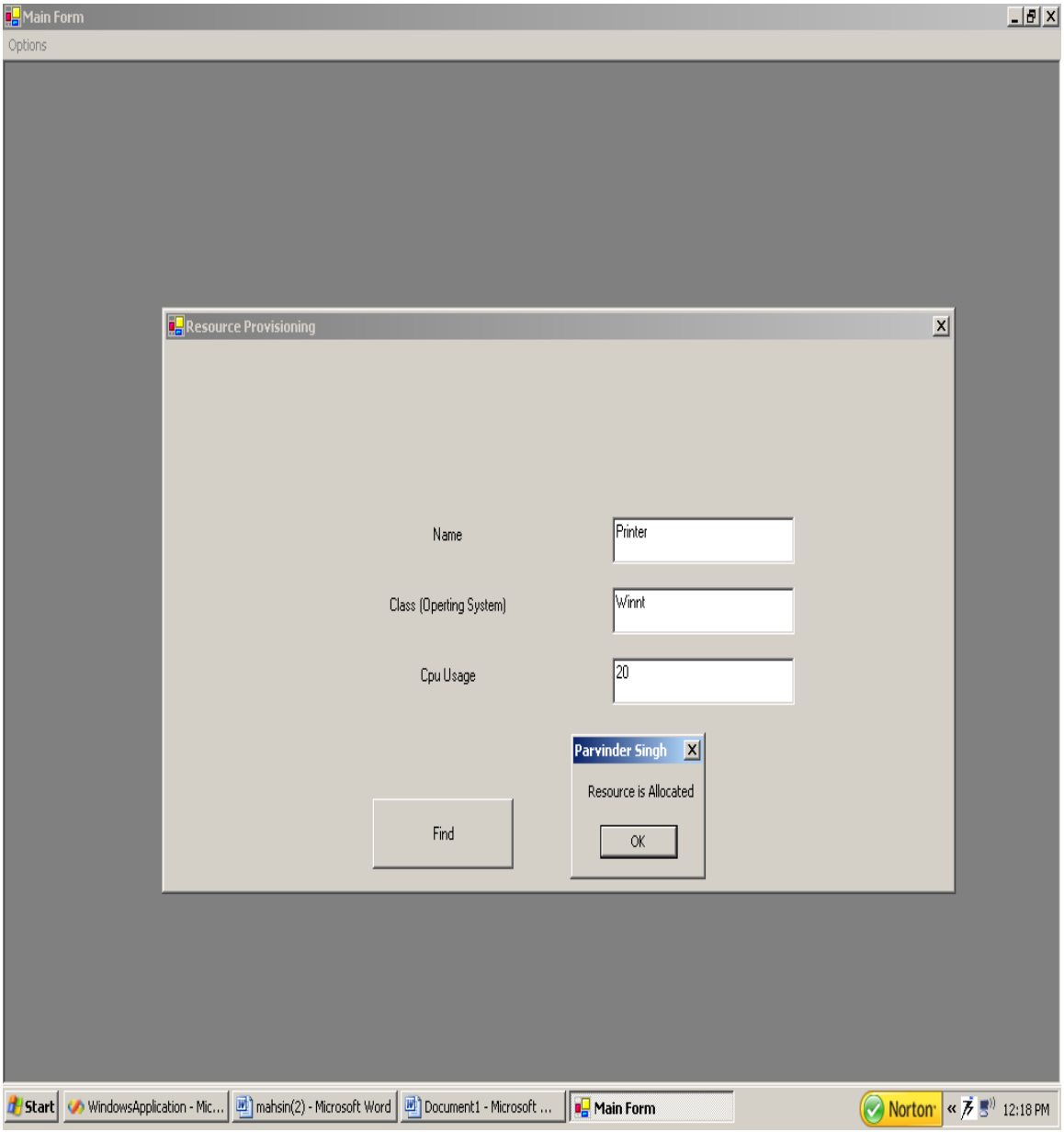

Figure 6.7 Resource Reservation

Reserved: The user enters the list of resources and time for which the resources are to be provisioned. This feature than displays the message notifying that resources are allocated.

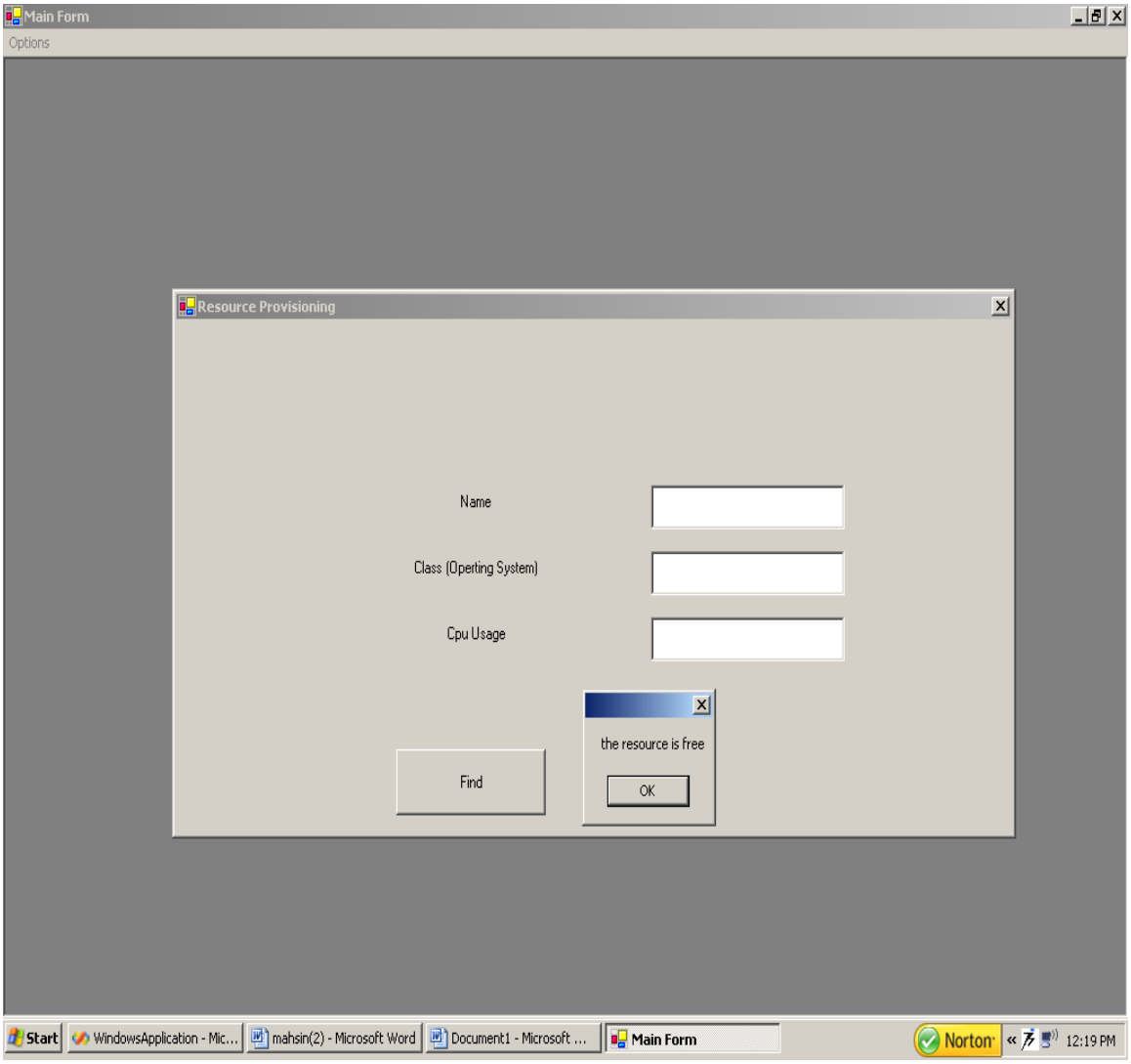

Figure 6.8 Resources De-allocated

De-allocation: The user uses the resources for the time entered and after the expiration of this time the resources are de-allocated. The entries for de-allocated resources are removed from reservation database and message is displayed.

6.2 Execution Time

For provisioning execution time of an application is important so that needs of large number of applications can be met. One of the way for finding the execution time is using historical information. Based on the parameters given below, the input parameters are matched against historical information and execution time is predicted depending on the degree of matching. In our thesis the historical database used has been taken from Thapar Grid, which contains information about execution time of different application executed in the past.

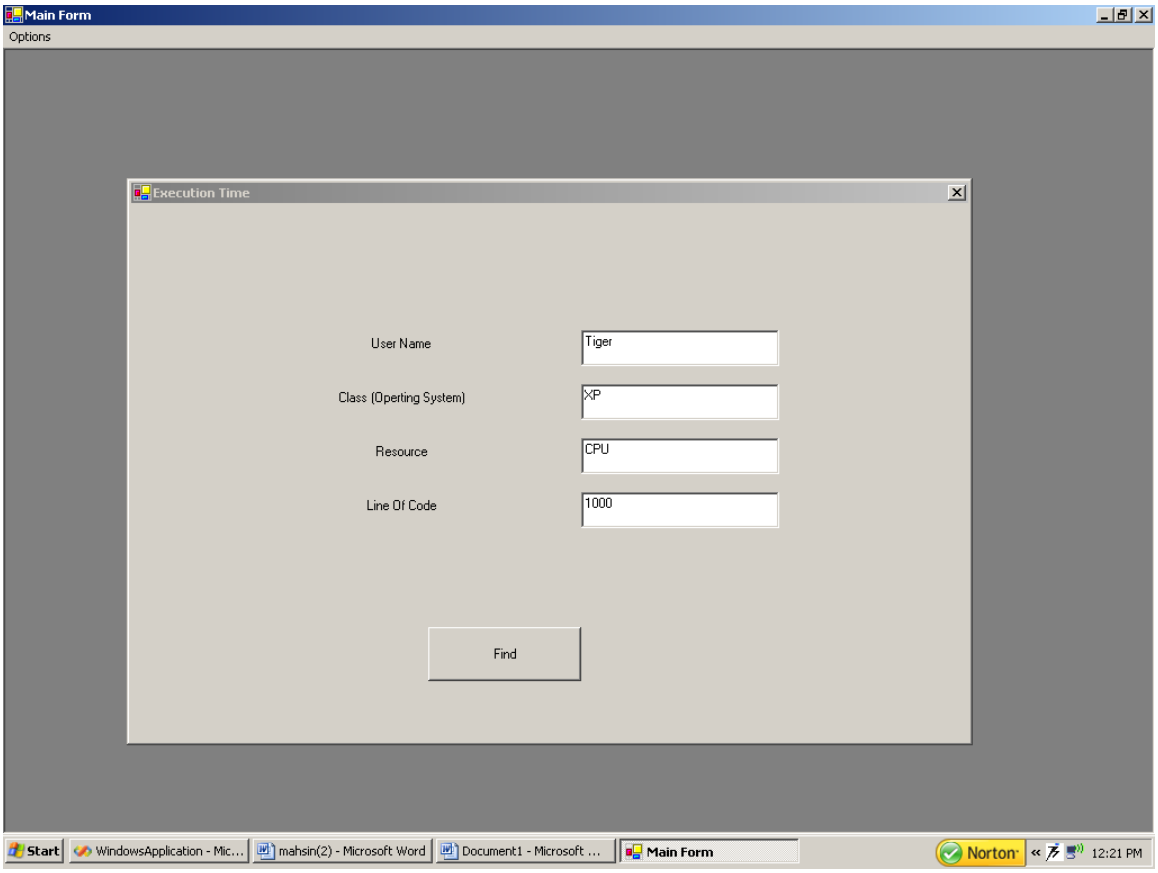

Figure 6.9 List of input parameters

Input Parameter: In order to predict the execution time of an application the input parameters are matched against historical database. The parameters on which output is predicted are:

- User Name
- Operating System
- Resource
- Line of Code

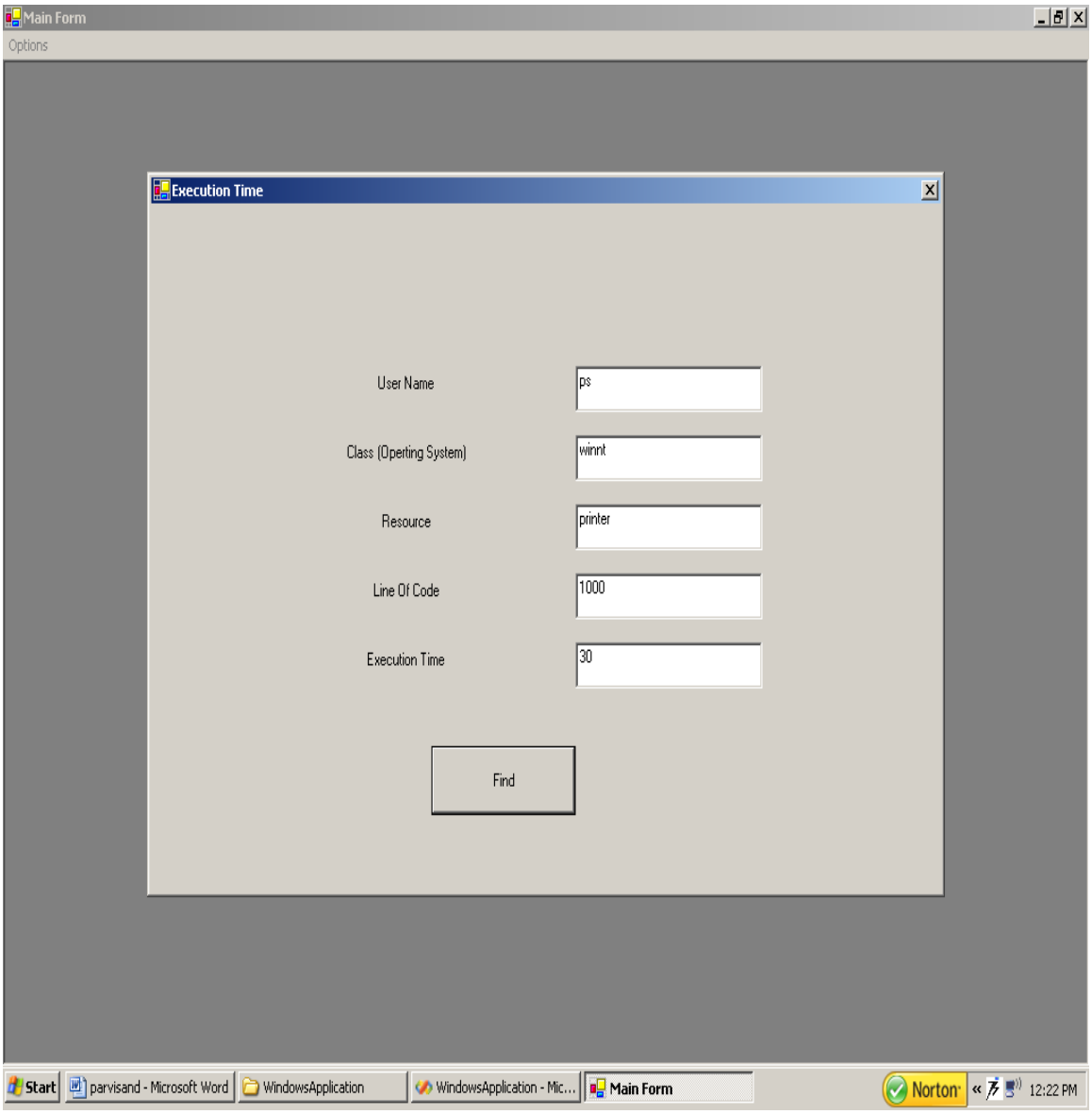

Figure 6.10 Execution Time

Output of Execution: The input parameters are matched against the historical database and the output is given in the time of execution time. This information is used to schedule different application so that more requests can be fulfilled using limited resources.

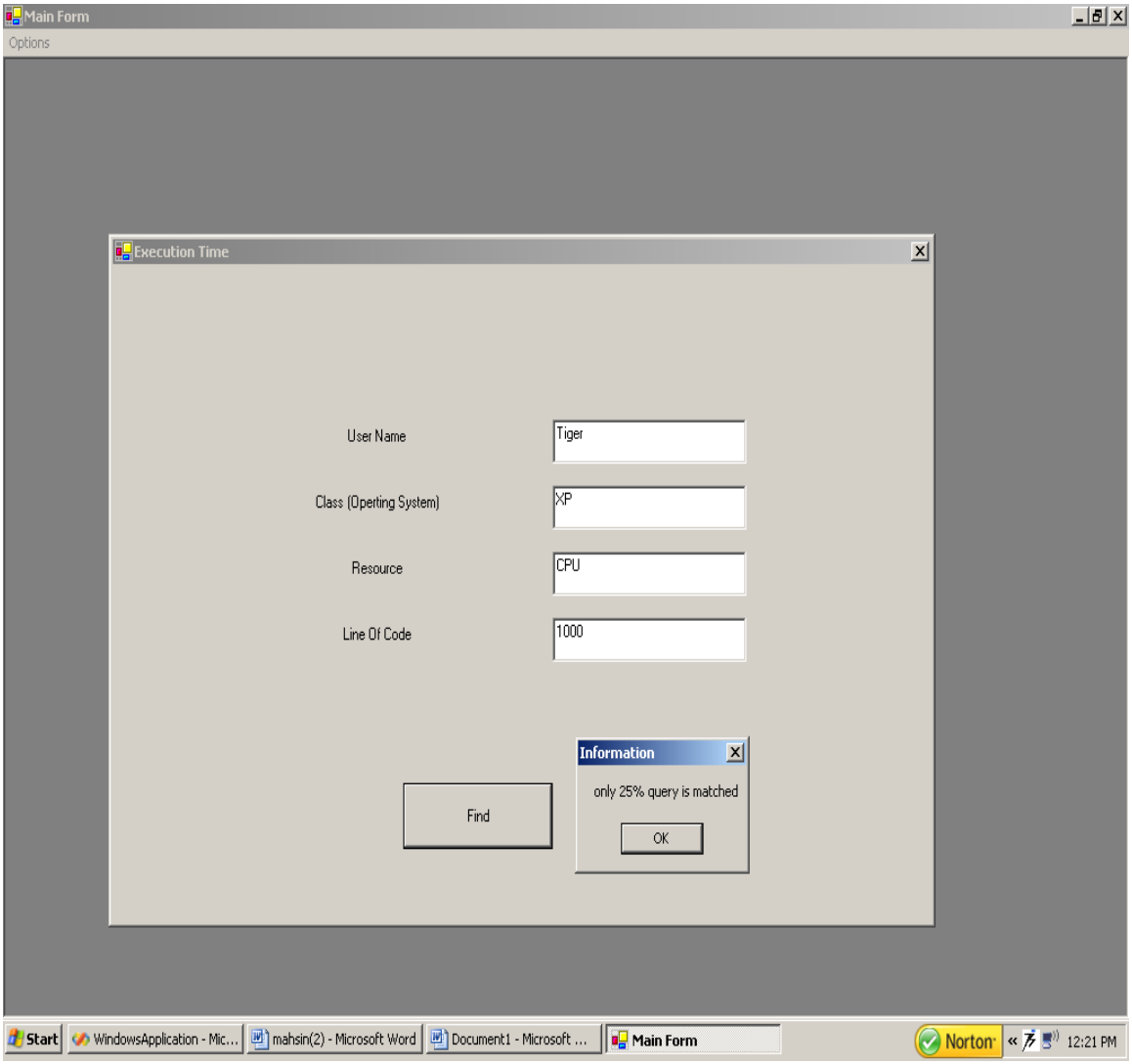

Figure 6.11 Percentage of matching

Output of Execution: The input parameters are matched against the historical database and the output gives the degree to which query is matched. Using this information the execution time is predicted the required level of confidence.

6.3 Implementing Web Service

As discussed in chapter 4, Grid applications are developed using one of existing grid infrastructure middleware technologies, such as Globus, Legion, ICENI, and others making them tightly coupled to the underlying middleware. Grid applications demand varying degrees and forms of QoS support from their grid middleware. Owing to the complex nature of these QoS requirements, it is not feasible for a single grid infrastructure middleware to provide an end-to-end solution that addresses all these challenges. A promising way to address the challenges in developing grid applications is to resolve the configuration, management, and deployment challenges of deploying QoS-enabled grid as Web services. Based on architecture proposed earlier, web service was developed as part of the thesis work and Alchemi's Cross-platform manager was used to submit job to the manager and then it was executed .The results obtained are shown in the form of screen shots:

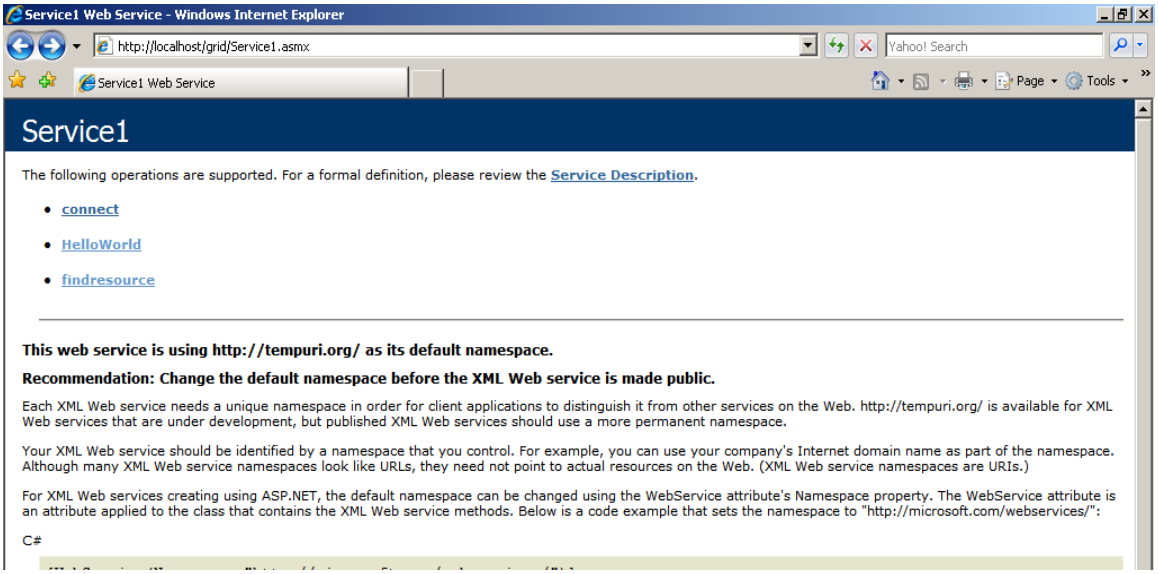

Figure 6.12 Service Invocation

Description: The Web services required to implement resource provisioning are:

- **Connect:** This service uses the IP address to connect to the database having list of resources that can be provisioned.
- **Find Resources:** This service takes as input the list of resources and search them in the required database for provisioning.

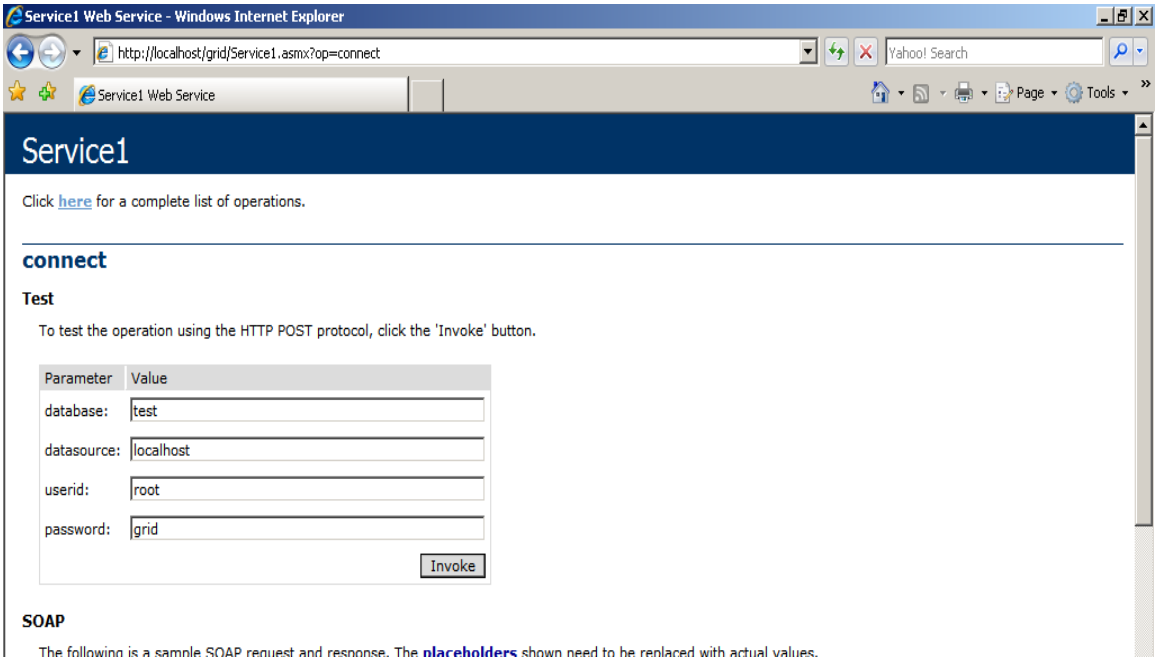

Figure 6.13 Database Connectivity

Description: The user enters the name of the database, the host having the list of resources and other authentication information required to connect to database.

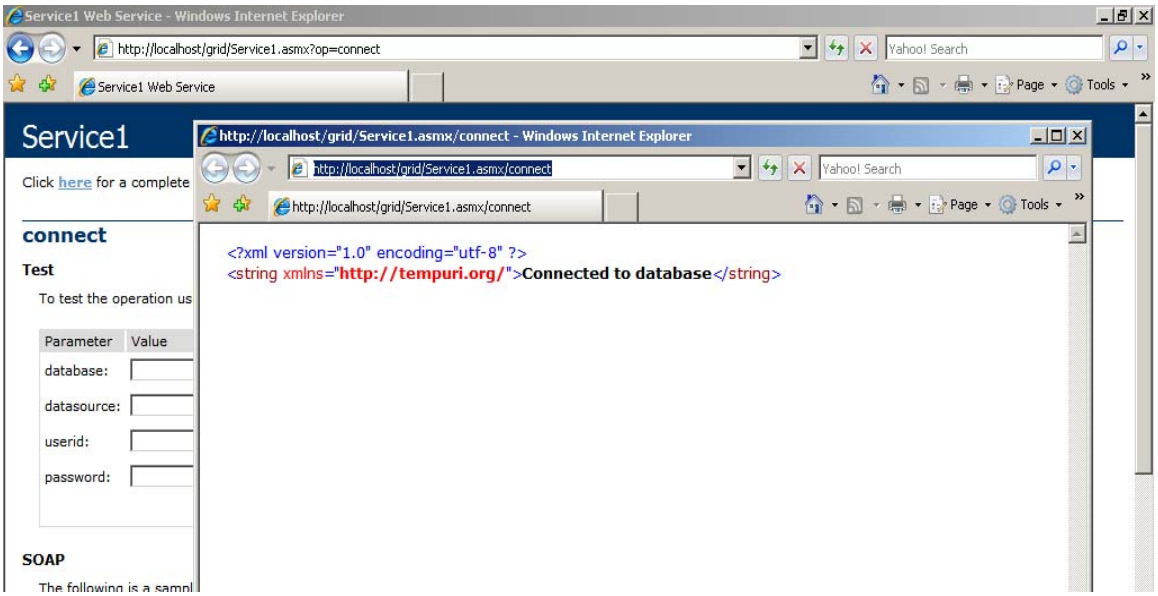

Figure 6.14 Connected To Database

Description: The information provided by user is used to connect to the database having resource information and than service is invoked. If connection to the database is established successfully than success message is displayed.

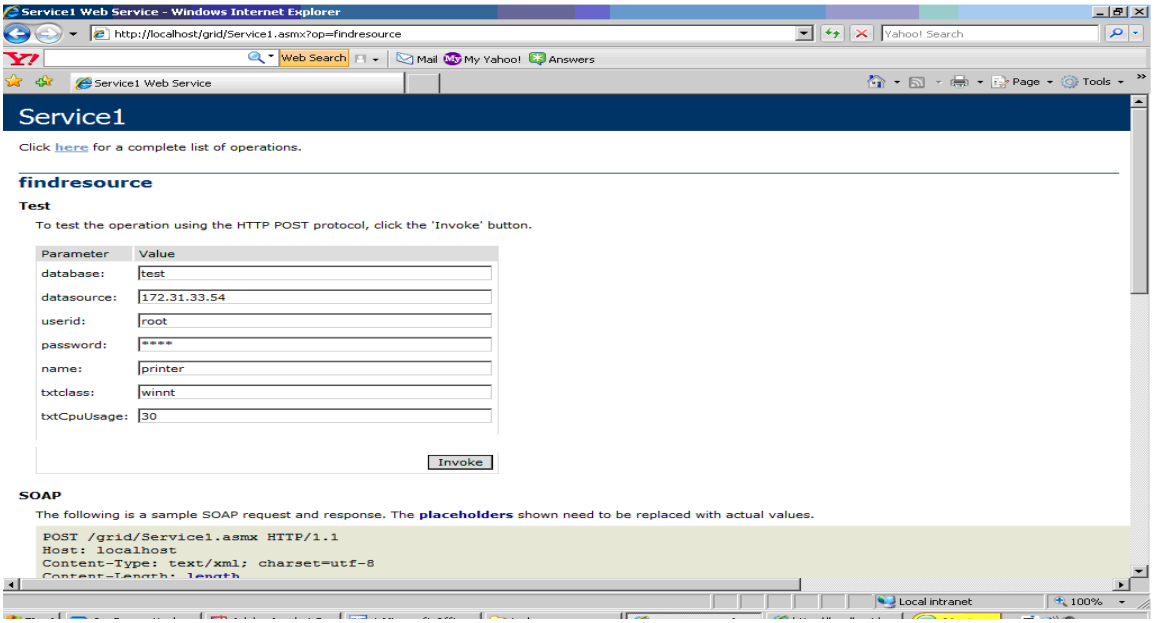

Figure 6.15 Finding resources

Description: This service is used to search for the resources required to run an application. The user enters the list of resources and than service is invoked to search for the required resources.

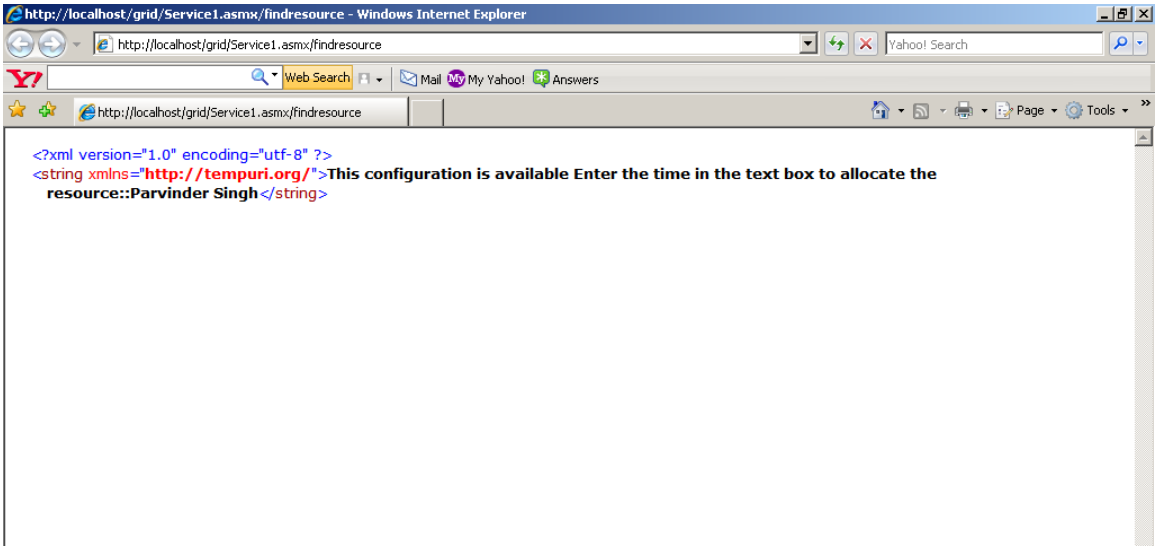

Figure 6.16 Resource Availability

Description: The service uses the resource requirement provided by user to search for the resources. If resources are found than success message is displayed.

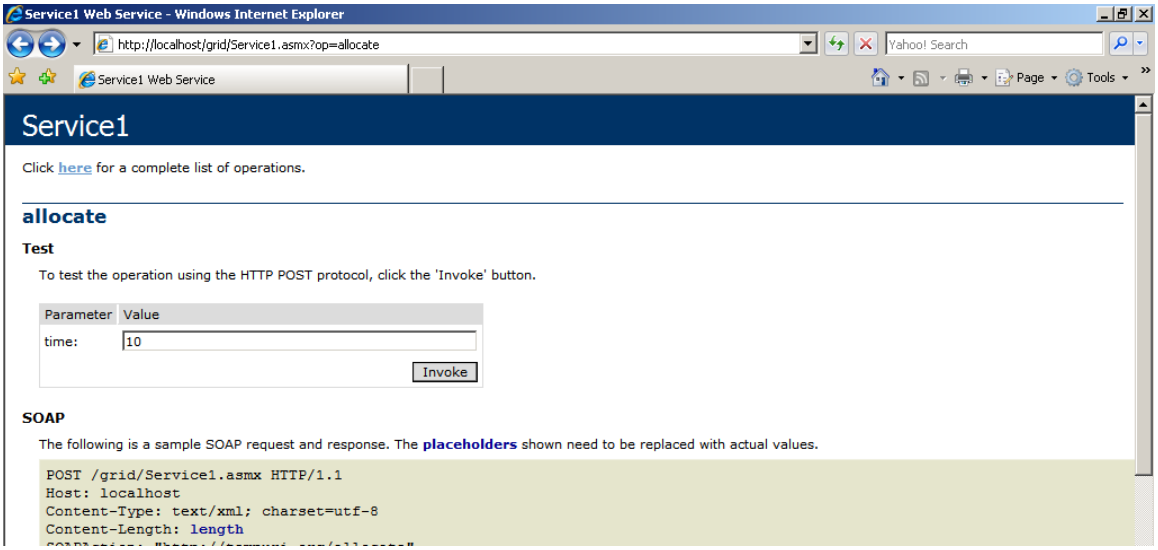

Figure 6.17 Enter Time for Allocation

Description: After the resources required by the user are found to be available, the user is asked to enter the time for which the resources are required. Thereafter the service is invoked to allocate resources to the user.

The resources will be provisioned for the user and after completion of the application and expiration of the time of provisioning the resources will be freed and returned back to the database.

This chapter focused on resource provisioning architecture and application used to implement it along with prediction of execution time of application. The results obtained running web services on Alchemi were shown. The web service implemented can be called using Cross Platform Manager and used in advanced reservation of resources. The next chapter summarizes this thesis work and suggests features that can be incorporated in future for an enhanced resource provisioning.

Chapter 7

Conclusions and Future Scope of Work

This thesis work provided an insight into Grid computing, the various approaches and algorithms currently used for Resource Provisioning in the Grid environment, and the problems found in these approaches. Then it described the framework for resource provisioning on Alchemi using Cross Platform Manager and web service and finally implementation of web service on Alchemi as solution to problems in traditional approaches.

6.1 Conclusions

At the heart of the Grid is the ability to discover, allocate, and negotiate the use of network accessible capabilities—be they computational services offered by a computer, application services offered by a piece of software, bandwidth delivered on a network, or storage space provided by a storage system. There are many facets to acquiring capabilities for a Grid application, and the term resource management describes all the aspects of the process: locating a capability, arranging for its use, and monitoring its state. Resource provisioning, in which resource capability is made available at a specified point of time, and for a specified duration is one of the key aspects of Resource Management.

Resource Management in existing grid infrastructure middleware, such as Globus, ICENI, and Legion, offer simplified application programming interfaces (APIs) for deploying Grid applications. However, Grid applications using these APIs become tightly coupled to their respective middleware infrastructure creating an impediment to interoperability, portability, maintenance and extensibility. Conventional grid infrastructure middleware provide only the means to securely access the resources from different service provider and the responsibility of reserving and accessing the resources is still the responsibility of the Grid applications.

Resource Management Systems make use of Resource Provisioning to search resources, select appropriate resources from them and than map application to those resources for execution. Selecting resources in advance and reserving them for execution guarantees complete execution of application and Resource provisioning helps in reservation of

resources in advance. However, as mentioned before most of the existing middleware uses APIs, which restricts interoperability between different platforms. One of the solution to overcome this problem is to implement Resource Provisioning using Web services. A web service provides functionality, which can be invoked and used from any platform, providing interoperability and portability.

This thesis attempts to solve the problems mentioned above and discussed in chapter 4 and provides a framework for resource provisioning in Alchemi using web service which helps:

- Free application developers from dependencies on particular software APIs, which ensures that the models can be used for a long time, even as existing software APIs become obsolete and replaced by newer ones.
- Provides interoperability, without concern for the middleware/OS platforms.
- Decouple Grid applications from the underlying Grid middleware by exposing the Grid applications as a Web service.
- Grid applications can then use standards-based web protocols such as http to access the underlying Grid middleware.

6.2 Future Scope of work

Resource Provisioning web service can be further improved by incorporating end-to-end multiple quality of service (QoS) properties, such as delay guarantees, security, scalability, high reliability and availability guarantees, and bandwidth and throughput guarantees to Grid applications.

Also, in future a feature of advance reservation can be incorporated in which resources can be reserved days before requirements and user can notify resource owner before some predefined time whether he will be using the resources or not. In case he does not wish to use the resources the resources can be reserved again and used by others

References

[1] Rajkumar Buyya and Srikumar Venugopal. "A gentle Introduction to Grid Computing and Technologies"http://www.gridbus.org/papers/GridIntro-CSI2005.pdf

[2] Introduction to Grid Computing with Globus, http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf

[3] Manish Parashar, The Applied Software system Laboratory Rutgers," Grid Computing: My View" The State University of Jersey http://www.caip.rutgers.edu/TASSL/Papers/gc-overview-mp-09-03.pdf

[4] Ian Foster, Argonne National Laboratory & University of Chicago, "What is the Grid? A Three Point Checklist," http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf

[5] Jim Waldo,Geoff Wyant,Ann Wollrath,Sam Kendall,"A Note on Distributed Computing", http://research.sun.com/techrep/1994/smli_tr-94-29.pdf.

[6] Victors Berstis,"Fundamentals of Grid Computing", http://www.redbooks.ibm.com/redpapers/pdfs/redp3613.pdf

[7] EU Data Grid Project, March 2004. http://www.eu-datagrid.org

[8] Jean-Christophe Durand," Grid Computing Conceptual and practical study:" http://www.hec.unil.ch/cms_inforge/Durand.pdf.

[9] Ian Foster, Carl Kesselman , Steven Tuecke," The Anatomy of the Grid", http://www.globus.org/alliance/publications/papers/anatomy.pdf.

[10] Ferreira, L., Bieberstein, N., Berstis, V., Armstrong, J., "Introduction to Grid Computing with Globus," Redbook, IBM Corp., http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf

[11] Foster, I., Kesselman C., Nick J.M. and Tuecke S. "Grid services for distributed system integration," Computer, vol. 35, no. 6, pp. 37–46, June 2002, http://www.globus.org/alliance/publications/papers/ieee-cs-2.pdf

[12] Foster, I., Kesselman, C. and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001. http://www.globus.org/research/papers/anatomy.pdf

[13] Chaitanya Kandagatla," Survey and Taxonomy of Grid Resource Management Systems", http://www.cs.utexas.edu/~browne/cs395f2003/projects/KandagatlaReport.pdf.

[14] Phil Schater, "Resource provisioning, Interoperability, and XML" http://xml.coverpages.org/XRPM-ProvForum091001.pdf,

[15] Ronald P. Doyle, Jeffrey S. Chase,Omer M. Asad, Wei Jin,Amin M. Vahdat , "Model-Based Resource Provisioning in a Web Service Utility," http://issg.cs.duke.edu/publications/mbrp-usits03.pdf

[16] James Frey, Todd Tannenbaum, Miron Livny Ian Foster, Steven Tuecke," Condor-G: A Computation Management Agent for Multi-Institutional Grids", http://www.cs.wisc.edu/condor/doc/condorg-hpdc10.pdf.

[17] "Project MegaGrid: Tools and Techniques for Performance Monitoring and Resource Provisioning" An Oracle, Dell, EMC, Intel Joint White PaperSeptember 2005,http://www.oracle.com/technologies/grid/docs/twp_megagrid_resource_provisioning.pdf

[12] Foster, I. and Kesselman, C., "The Grid: Blueprint for a New Computing Infrastructure," San Mateo, CA: Morgan Kaufmann, 1999.

[13] Foster, I., Kesselman C., Nick J.M. and Tuecke S. "Grid services for distributed system integration," Computer, vol. 35, no. 6, pp. 37–46, June 2002, http://www.globus.org/alliance/publications/papers/ieee-cs-2.pdf

[14] Foster, I., Kesselman C., Nick, J.M., and Tuecke, S., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002, http://www.globus.org/alliance/publications/papers/ogsa.pdf

[15] Foster, I., Kesselman, C. and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001. http://www.globus.org/research/papers/anatomy.pdf

[16] Thomas Lehman, Jerry Sobieski, Bijan Jabbari, "DRAGON: A Technique for Service Provisioning in Heterogeneous Grid Networks," http://ieeexplore.ieee.org/iel5/35/33764/01607870.pdf

[18] Foster, I., Roy A., Sander, V., "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation," 8th International Workshop on Quality of Service, 2000.

[19] Gurmeet Singh, Carl Kesselman, Ewa Deelman., "Application-level Resource Provisioning on the Grid", http://pegasus.isi.edu/publications/Singh-AppResProvisioning.pdf

[20] Mandal, A., et al. Scheduling Strategies for Mapping Application Workflows onto the Grid. in The 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14). 2005.

[21] A. Mayer, S. McGough, M. Gulamali, L. Young,J. Stanton, S. Newhouse, and J. Darlington. Meaning and behaviour in grid oriented components. In 3rd International Workshop on Grid Computing, Grid 2002, volume 2536 of Lecture Notes in Computer Science, Baltimore, USA, November 2002.

[22] N. Furmento, A. Mayer, S. McGough, S. Newhouse,T. Field, and J. Darlington. Optimisation of component-based applications within a grid environment. In Proceedings of the 2001 ACM/IEEE conference on Supercomputing. ACM Press, November10-16 2001.

[23] Mumtaz Siddiqui, Alex Villaz´on, J¨urgen Hofer and Thomas Fahringer. "GLARE: A Grid Activity Registration, Deployment and Provisioning Technique," http://vtcpc.isi.edu/wiki/images/e/e7/Askalon3.pdf.

[24] Ewa Deelman et. al. Mapping abstract complex workflows onto grid environments. Journal of Grid Computing, LNCSD9, ISSN 1570-7873, 1:25–39, 2003.

[25] Ian Foster, Jens Vockler, Michael Wilde, and Yong Zhao. Chimera: A Virtual Data System For Representing, Querying, and Automating Data Derivation. In 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002), September 2002.

[26] Ewa Deelman et. al. Transformation Catalog Design for GriPhyN. Technical

report griphyn-2001-17, 2001.

[27] Ian Foster, Jens Vockler, Michael Wilde, and Yong Zhao. Chimera: A Virtual Data System For Representing, Querying, and Automating Data Derivation. In 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002), September 2002.

[28] Ken Kennedy et.al." New Grid Scheduling and Rescheduling Methods in the GrADS Project." In International Parallel and Distributed Processing Symposium, Workshop for Next Generation Software.IEEE Computer Society Press, April 2004.

[29] The Condor Team. http://www.cs.wisc.edu/condor/.

[30] H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-based Resource Matching in the Grid-The Grid meets the Semantic Web. In Second International Semantic Web Conference, Sanibel-Captiva Islands, Florida. October 2003.

[31] Cross Grid. http://www.crossgrid.org/.

[32] H.N. Lim, Choi Keung, J.R.D. Dyson, S.A. Jarvis, andG.R. Nudd. Performance Modelling of a Self-Adaptive and Self-Optimising Resource Monitoring System for Dynamic Grid Environments. In AHM2005, Fourth All Hands Meeting, Nottingham, 2005.

[33] POVray. http://www.povray.org/.

[34] Piotr Poznanski, German Cancio Melia, Rafael Garca Leiva, and Lionel Cons. Quattor - a technique for managing grid-enabled large-scale computing fabrics. In workshop of the 4th Cracow Grid Workshop 2004, December 2004.

[35] IT Innovation. Workflow enactment engine, October 2002. http://www.itinnovation.soton.ac.uk/mygrid/workow/.

[36] Corporation for national Resource Initiatives. HandleSystem. http://www.handle.net.

[37] Aniruddha Gokhale1 and Balachandran Natarajan., "Composing and Deploying Grid Middleware Web Services using Model Driven Architecture", http://www.dre.vanderbilt.edu/~gokhale/WWW/papers/doa02-grid.pdf

[38] W3C. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, August 3, 2004. W3C Working Draft.

[39] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal," Alchemi: A .NET-based Grid Computing Technique and its Integration into Global Grids", http://gridbus.csse.unimelb.edu.au/papers/Alchemi.pdf.

[40] Thanapol Rojanapanpat and Putchong Uthayopas," OpenUCI: A .NET Based Utility Computing Framework", http://hpcnc.cpe.ku.ac.pdf.

[41] UDDI: Universal Description, Discovery and Integration. www.uddi.org.

[42] W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, August 3, 2004. W3C Working Draft http://www.w3.org/TR/2004/WD-wsdl20-20040803

Paper Published

Parvinder Singh, Ms. Inderveer Chana, "**Resource Provisioning in a Grid Environment**", National Seminar on "Recent Advances on Information Technology" **(RAIT-2007)**, Indian School of Mines University,Dhanbad, India, February 26-27, 2007 **(Published in Proceedings).**