Performance Optimization in Grid Resource Allocation Using Linear Programming

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

Master of Engineering
in
Software Engineering

By:
Anurag Gothi
(Roll No. 8053107)

Under the supervision of
Dr. (Mrs.) Seema Bawa
Professor & Head, CSED
Thapar University, Patiala.

MAY 2007

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Performance Optimization in Grid Resource Allocation Using Linear Programming"**, 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 Dr. (Mrs.) Seema Bawa.

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

(***Anurag Gothi)***

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

(Dr. (Mrs.) Seema Bawa)
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

First and foremost, I would like to express my sincere gratitude to my guide **Dr.(Mrs.) Seema Bawa**, Professor and Head, Computer Science and Engineering Department for immense help, guidance, stimulating suggestions and encouragement all the time with this thesis work. She always provided a motivating and enthusiastic atmosphere to work with; it was a great pleasure to do this thesis under her supervision. The successful completion of this thesis is a direct consequence of the moral and material support extended by **Centre for Excellence in Grid Computing**, TU throughout this thesis.

I am equally grateful to **Dr. Maninder Singh**, Assistant Professor, Computer Science and Engineering Department, Ms. Inderveer Chana, Lecture Computer Science and Engineering Department, Mrs. Shivani Goel, P.G. Coordinator, Computer Science and Engineering Department for the motivation and their invaluable suggestions that triggered me for my thesis work.

I would also like to thank all the staff members and PhD Scholars Ms Anju Sharma and Ms. Shashi Bhanwar who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of my thesis.

I would also like to express my appreciation to my co-worker and my great friends Abhishek Varshney, Santosh Jaiswal, Ratnesh Amarnath, Puneet Khurana, Gaurav Vohra, Ruchi Garg; it was a great pleasure working with them in this thesis together.

Finally, my special thanks go to authors whose works I have consulted and quoted in this work. The most valuable thing I acquired from the two years study in TU is to protect, survive and endure myself in this world with increased confidence and additional faith in my abilities to achieve. 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.

Anurag Gothi

(8053107)

Abstract

In Grid computing environment, range of computing devices coexists stating from personal computers to supercomputers. These devices are inter-connected to provide a variety of computational capabilities in order to execute applications that have diverse requirements. An important decision for such computing infrastructure is how to optimally allocate computational and communication resources to these applications and to schedule their execution in order to maximize performance benefits. It is apparent that resource heterogeneity impacts the resource allocation in quite significant way in terms of overall performance, reliability, robustness, scalability and fault tolerance. So, the system managing these complex environments needs to be scalable, reliable, smart and adaptable to change its allocation mechanism depending upon the environment and its user requirements. Therefore, a scalable approach for resource allocation where the system can adapt itself to the changing environment and fluctuating resources, is essentially needed.

Given limited resources and competing constraints we can formulate the problems as the problem of maximizing or minimizing an objective function. So, we specify the objective function as a linear function of certain variables, and constraints on resources as equalities or inequalities on those variables, and we get Linear Formulation of the problem. For this reformulated problem; a linear programming based resource allocation method is proposed. The method is composed of a large numbers of independent tasks, with the goal of using idle computing resources.

The method has been implemented using Java. The result obtained by this method demonstrates the proposed method optimally allocates resources on Grids and finally compared with Round Robin Approach

TABLE OF CONTENTS

CONTENTS **PAGE NO.**

LIST OF FIGURES

CONTENTS **PAGE NO.**

Chapter1. Introduction

In Distributed computing environment where different computing nodes simultaneously run an application located on different computers that are interconnected by a network locally or globally. Grid computing is distributed computing taken to the next evolutionary level. The ever-growing computational power requirements have led to the emergence of Grid computing. Grid computing uses the resources of many separate computers connected by a network (usually Internet) to solve large-scale computational problems. The resources may involve CPU cycles, memory, bandwidth, disk, applications, databases on remote systems, scientific data in some organizations for research purposes etc. These heterogeneous resources are distributed and owned geographically by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce. In resource allocation mechanism access of resources takes place through selection of jobs. The scheduling mechanism employs different strategies to determine which jobs are executed at which time on which resources and at what prices. As the resources are owned by different organizations and each organization has its own resource usage policy and different pricing systems, we have to define a resource allocation strategy that respects every resource owner's resource usage policies. In this thesis, optimized resource allocation mechanisms in terms of optimized performance and cost incurred on different Grid environment are proposed.

To deal with the resource allocation problem, in this work linear programming based resource allocation mechanism has been used. In linear programming based resource allocation each resource allocation is a variable and locations are viewed as constraints. Basically, in this thesis a mathematical approach has been proposed to introduce organization of resources, so that Grid can modify its infrastructure as a function of the current workload and its structural state.

1.1 Motivation

Resource management is an important infrastructure in the Grid computing environment. The management of resources in Grid environment becomes complex as the resources are geographically distributed, heterogeneous in nature, owned by different individual or organizations with their own policies, have different access and cost models, and have dynamically varying loads and availability. Due to the highly heterogeneous and complex computing environments, availability of resources may fluctuate in Grid environments; therefore it is necessary to design a mechanism to allocate resources efficiently in such infrastructure. Managing resources (resource discovery, resource selection and resource access) in highly dynamic networks where the availability of resources change at a high rate, requires a mechanism that is scalable, adaptable, robust and reliable.

The conventional resource management schemes are based on relatively static model that have centralized controller that manages jobs and resources accordingly. These management strategies might work well in those scheduling regimes where resources and tasks are relatively static and dedicated. However, this fails to work efficiently in many heterogeneous and dynamic system domains like Grid where jobs need to be executed by computing resources, which are difficult to predict. The complex, heterogeneous and dynamic systems present new challenges in resource management such as: scalability, adaptability and reliability. This poses numerous research questions:

- In Grid environment, computing nodes can join and leave the system at any time without any dedicated commitment, another issue concerned for Grid management is how to tolerate such failures and recover from them when crashes occur? How to re-allocate task and resource automatically?

- How to manage the intricacy, complexity, and performance analysis in such dynamic system where nothing is predictable?

- Grid is a dynamic environment where location, type and performance of components are constantly changing; hence their computational requirement

varies over time. How to adapt to such conditions when computational capacity fluctuates in high ratio?

1.2 Thesis Organization

This section discusses the organization of this thesis. This thesis is organized as follows:

Chapter 2 reviews the background of this thesis from two aspects: First, it discusses about distributed computing its motivation, issues and challenges. Second, it presents Grid computing, its characteristics, its challenges.

Chapter 3 reviews and analyzes resource allocation in Grid environment. First part defines the resource allocation problem then it reviews the structure of such environment. Second, it summaries resource management issues and challenges in Grid and currently available approaches to overcome these challenges.

Chapter 4 introduces problem formulation for resource allocation in Grid environment.

Chapter 5 introduces a framework for solving resource allocation problem in Grid environment. It presents resource management infrastructure based on linear formulation and presents its system architecture.

Chapter 6 discusses about the implementation of linear formulation mechanism. It also discusses the process of the experiment. At first, it defines the experimental model, process steps and evaluation criteria. Finally, it analyzes the results and presents main findings of the experimentation.

Chapter 7 presents the conclusion of the thesis, describes the main contribution of the thesis and highlights future research direction.

Chapter 2. Introduction to Grid Computing

Grid computing and Peer-to-Peer computing is observed as promising next generation computing platforms. They enable the creation of virtual enterprises for sharing resources distributed across the world. However, resource management, application development and usage models in these environments are complex undertakings. This is due to the geographic distribution of resources that are owned by different organizations or peers. The resource owners of each of these resources may have different usage or access policies and cost models, and varying loads and availability.

2.1 Concept of Grid

The term "Grid" was introduced in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. Considerable progress has since been made on the construction of such an infrastructure, but the "Grid" has also been conflated, at least in popular perception, to embrace everything from advanced networking to artificial intelligence. Definition is describes as: "A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [21].

The definition was refined 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"[22]. 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. There was noted: "The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is 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 we call a virtual organization."

2.2 Virtual Organizations (VOs)

Computational Grids and peer-to-peer (P2P) computing systems are emerging as a new paradigm for solving large-scale problems in science, engineering and commerce. They enable the creation of Virtual Enterprises (VEs) or Virtual Organizations (VOs) for sharing and aggregation of numerous resources geographically distributed across organizations and administrative domains. They comprise:

- Heterogeneous resources (such as PCs, workstations, clusters and supercomputers),
- Management systems (such as single system image OS, queuing systems), supporting
- Different kind of applications (such as scientific, engineering and commercial), and demanding
- Various requirements (for CPU, I/O, memory and/or network intensive).

The producers (resource owners) and consumers (resource users) most of the time have different goals, objectives, strategies and supply-and-demand patterns. More importantly, both resources and end-users are geographically distributed with different time zones. In managing such complex environments, traditional approaches to resource management that attempt to optimize system-wide measure of performance cannot be employed. Traditional approaches use centralized policies that need complete state information and a common management policy, or a decentralized consensus-based policy. Due to the complexity in constructing successful Grid-aware environments, it is impossible to define an acceptable system-wide performance matrix and common fabric management policy.
The main problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not file exchange but rather direct access to computers, software, data, and other resources, as is 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. The following are examples of VOs:

The application service providers, storage service providers, cycle providers, and consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory. This scenario may involve members of industrial consortium bidding on a new aircraft, a crisis management team using simulation systems that they use to plan a response to an emergency situation, and finally members of large, international, multiyear high energy physics collaboration. Each of these examples represents an approach to computing and problem solving based on collaboration in computation- and data-rich environments.

VOs vary tremendously in their purpose, scope, size, duration, structure, community, and sociology. Nevertheless, careful study of underlying technology requirements leads us to identify a broad set of common concerns and requirements. In particular, we see a need for highly flexible sharing relationships, ranging from client – server to peer-to-peer, and for sophisticated and precise levels of control over how shared resources are used, including fine-grained and multi-stakeholder access control, delegation, and application of local and global policies. Furthermore, there is a great need for sharing of varied resources, ranging from programs, files, and data to computers, sensors, and networks, and for diverse usage modes, ranging from single user to multi user and from performance sensitive to cost-sensitive. Issues of quality of service, scheduling, co-allocation, and accounting are also very important.

Current distributed computing technologies do not address the concerns and requirements just listed. For example, current Internet technologies address communication and information exchange among computers, but do not provide integrated approaches to the coordinated use of resources at multiple sites for computation. Business-to-business exchanges focus on information sharing often via centralized servers. So do virtual enterprise technologies, although in this case sharing may eventually extend to applications and physical devices. Enterprise distributed computing technologies such as

CORBA and Enterprise Java enable resource sharing within a single organization. The open group's Distributed Computing Environment – DCE supports secure resource sharing across sites, but most VOs would find it too burdensome and inflexible.

It is here that Grid technologies enter the picture. Over the past ten years, research and development efforts within the Grid community have produced protocols, services, and tools that address precisely the challenges that arise building scalable VOs. These technologies include security solutions that support management of credentials and policies when computations span multiple institutions; resource management protocols and services that support secure remote access to computing and data resources, and the co-allocation of multiple resources; information query protocols and services that provide configuration and status information about resources, organizations, and services; and data management services that locate and transport datasets between storage systems and applications. However, because of their focus on dynamic cross-organizational sharing, Grid technologies complement rather than compete with existing distributed computing technologies. For instance, enterprise distributed computing systems can use Grid technologies to achieve resource sharing across institutional boundaries or establish dynamic markets for computing and storage resources.

2.3 Grid Classification

There are many Grid projects that build software tools, applications, and testbeds. These precursors of Grid are used for specific cases and can not be seen as a "virtual computer" that adaptable to user's needs in general case. However, they are the building blocks that will be used to fulfill the Grid's vision in the future. This view is correct but not complete. Grid can be used also in many other kinds of applications. The current Grid's implementations can be classified according to their uses and characteristic [44], [21].

2.3.1 Grid Uses

- **Computational intensive support (High performance computing):** allows applications to get enough computational resources in order to reduce the runtime of jobs that can not be solved or take very long time to run on a single system.

This can also be achieved by Parallel or Cluster systems. The different is that instead of using highly expensive resources (parallel, multiple processors) or required of homogeneous computers (in case of Cluster), a Grid implementation can make use of heterogeneous, cheap resources that are already available. However, there are challenges for the scalability of protocols and algorithms with a large number of nodes, latency-tolerant to achieving high levels of performance. Typical applications are weather forecast, simulations (car-crash, physical and chemical experiments).

- **Data intensive applications (increase data storage capability):** There are applications that generate terabytes of data per day, for example LHC@home [40], DataGrid [6]. The total generated data would exceed the capacity of any centralized storage system.

- **Optimize use of resources (High throughput computing):** CPUs are often in idle. Giving tasks to unused processor cycles is called CPU scavenging. Disk storage, when aggregate from millions users can provide a huge virtual storage system. Examples of these applications are: Monte Carlo simulations, BOINC [3] code-based projects (Seti@home; Folding@home; LHC@home, etc). Another example of high throughput computing is the aggregate of network bandwidth. If a user needs to increase his total bandwidth to the Internet (for example a Web mining search engine), the tasks can be split among Grid machines that have independent Internet connection. Within an organization where the computers may shared the Internet connection, there would not very much gain in bandwidth.

- **Collaborative works across multiple virtual organizations:** enabling and enhancing human-to-human interactions. Such applications also have characteristics of other application described above, as they enable the sharing of computational and storage resources. The challenging issues of these applications are real time requirements imposed by human perceptual capabilities and the rich

variety of interactions [21].

2.3.2 Grid Characteristics

- Large scale: the number of resources in a Grid can be range from just a few to millions.
- Geographical distribution: resources may be located at distant places.
- Heterogeneity: hardwares and softwares resources can be varied: different devices (super computers, PC, PDA, mobile phone, etc), different operating systems, etc.
- Resource sharing: users and organization contribute their resources to Grid.
- Multiple administrations: each virtual organization can establish different security policies under which their resources can be accessed and used.
- Resource coordination: coordinate use of resources to form an aggregate power.
- Transparent access: a Grid is seen as a single virtual computer.
- Predictable performance: A Grid should ensure several QoS requirements (e.g.: run time, availability of resources, etc). Even though, it is very difficult to achieve this character in a heterogeneous environment and the lack of central control. Several fault-tolerance techniques can be used to solve this problem such as resubmit jobs, re-allocation when some resources are down during the job execution, redundant job execution (multiple copies of job can be run on different machines), etc.

Consistent access: standard services protocol and interfaces to access resource from different Grid's implementation.

2.4 The Nature of Grid Architecture

In this section, we identify requirements for general classes of components in the Grid architecture. The result is an extensible, open architectural structure within which solutions to key VO requirements can be placed and which is organized in layers, as shown in figure 2.1. In specifying the various layers of the Grid architecture, we follow the principles of the "hourglass model" as presented in [23]. The narrow neck of the hourglass defines a small set of core abstractions and protocols e.g., TCP and HTTP on

the internet, onto which many different high-level behaviors can be mapped (the top of the hourglass), and which themselves can be mapped onto many different underlying technologies (the base of the hourglass). By definition, the number of protocols defined at the neck must be small. The neck of the hourglass consists of Resource and Connectivity protocols, which facilitate the sharing of individual resources. Protocols at these layers are designed so that they can be implemented on top of a diverse range of resource types, defined at the Fabric layer, and can in turn be used to construct a wide range of global services and application-specific behaviors at the Collective layer – so called because it involves the coordinated "collective" use of multiple resources. In the following sections we present the basic Grid architecture layers into more detail.

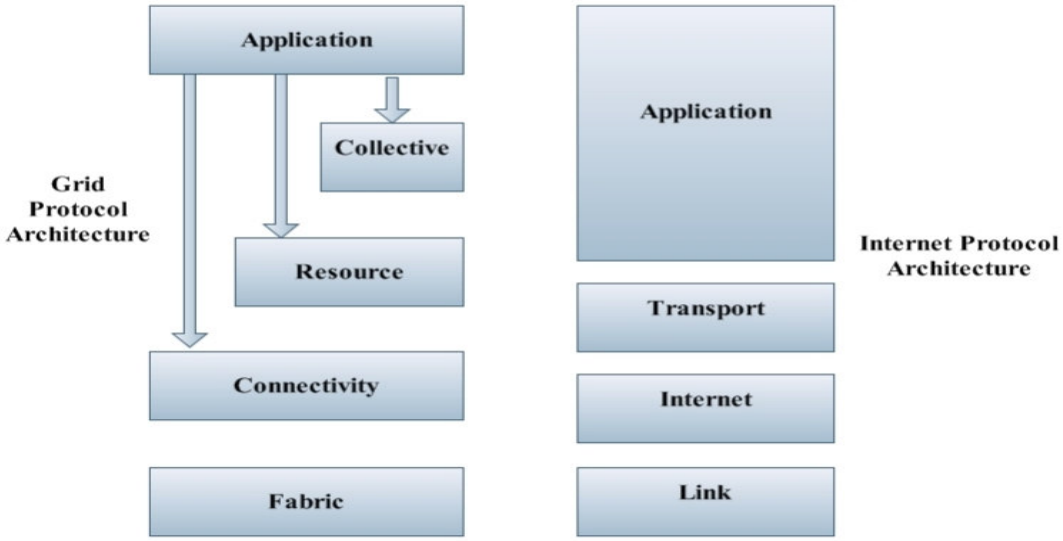


Figure 2.1: Grid Architecture's relationship with Internet Protocol Architecture [23]

2.4.1 Fabric Layer: Interfaces to Local Control

The Fabric layer provides the resources to which shared access is mediated by Grid protocols: for example, computational resources, storage systems, catalogs, network resources, and sensors. A resource may be a logical entity, such as a distributed file system, a computer cluster, or distributed computer pool; in such cases, a resource implementation may involve internal protocols e.g., the NFS storage access protocol or a

cluster resource management system's process management protocol; but these are not the concern of that Grid architecture.

Fabric components implement the local, resource-specific operations that occur on specific resources (physical or logical) as a result of sharing operations at higher levels. There is thus a tight and subtle interdependence between the functions implemented at the Fabric level, on the one hand, and the sharing operations supported, on the other. Richer Fabric functionality enables more sophisticated sharing operations; at the same time, if we place few demands on Fabric elements, then deployment of Grid infrastructure is simplified. For instance, resource-level support for advance reservations makes it possible for higher-level services to aggregate (co-schedule) resources in interesting ways that would otherwise be impossible to achieve.

Experience suggests that at a minimum, resources should implement enquiry mechanisms that permit discovery of their structure, state, and capabilities e.g., whether they support advance reservation or not on the one hand, and resource management mechanisms that provide some control of delivered quality of service, on the other. The following brief and partial list provides a resource-specific characterization of capabilities:

Computational resources: Mechanisms are required for starting programs and for monitoring and controlling the execution of the resulting processes. Management mechanisms that allow control over the resources allocated to processes are useful, as are advance reservation mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant state information such as current load and queue state in the case of scheduler-managed resources.

- Storage resources: Mechanisms are required for putting and getting files. Third-party and high-performance; e.g., striped transfers are useful [4]. So are mechanisms for reading and writing subsets of a file and/or executing remote data selection or reduction functions [36]. Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU) are useful, as are advance reservation mechanisms. Enquiry

functions are needed for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.

- Network resources: Management mechanisms that provide control over the resources allocated to network transfers (e.g., prioritization, reservation) can be useful. Enquiry functions should be provided to determine network characteristics and load.

- Code repositories: This specialized form of storage resource requires mechanisms for managing versioned source and object code.

- Catalogs: This specialized form of storage resource requires mechanisms for implementing catalog query and update operations: for example, a relational database [5].

2.4.2 Connectivity Layer: Communicating Easily and Safely

The Connectivity layer defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between Fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources. Communication requirements include transport, routing, and naming.

While alternatives certainly exist, these protocols are mostly drawn from the TCP/IP protocol stack: specifically, the Internet (IP and ICMP), transport (TCP, UDP), and application (DNS, OSPF, RSVP, etc.) layers of the Internet layered protocol architecture. This is not to say that in the future, Grid communications will not demand new protocols that take into account particular types of network dynamics. With respect to security aspects of the Connectivity layer, many of the security standards developed within the context of the Internet protocol suite are applicable.

According to [49], authentication solutions for VO environments should have the following characteristics:

Single sign on: Users must be able to "log on" (authenticate) just once and then have access to multiple Grid resources defined in the Fabric layer, without further user intervention.

- Delegation [24, 30, and 37]: A user must be able to endow a program with the ability to run on that user's behalf, so that the program is able to access the resources on which the user is authorized. The program should (optionally) also be able to conditionally delegate a subset of its rights to another program (sometimes referred to as restricted delegation).

- Integration with various local security solutions: Each site or resource provider may employ any of a variety of local security solutions, including Kerberos and UNIX security. Grid security solutions must be able to interoperate with these various local solutions. They cannot, realistically, require wholesale replacement of local security solutions but rather must allow mapping into the local environment.

- User-based trust relationships: In order for a user to use resources from multiple providers together, the security system must not require each of the resource providers to cooperate or interact with each other in configuring the security environment. For example, if a user has the right to use sites A and B, the user should be able to use sites A and B together without requiring that A's and B's security administrators interact.

Grid security solutions should also provide flexible support for communication protection e.g. control over the degree of protection, independent data unit protection for unreliable protocols, support for reliable transport protocols other than TCP, and enable stakeholder control over authorization decisions, including the ability to restrict the delegation of rights in various ways.

2.4.3 Resource Layer: Sharing Single Resources

The Resource layer builds on the Connectivity layer communication and authentication protocols to define protocols (and APIs and SDKs) for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. Resource layer implementations of these protocols call Fabric layer functions to access and control local resources. Resource layer protocols are concerned entirely with individual resources and hence ignore issues of global state and atomic actions across distributed collections; such issues are the concern of the Collective layer discussed below. Two primary classes of Resource layer protocols can be distinguished:

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

- Management protocols are used to negotiate access to a shared resource, specifying, for example, resource requirements (including advanced reservation and quality of service) and the operation(s) to be performed, such as process creation, or data access. Since management protocols are responsible for instantiating sharing relationships, they must serve as a policy application point, ensuring that the requested protocol operations are consistent with the policy under which the resource is to be shared. Issues that must be considered include accounting and payment. A protocol may also support monitoring the status of an operation and controlling (like as terminating) the operation. While many such protocols can be imagined, the Resource (and Connectivity) protocol layers form the neck of our hourglass model, and as such should be limited to a small and focused set. These protocols must be chosen so as to capture the fundamental mechanisms of sharing across many different resource types e.g. different local resource management systems, while not overly constraining the types or performance of higher-level protocols that may be developed.

2.4.4 Collective Layer: Coordinating Multiple Resources

While the Resource layer is focused on interactions with a single resource, the next layer in the architecture contains protocols and services (and APIs and SDKs) that are not associated with any one specific resource but rather are global in nature and capture interactions across collections of resources. This layer of the architecture is known as the Collective layer. Because Collective components build on the narrow Resource and Connectivity layer 'neck' in the protocol hourglass, they can implement a wide variety of sharing behaviors without placing new requirements on the resources being shared. For instance:

Directory services allow VO participants to discover the existence and/or properties of VO resources. A directory service may allow its users to query for resources by name and/or by attributes such as type, availability, or load [33].

Co-allocation, scheduling, and brokering services allow VO participants to request the allocation of one or more resources for a specific purpose and the scheduling of tasks on the appropriate resources. Examples include AppLeS [13, 14], Condor-G [29], Nimrod-G [7], and the DRM broker [26].

Monitoring and diagnostics services support the monitoring of VO resources for failure, adversarial attack (intrusion detection), overload, and so forth.

- Data replication services support the management of VO storage (and perhaps also network and computing) resources to maximize data access performance with respect to metrics such as response time, reliability, and cost [2, 56].

- Grid-enabled programming systems enable familiar programming models to be used in Grid environments, using various Grid services to address resource discovery, security, resource allocation, and other concerns. Examples include Grid-enabled implementations of the Message Passing Interface [12, 25] and manager-worker frameworks [16, 31].

- Software discovery services discover and select the best software implementation and execution platform based on the parameters of the problem being solved [17]. Examples include NetSolve [18].

- Community authorization servers enforce community policies governing resource access, generating capabilities that community members can use to access community resources. These servers provide a global policy enforcement service by building on resource information, and resource management protocols (in the Resource layer) and security protocols in the Connectivity layer. Akenti [42] addresses some of these issues.

- Collective functions can be implemented as persistent services, with associated protocols, or as SDKs (with associated APIs) designed to be linked with applications. In both cases, their implementation can build on Resource layer (or other Collective layer) protocols and APIs. For example, figure 2.2 shows a Collective co-allocation API and SDK (the middle tier) that uses a Resource layer management protocol to manipulate underlying resources. Above this, there is a co-reservation service protocol and implement a co-reservation service that speaks this protocol, calling the co-allocation API to implement co-allocation operations and perhaps providing additional functionality, such as authorization, fault tolerance, and logging. An application might then use the co reservation service protocol to request end-to-end network reservations.

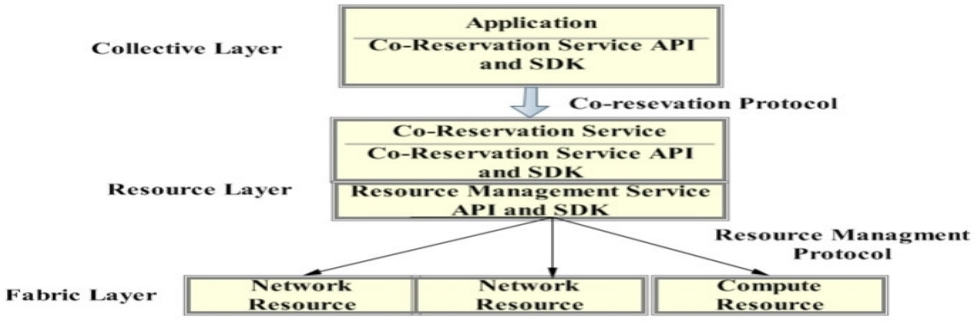


Figure 2.2: Collective and Resource layer protocols, services and API [23].

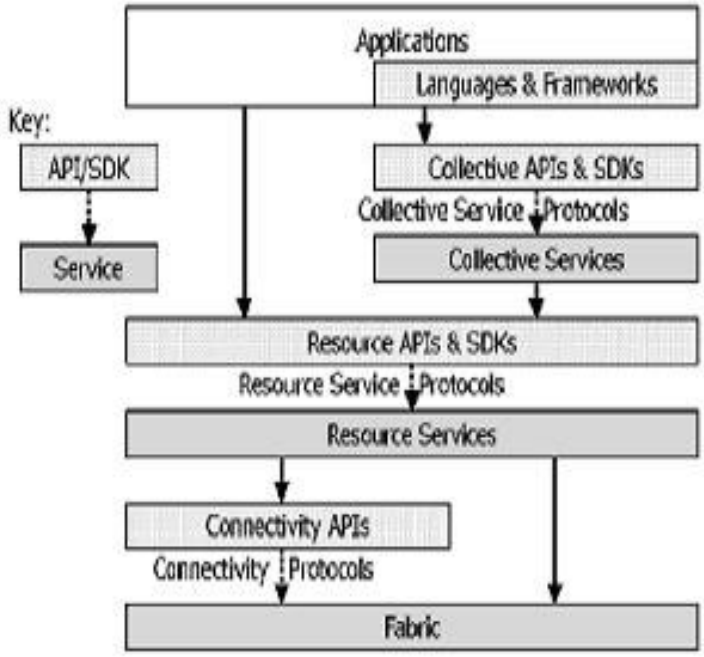


Figure 2.3: APIs implementation by Software Development Kits (SDKs)[23].

Collective components may be tailored to the requirements of a specific user community, VO, or application domain, for example, an SDK that implements an application-specific coherency protocol, or a co-reservation service for a specific set of network resources. Other Collective components can be more general-purpose, for example, a replication service that manages an international collection of storage systems for multiple communities, or a directory service designed to enable the discovery of VOs. In general, the larger the target user community, the more important it is that a Collective component's protocol(s) and API(s) be standards based.

2.4.5 Applications Layer

The final layer in Grid architecture comprises the user applications that operate within a VO environment. Figure 2.3 illustrates an application programmer's view of Grid architecture. Applications are constructed in terms of, and by calling upon, services defined at any layer. At each layer, there are well-defined protocols that provide access to some useful service: resource management, data access, resource discovery, and so forth. At each layer, APIs may also be defined whose implementation (ideally provided by

third-party SDKs) exchange protocol messages with the appropriate service(s) to perform desired actions. In the following section, we present an example about how Grid architecture functions cooperate in practice [23].

2.4.6 Grid Architecture in Practice

In figure 2.5 we present the services that might be used to implement the multidisciplinary simulation and cycle sharing (ray tracing) applications introduced in figure 2.4. The basic Fabric elements are the same in each case: computers, storage systems, and networks. Furthermore, each resource "speaks" standard Connectivity protocols for communication and security, and Resource protocols for enquiry, allocation, and management. Above this, each application uses a mix of generic and more application-specific Collective services.

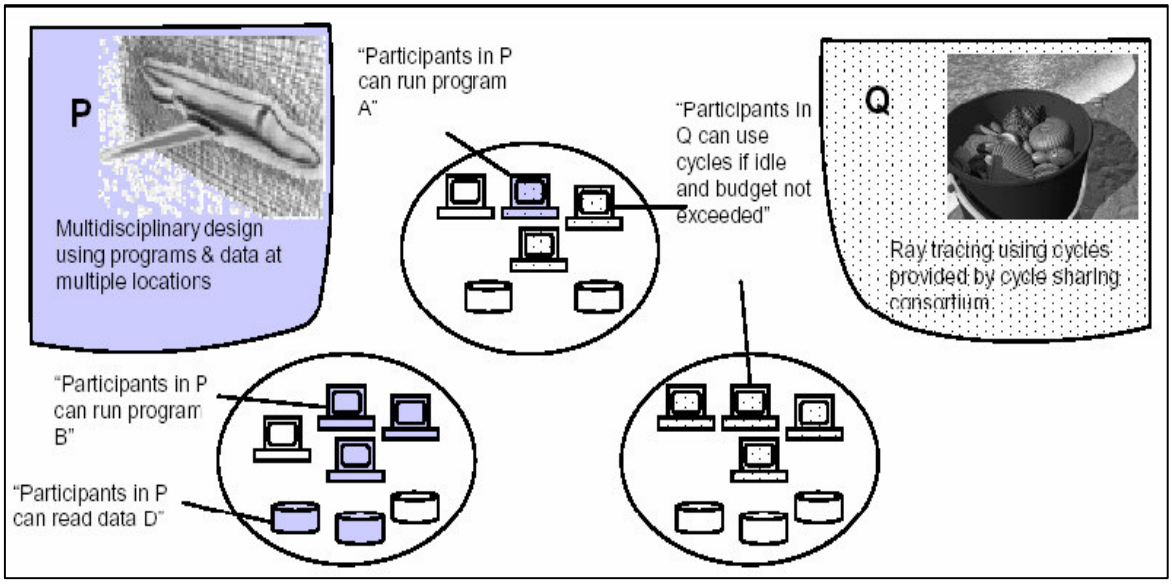


Figure 2.4: An organization participation in one or more VOs [23]

	Multidisciplinary Simulation	Ray Tracing
Collective Layer (application-specific)	Solver coupler, distributed data archive	Check-pointing, job management failover, staging
Collective Layer (generic)	Resource discovery, resource brokering, system monitoring, community authorization, certificate revocation	
Resource Layer	Access to computation; access to data; access to information about system structure, state, performance	
Connectivity Layer	Communication (IP), service discovery (DNS), authentication, authorization, delegation	
Fabric Layer	Storage systems, computers, networks, code repositories, catalogs	

Table 2.1: The Grid services used to construct two example applications of Figure 2.4.

In the case of the ray tracing application, we assume that this is based on a high throughput computing system [29, 43]. In order to manage the execution of large numbers of largely independent tasks in a VO environment, this system must keep track of the set of active and pending tasks, locate appropriate resources for each task, stage executables to those resources, detect and respond to various types of failure, and so forth. An implementation in the context of the Grid architecture uses both domain specific Collective services (dynamic checkpoint, task pool management, failover) and more generic Collective services (brokering, data replication for executables and common input files), as well as standard Resource and Connectivity protocols. Condor-G represents a first step towards this goal [29].

2.5 Relationships with Other Distributed System Technologies

2.5.1 WWW

The World Wide Web is a global information space. It works very well for information exchange, linking related information together but does not focus on the coordinated use of resources at multiple sites. In addition, the Web lacks several features that often required in Grid environment, such as single sign-on, delegation, or secure, reliable, and efficient large data movement.

2.5.2 P2P

P2P networks are often characterized by a large number of peers (millions) that collaborate equally, dynamically to share data (text, audio, video, etc). Data are often replicated in a P2P system. A common P2P model is decentralized systems where nodes join and leave in a dynamic and ad hoc manner and there is no central repository. Examples of this type are Freenet [59], P-Grid [62]. Decentralized P2P systems trade-off short response time and expensive servers for simplicity and scalability. Only a few P2P systems have central repository, for instance Napster [61]. The disadvantages of a centralized system are the problem of scalability and single point of failure. However, these systems have faster search times and guaranteed to find all results if exists. There are also some hybrid approaches in which there are central repositories for each small set of peers (Kazaa [60]). File sharing without access control and the freedom for a peer to act as client or server or both in P2P systems introduces security issues, for example giving wrong answer, distributing malicious code.

Contrarily, in a Grid, the sharing resources that we are concerned with are not only files but also computational resources, data storage, software, networks, and other resources. Furthermore, these resources are highly controlled by local policies.

2.5.3 Middleware

Middleware technologies such as CORBA, RMI, and DCOM provide remote invocation mechanisms that simplify the implementation of distributed applications by hiding the complexity network communication. However, they require centralize administration and

are often used within a single organization and homogenous environment. The form of interaction is client-server, rather than the coordination of multiple resources.

2.5.4 Parallel Computing

The thousands of processors in a Grid provide a significant computational power. However, this does not imply that traditional parallel high-performance computers are obsolete. Many problems require low latencies and high communication bandwidths. Such requirements are, up to date, still difficult to achieve in a Grid with heterogeneous hardwares and softwares components.

In summary, a Grid is a hardware and software infrastructure that expose to users as a single virtual computer. So in a Grid, there can be diverse type of hardwares (supercomputer, desktop PC, PDA, etc), diverse types of softwares (different OS, middlewares such as RMI, CORBA, Web services) but all of these complex, heterogeneous resources are expose to the users simply as Grid services, and they can access them simply by using a set of standard protocols and interfaces.

An important point that has to be emphasized here is about Grid computing resource management. Resource management is to run the jobs, monitor their status, and return outputs when jobs are done. In next chapter we see how resource allocation in Grid is done.

Chapter3. Resource Allocation in Grids

Resource Scheduling for Grid computing has received a lot of attention recently. Because of scheduling is NP complete, designing heuristic and evaluating their performance become the key issue. Static scheduling for a set of independent task is studied in [50], whereas the related dynamic scheduling problem is studied in [41].

The condor project [10] developed as a software infrastructure so that heterogeneous resources with distributed ownerships can be utilized to provide large amount of processing capacity over long period of time. The work presented in this thesis is also related to divisible load scheduling problem, where the load can be partitioned into tasks with arbitrary sizes. The divisible load-scheduling problem has been studied in [1] for systems with network topologies. Compared with these efforts, the proposed work studies a more general problem that allows an arbitrary network.

Distributed computing is an environment where you can harness idle CPU cycles and storage space of tens, hundreds, or thousands of networked systems to work together on a particularly processing-intensive problem. Due to the partly fact that a single parallel architecture may not be adequate for exploiting all of a program's available parallelism, there has been a recent increase of interest in heterogeneous computing system as in most of the cases heterogeneity has been shown to produce higher performance than a single large machine in a cost effective way. The major challenge for such heterogeneous and dynamic infrastructure is to develop an efficient and robust resource allocation and access mechanism, which is scalable, reliable and adaptable to changing circumstances.

This chapter is organized as follows: Section 3.1 defines the resource allocation and resource management. Section 3.2 reviews the structure and characteristic of heterogeneous systems and proclaims the need for heterogeneity. Section 3.3 discusses about resource management in Grid and section 3.4 surveys on currently available systems for such management. Section 3.5 summarizes the issues and resource

management problems for heterogeneous system. Section 3.6 summarizes currently resource allocation approaches and finally concludes this chapter.

3.1 Resource Allocation

Resource allocation is the process of mapping the resource request from jobs to the resources in a way that will satisfy both the application jobs and resource administrators. Resource management is the process of managing available resources and system workloads accordingly. Resource includes resources such as CPU cycles, storage systems, applications, memory, network bandwidth etc. Resources have their owners who may charge others for using resources. Resources can be shared or exclusive, they can be used for a period of time or may or may not be renewable. A job is a collection of tasks working together. Jobs are hierarchical entities, and may have recursive structure i.e. jobs can be composed of Sub jobs or tasks and sub jobs may themselves contain sub jobs. Resource management becomes more important in the case where some resources are idle and could be used to relieve overloaded nodes in the same network. There are basically two principal reasons to address resource management in distributed system. First, available system resources should be maximally (or optimally) used in order to balance the distribution of the tasks over the different computing nodes. Second, when events such as local congestion, a node failure or communication failure occurs, it is necessary to manage (re-) allocation jobs and resources accordingly.

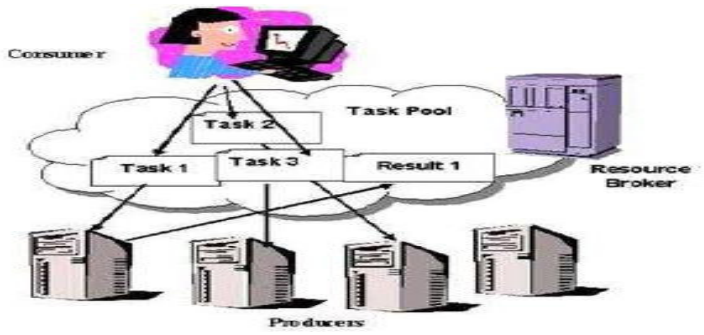


Figure 3.1: Resource Allocation in Grid [15]

3.2 Characteristics of Heterogeneous Networks

Computer network derived from computers using similar configuration and protocol is homogeneous network. Unlike, homogeneous network, computer network that combine one or more different types of computer or computer network and/or different types of protocol is called heterogeneous network. This type of network consists of dissimilar hardware or software devices ranging from simple PC or mobile devices to complicated supercomputer the simple and popular example of such network is world wide Internet which comprises dissimilar constituents of hardware and software interfaces.

Figure 3.2: Structure of Grid

Heterogeneous computing systems built up by closely connected networks of workstations or personal computers are a cheap alternative to dedicated parallel supercomputer systems. As these systems are widely available in academic and industrial environments it is becoming increasingly popular to use these resources to solve time consuming calculations. There has been a recent increase of interest in heterogeneous computing systems, due to party fact that a single parallel architecture may no be adequate for exploiting all of a program's available parallelism. In most of the cases heterogeneous systems have been shown to produce higher performance for lower cost than a single large machine or homogeneous networks. Homogeneous computing, which

uses one or more machines of the same type, has provided adequate performance for many applications in the past.

Many of these applications had more than one type of embedded parallelism such as single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD). On the other hand, as the amount of homogeneous parallelism in application decreases, homogeneous systems can't offer the desired speedups. To exploit the heterogeneity in computations with significant increase in speed and performance at affordable cost, researchers are investigating a suite of heterogeneous architecture.

3.2.1 Structure of Heterogeneous Network

The heterogeneous system includes heterogeneous machines, high speed networks, interfaces, operating systems, communication protocols and programming environments, all combining to produce a positive impact on ease of use and performance. In a heterogeneous computing environment, a suite of different machines is loosely interconnected to provide a variety of computational capabilities to execute collections of application tasks that have diverse requirements. The execution times of a task will vary from one machine to the next, and tasks will compete for machines in the suite. There are many types of heterogeneous systems, including parallel, distributed, clusters and Grids. An important research problem for heterogeneous computing is how to assign computation and communication resources to tasks and to schedule the order of their execution to maximize some performance criterion, a process known as mapping. Factors that must be considered include machine and network loading, how well the execution needs of a task match the computational capabilities of a machine, any inter-task communications, operating constraints and the performance criterion to be optimized.

3.2.2 Need for Heterogeneity

Large scale homogeneous distributed computing, which comprises machines of the same architecture and similar computational power, has been adequate for regular applications that spend majority of their execution performing on type of computation like sorting. Irregular applications, such as protein sequence matching and climate modeling, have a

spectrum of computational requirements like large-scale data acquisition, preprocessing, matrix computation, and scalar processing and data visualization. A heterogeneous distributed computing environment that comprises diverse machines designed for executing different types of computation is better suited for such applications. So for this case, an example of heterogeneous computing environment comprises a SIMD machine (for data acquisition and preprocessing), a vector machine (for highly parallel matrix computation), a general-purpose workstation (for scalar processing) and graphics machine (for visualization).

3.3 Resource Management in Grids

The overall aim of the resource management is to efficiently schedule applications that need to utilize the available resources in heterogeneous and dynamic environments. It if often the case in heterogeneous computing networks that an application requires multiple resources of different types to be allocated simultaneously. It has been shown that resource heterogeneity impacts the resource allocation in quite significant way in terms of performance, reliability, robustness, scalability and fault tolerance. While implementing complex systems in such heterogeneous and dynamic environments where resource usage and task availability are unpredictable, these challenges become more crucial. So, the system managing those complex environments needs to be smart and adaptable to change its allocation mechanism depending on the environment and its user requirements. The resource management mechanism should be able to manage the resources at two levels: Allocating system resources to a job as needed and separately managing the resources assigned to each job.

To design resource allocation and control mechanism in Grid environment the following goals need to be considered:

- Partitioned large complex allocation problems into smaller, disjoint allocation problems.
- Decentralized resource access, allocation and control mechanism.
- Design reliable, fault tolerant and robust allocation mechanism.
- Design scalable architecture for resource access in a complex system

- Provide guarantees to users and applications on performance criteria. Some of the performance criteria in distributed systems include the following:
    - Average response time
    - Throughput
    - User access time
    - Message delay time
    - Information loss
    - Application failure probability
- Define system performance criteria that reflect in aggregate the diverse individual criteria of users and applications
- Designs unified frameworks in which users have transparent access to the services of a distributed system, services are provided in as efficient manner. This framework should hide the complexity or multiple resource suppliers and resource allocation policies.

The resource allocation strategies should be smart enough to provide: application efficiency to measure how effective is the allocation in terms of application computation and resource efficiency to measure how well it manages its resources without wasting them and how thrift is the management policy?

In general, heterogeneous computing systems have computing resources with different capabilities, input/output devices, data repositories, and other resources all interconnected by heterogeneous local and wide area networks. The major challenges in using heterogeneous system are to effectively use all the available resources. Grids are becoming attractive and promising platform for solving large-scale (problem solving) application of multi-institutional interest with heterogeneous resources. However, the management of resources and scheduling computations (task and resources) in the Grid environment is a complex undertaking as they are geographically distributed, heterogeneous in nature, owned by different individuals or organizations with their own policies, different access and cost models and have dynamically varying loads and availability. This introduces a number of challenging issues such as site autonomy, heterogeneous substrate, policy extendibility, resource allocation and co-allocation,

online control, scalability, transparency and so on. Some of these issues are being addressed by a number of(on going) Grid computing projects world wide such as condor, Legion, Globus, Nimrod-G etc. These systems are discussed in detail in section 3.4. The Grid approach provides the ability to access, utilize and manage a variety of heterogeneous resource in virtual organizations across multiple domains and institution. Selecting appropriate resource within such a distributed Grid environment to satisfy quality of service requirement is a complex and difficult task. Resource allocation and management problem includes:

- Matching problem: the problem of assigning application tasks to suitable resources scheduling problem: ordering tasks executions in time to optimize a specific objective function.
- Co-allocation problem: can be defined as the problem of simultaneously allocating multiple resources of different types of application in order to meet specific performance requirements.
- Load balancing problem: to increase the utilization of the resources and task in the network.

Another resource management issue made complex due to the site autonomy is co-allocation of the resources. Co-allocation is the problem of allocating resources in different sites to an application simultaneously. Different domains employ different local resource managements systems like LSF etc. A Grid resource management system should be able to interface and inter-operate with these local resource managements. A resource management system for heterogeneous system should support such negotiation. In such system resources are added and removed dynamically. Different types of applications with different resource requirements are executed. Resource owners set there own resource usage policies and costs. This necessitates a need for negotiation between re- source users and resource providers. In Grid different types of application 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 framework should allow new policies to be incorporated into it without requiring substantial changes to the existing code.

In resource management of a Grid environment the following points are the basic requirements:

- Members should be trustful and trustworthy
- Sharing is conditional
- Should be secure
- Sharing should be able to change dynamically
- Discovery and registering of resources
- Can be peer to peer or client/server
- Same resource may be used in different ways
- Well defined architecture and protocols

3.4 Survey on Grid Resources Management Systems

The resource management system is the central component of Grid computing systems. Depending on the focus and application target, the Grid Resource Management Systems are broadly classified into Computational Grids, Data Grids and Service Grids. [35], a taxonomy for Grid resource management system is developed which classifies resource management systems by characterizing different attributes such as service focus of Grid, machine organization (flat, hierarchical) , resource model(schema, object) , scheduler organization(centralized, hierarchical, decentralized) ,scheduling policy(system centric, user centric), resource discovery(query, agents) etc. Previous approaches for resource management can be listed as:

- Condor [11] is a resource management system designed to support high-throughput computations by discovering idle resources on a network and allocating those resources to application tasks. Condor has a centralized scheduling model. Each condor work station submits the jobs in its local queue to the central scheduler which is responsible for finding suitable resources for the job execution. Condor uses the matchmaker/entity structure which can be both provider and requestor, in which the matchmaker becomes the bottleneck of the

system. Achieving scalability is difficult and focuses more on protocol issues than on performance issues.

- Nimrod-G [47] is a Grid resource broker for managing and steering task farming applications such as parameter studies on computational Grids. In follows a computational market-based model for resource management. Nimrod/G is being used as a scheduling component in a new framework called Grid Architecture for Computational economy (GRACE)[11] which is based on using economic theories for a Grid resource management system. It uses the service of other systems such as Globus and Legion for resource discovery and dissemination.

- Globus [55] is a research and development project focuses on enabling the application of Grid concepts to scientific engineering computing. It provides a basic infrastructure platform that integrates geographically distributed resources. The goal of Globus is to understand application requirements for a usable Grid and to develop the essential technologies required to meet these requirements. Globus is designed based on a layered architecture and intends to provide only the infrastructure and mechanism that consists of command line utilities and application programming interfaces (APIs).

Resource management of Globus consists of resources like resource brokers, resource co-allocator and resource manager or GRAM. The resource requests are specified in extensible resource specification language (RSL). Globus has a decentralized scheduling model. Scheduling is done by application translate the application requirements into more specific resource specification.

3.5 Resource Allocation Challenges

Traditional resource management systems work under the assumption that they have complete control on the resource and thus can implement the mechanism and policies needed for effective use of that resource. But in Grid, resources are distributed across separate administrative domains this results in resource heterogeneity, differences in usage, scheduling policies, security mechanisms. The grand challenge in such system is

to adaptively allocating resources to dynamic computations acting on behalf of millions of users exploiting even more of heterogeneous devices. It has been shown that resource heterogeneity impact the resource allocation in quite significant way in terms of performance, reliability, robustness and scalability. The system robustness increases as system complexity increases. System complexity is the function of system resources and available computing tasks. The resource allocation challenges in general can be listed as follows:

3.5.1 Scalability

In large distributed and heterogeneous systems, the issue of scalability is one of the important challenges to be addressed while developing resources allocation schemes. Scalability requires that an increase in the number of new agents and resources in the system has no noticeable influence on performance and on administrative tasks. A heterogeneous network like Grid has the potential to encompass a large number of high performance computing resources operating together. Each constituents of such system has its own function, its own resources and environment. These components are not necessarily fashioned to work together in the overall system. They may be physically located in different organization and may not be aware of each others capabilities. The resource management framework for such system should not degrade its speed, performance and efficiency significantly (or at least degrade gracefully) when the number of processors becomes large or when the requests for placement become more frequent. Currently available resource management for Grid systems are based on the traditional client-server paradigm or centralized system like condor [11], Legion[53]which may not be able to scale to millions of nodes as this systems have bottle neck associated with central controller. One approach to manage this challenge is to decentralize the system as system increases with more increase in computing infrastructure. A way to implement decentralized system in such environment is to adopt peer-to-peer models in order to improve scalability and reliability. An agent based hierarchical model [28] can be used to address the problem of scalability in the Grids. In agent based hierarchy, intelligent agents form hierarchical structure and each agent can only have one connection to an

agent higher in the hierarchy to register with but be registered with many lower level agents.

3.5.2 Adaptability

A heterogeneous infrastructure such as Grid is dynamic environment where the location, type and performance of the components are constantly changing. For example, a component resource may be added to or removed from Grid at any time. These resources may not be entirely dedicated to such environment and therefore the computational capabilities of the system will vary over time. The existing architecture of resource allocation for Grids features virtually no mechanism for automatic adaptation of the system or its parts to its parts to new internally or externally imposed conditions. To develop adaptive Grid system it is necessary to provide mechanism for automated adaptation and reconfiguration of the network on all hierarchy levels. Such mechanism include monitoring of the state of the computing nodes of heterogeneous network, its analysis together with decision taking, intelligent decision execution and finally delegation of control to human operators. In such environment, availability of resources may fluctuate which may result from connection or disconnection of computing resources, human interaction or interruption of the computers etc. An approach to adapt to the change in system state is to enable an adaptive negotiation between intelligent agents [63]. The negotiation between agents as also used to optimize computing resources allocation to computation jobs over time. Resource agents are used to manage all the applications resources, job agents find candidate resource agents and select suitable negotiation protocol based on specific needs.

3.5.3 Fault Tolerance and Reliability

As a result of the complex nature of heterogeneous networks, this is a major concern for the network administrators, and there are various ways that detection of such occurrences can be accomplished. When a fault occurs, it is important to, as rapidly as possible: Determine exactly where the fault is, Isolate the rest of the network from the failure so that it can continue to function without interference, Reconfigure or modify the network

in such a way as to minimize the impact of operation without the failed component or components, repair or replace the failed components to restore the network to its initial state. As Grid is a distributed and unreliable system involving heterogeneous resources located in different geographical domain, for this case fault tolerant resource allocation services have to be provided. In, particular when crashes occur, tasks have to be reallocated quickly and automatically, in a completely transparent way from the user's point of view. A Grid is a distributed system involving heterogeneous resources located in different geographical domains that are potentially managed by different organizations. Therefore, most of the difficulties encountered when designing Grid software are related to well known problems in distributed computing. One of the main challenges for Grid computing is the ability to tolerate failure and recover from them (ideally in a transparent way). Current Grid middleware still lacks mature fault tolerant features and next generation Grids needs to solve this problem providing a more dependable infrastructure to execute large-scale computations by using remote clusters and high performance computing system.

3.5.4 Load Balancing

When the demand for computing power increases the load balancing problem becomes important. We can state the load balancing problem as follows: given the initial job arrival rates at each computer in the system find an allocation of jobs among the computers so that the response time of the entire system over all jobs is minimized. In general we can distinguish static and dynamic load balancing algorithms. In the case of a static load balancing policy, a fixed process graph, which represents the distributed computation, mapped onto the interconnection network. In this case the aim is to minimize edge dilation, processor load difference and edge congestion. If the load situation changes in an unpredictable way, as it is the case for many applications, it is necessary to use a dynamic load balancing strategy which is adaptive to this changing load situation. To be efficient for large distributed systems the load balancing algorithm itself should be distributed. Any dynamic distributed load balancing strategy can be separated into a decision part and a migration part. In the decision part of the algorithm a decision can be based on the local load situation and that of the neighboring processes or

it depends on the load situation of any subset of the whole network. In the first case it is called a 'local decision base' whereas the second case is called 'global decision base'[19]. The load balancing problem is to design algorithms that minimizes the mean job waiting time by migrating the jobs to balance the workloads of the processor nodes. This can also be done using the bidding and auctioning mechanism, and price controls. In the load balancing economy, jobs migrate in search of suppliers based on the job preference model. Each job independently computes the best place (node) to be served based on its preferences and wealth and the resources prices. In microeconomics-based algorithm for load balancing [34], the system is partitioned into sets of completely independent agents that compete for the resources in the system. Through the laws of supply and demand, this competition dynamically sets prices that are charged for purchasing the available resources. A globally effective allocation of the resources is obtained indirectly through the selfish optimization and competitive behavior of the economic agents. In this approach a job attempts to purchase resources and be serviced. To do, it computes budget set based on price and service time and then generate a bid for the most preferred budget set element. A job is allowed to migrate through the system in search of CPU service, but it must pay each time it crosses a link. Job must return to the processor at which it entered the system after receiving CPU service.

3.6 Grid Resource Allocation Approaches

There are various approaches for allocating resources in heterogeneous and distributed networks. Some of these approaches are listed as follows:

3.6.1 Matchmaking and Brokering Approach

The underlying idea of matchmaking and brokering approach is relatively simple. Servers and customers advertise their presence to a common advertising service by describing their characteristics in advertisements. These advertisements also contain qualitative descriptions of the entities the agents would like to be matched with. A matchmaker discovers compatible providers and customers with a generic matching operation and notifies the matched agents, which then employ a protocol to connect to each other and

enable exchange of service. In this kind of resource allocation approach, there are three different kinds of agent categories involved: service providers, service requester and middle agents. Match making is the process of finding an appropriate provider for the requester through a middle agent. Provider agents advertise their resources to middle agents. Middle agents store these advertisements. A requester asks some middle agent whether it knows of providers with desired resources.

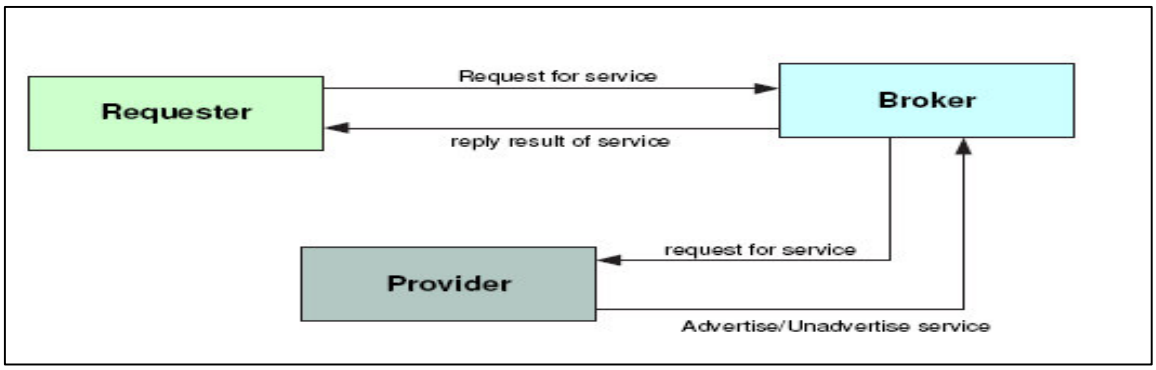


Figure 3.3: Matchmaking Process [9]

The middle agent matches the request against the stored advertisements and returns the result. While this process at first glance seems very simple, it is complicated by the fact that provides and requesters are heterogeneous and incapable in general of understanding each other.

Matchmaking within multiple agents through broker or middle agent is the common and popular approach of resource allocation [9, 34]. One of the major problems facing multi agent systems is finding the services and information that the agent needs and connecting to the agent that is providing this service. There are different types of middle agents such as: matchmaker, broker, billboard etc.

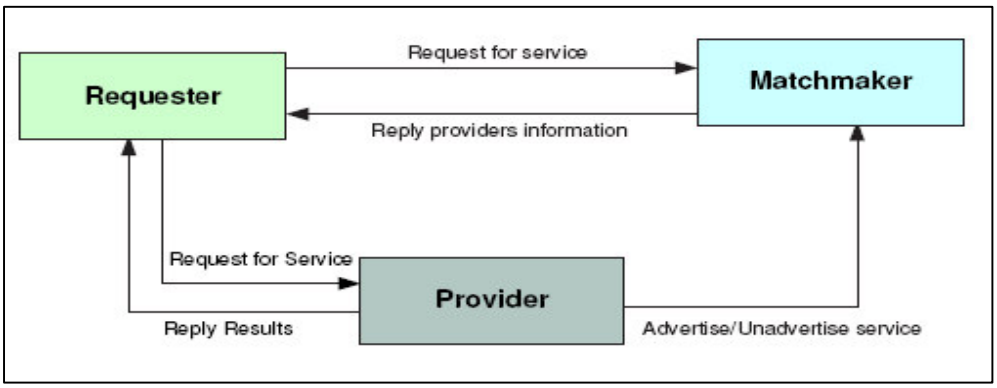


Figure 3.4: Brokering Process [9]

In brokering approach, requester request for the service to the broker and broker provides that service on behalf of the provider. From requester point of view, a broker is indistinguishable from a server. In contrast to broker agent, in matchmaking, a matchmaker doesn't deal with the task of contacting the relevant providers, transmitting the service requester. This avoids data transmission bottleneck, but it might increase the amount of interaction among agents. Figure 3.3 and figure 3.4 show matchmaking and brokering process.

3.6.2 Market Based Approach

In the real world market, there exist various economic models for setting the price of services based on supply-and-demand and their value to users. These real world economy models such as commodity market model, market bidding, auction model, bargaining model etc can also be applied to allocate resources and tasks in distributed computing [48]. In commodity market model resource owners or producers specify their service price and charge users according to the amount of resource consume. Each buyer and seller will set price and quantity of how much it is willing to buy or sell. Consumer pays for their access to various resources they use like CPU cycles, memory, network bandwidth etc. In real world, auctions are used extensively for selling goods/items within a set duration. The most common auction models are one-to-many and many-to-many auction models. Service providers announce their services and invite bids. Different consumers offer their bids for the auction. This process continues until no one is willing to bid higher price or auctioneer stops. Finally, service provider offers its service to one who wins and the consumer uses the service. There are several auctioning mechanism such as the sealed bid auction, Dutch auction and English auction. As distributed systems grow in size and the nodes become more powerful and complex, the complexity of making resource allocation decisions grows dramatically. Microeconomic approaches limit this complexity by portioning the complex global problem of allocating many resources, supplied by many sources and by many types of consumers into a set of simple, independent sub-problems. In Grid economy, decentralization is provided by the fact that economic model consists of agents which selfishly attempt to achieve their goals. Paper [27] provided decentralized algorithms to allocate resource in a co-operative

and non-competitive manner among included communication and average processing delay. Resource consumers adopt the strategy of solving their problems at low cost within a required time frame and resource providers adopt the strategy of obtaining best possible return of their investment. The load-balancing problem can also be solved using the bidding and auctioning mechanism and price controls. In the load balancing economy, jobs migrate in search of suppliers based on job preference model. Each job independently computes the best place to be served based on it preferences and wealth, and he resources prices. The paper [8] purposed a microeconomic algorithm for load balancing in distributed computer systems. In this approach system is partitioned into sets of completely independent agents that compete for the resources in the system. Through the laws of supply and demand, this competition dynamically sets prices that are charged for purchasing the available resources.

3.6.3 Peer to Peer Resource Allocation

In contrast to the client/server architecture, there is no single central system (server) in a P2P network that provides all data and services to the consumers (clients). Information is directly exchanged between providers and consumers. Most of these systems align their peers in a overlay network to existing network interfaces. P2P network is a distributed network composed of a large number of distributed, heterogeneous, autonomous and high dynamic peers in which participants share a part of their own resources such as processing power, storage capacity, software and files content. These kinds of networks are managed without any centralized control or hierarchical organization. Each participating node in such system is functionally equivalent and can act as server and a client at the same time. A vast number of p2p based routing protocol and platforms have been developed over the past few years eg. Gnutella [57], Pastry, Chord, CAN and JXTA [58]. Peers are accessible by other node directly without passing intermediary entities. Decentralized and distributed nature of P2P systems makes them robust against certain kinds of faults, making them potentially well suited for long term storage or lengthy computation. Peer-to-peer systems offer a number of advantages over conventional client server or centralized systems such as scalability, fault tolerance, performance. In a P2P network end-users share resources via direct exchange between computers and

information is distributed among the member's nodes instead of centralized at a single server. Different approaches have been used to provide self-organizing behavior of the peers and thus relieve the users of configuring the rapidly changing network manually. A Peer to peer communication uses local neighborhood [38, 46]. In localized matchmaking providers and consumers look only for an acceptable match within their local neighborhood. However, this might not yield with the best possible match. In this case, computing nodes are connected by random peer-to-peer communication mechanism which search partners for tasks by forming small group that locally modify their network connections. As agents find compatible neighbors to form cluster and gradually get larger.

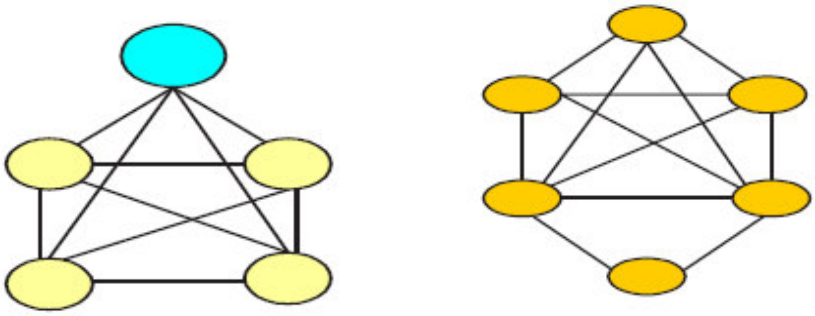


Figure 3.5: Pure P2P and Hybrid P2P [38]

A successful trail of the system is one that ends up with most of the agents connected to each other, usually in a single large cluster. So in this model cluster size is not limited and thus the design goal of avoiding centralized control is fully met

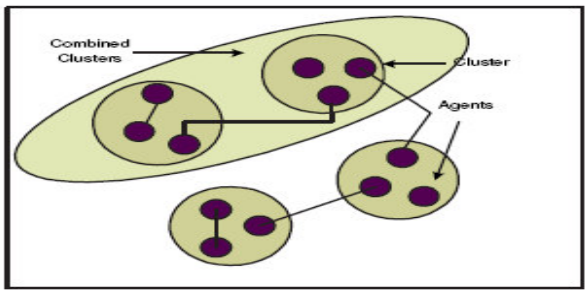


Figure 3.6: Localized Matchmaking using peer-to-peer [38]

The overall aim of the resource management is to efficiently schedule applications that need to utilize the available resources in heterogeneous and dynamic environments. These resource management scenarios often include resource discovery, resource inventories, fault isolation, resource provisioning, resource monitoring, a variety of autonomic capabilities, and service-level management activities. The resource allocation in Grid posses various challenges like scalability, adaptability, flexibility and load balancing. The most interesting aspect of the resource management area is the selection of the correct resource from the Grid resource pool, based on the service-level requirements, and then to efficiently provision them to facilitate user needs. In this chapter, we examined resource allocation problems studied resource management in Grid and related issues and challenges and surveyed on various resource management approaches available for Grids. The next chapter of the thesis will discuss an efficient resource management system for Grid to address the above-mentioned issues and challenges.

Chapter 4. Problem Formulation

As seen in previous chapters the resource allocation in Grid posses various challenges like scalability, adaptability, flexibility and load balancing. Existing methods of resource management in Grid and related issues and challenges have been studied and reviewed.
For Grid Resource Allocation various Approaches like Matchmaking and Brokering Approach [9, 34], Market based Approach [48], Peer to Peer Allocation Approach [38,46] exist. The problem with Matchmaking Approach is finding the services and information that the agent needs and connecting to the agent that is providing this service.

These approaches do not focus much on performance optimization. Therefore, this work here focuses on performance and cost optimization during resource allocation using linear programming.

This thesis aims is to design and develop a performance optimization in Grid resource allocation, which overcomes the shortcomings of the current state of the art in the context. It also assumes that numbers of resources are finite in the Grid.

Main Objectives are

- Allocation of best resource using linear formulation discovered by resource discovery approaches.
- The allocation approach fine-tunes the cost and performance optimization tradeoff.
- Use of Simplex Approach [20] of linear programming problem to check complexity bounds.

Chapter 5. Proposed Linear Programming Based Approach

Many problems can be formulated as the problem of maximizing or minimizing an objective function, given limited resources and competing constraints. If we can specify the objective as a linear function of certain variables, and if we can specify constraints on resources as equalities or inequalities on those variables, then we can have Linear Programming.

In this chapter, we consider resource allocation problem in Grid computing environment. We consider a general problem in which interconnection between the nodes is modeled using a network diagram. Initially, the source data for all the tasks resides on the root node; other nodes in the system receive tasks from the root and some of their neighbors, compute some of these tasks, and push some tasks to some selected neighbors. We maximize the performance of the system by using a linear programming formulation. We transform graph into linear programming formulation, which can be further converted into network diagram. We can traverse this network diagram efficiently using Bellman-Ford algorithm[53].

The problem of computing a large set of variable sized independent tasks on a Grid-computing environment is considered here. The system consists of a collection of heterogeneous compute resources, connected via network links. We model the system as an undirected graph, where each node in the graph represents a compute resource and each edge in the graph represents a network link. A compute node needs to receive the source data for a task before executing the task. We assume that the source data for all the tasks initially reside on a single node in the system, which we call the root node. A compute node in the system can communicate with not only the root node, but also its neighbors. Every compute node thus needs to determine: from where the tasks come from and there are how many tasks to compute locally, and where to transfer the rest of the tasks that it has received. We denote it as a graph structured computing paradigm. To minimize the overall execution time of all tasks is a NP-complete problem [45].

5.1 System Model

Given system is represented by a weighted undirected graph G (V, E), where V represents a set of vertices and E represents a set of edges. The number of vertices is given by n=|V| and the number of edges are given by m =|E|.

Each node $V_i \epsilon V$ in the graph represents a compute resource. The weight of V_i is denoted by w_i, w_i represents the processing power of node V_i, that is, V_i can perform one unit of computation in $1/w_i$ time. Each edge $E_{ij} \epsilon E$ in the graph represents a network link. The weight of E_{ij} is denoted by ew_{ij}. ew_{ij} represents the communication link E_{ij} ,that is , link E_{ij} can transfer one unit of data from V_i to V_j in $1/ew_{ij}$ time. Links are bidirectional so G is undirected and $E_{ij} = E_{ji}$. This graph model is denoted as base model.
Now we express this problem as a linear program, for this we determine a set of variables and constrain that define shortest path from s to t.

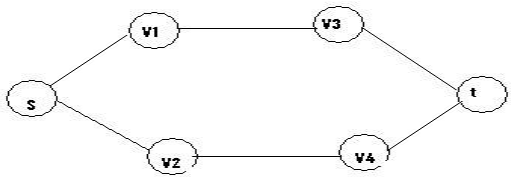


Figure 5.1: Graph Representation G (V, E) [54]

Compute node can send and receive data concurrently to its neighbor nodes. For each compute node V_i, we introduce two parameter I_i and O_i. These two parameters indicate the capacity of receive and send data with one unit of time.

System performance is defined as the number of tasks processed by the system in one unit of time. Let f (V_i, V_j) denote the number of tasks transferred from V_i to V_j during this time interval. Note that f (V_i,V_j) is directional, although the corresponding link E_{ij} is not. If actual data transfer is from V_i to V_j, we define f (V_j,V_i)= - f(V_i,V_j)

Base Problem: Given an undirected graph G (V, E), where node V_i has weight w_i and associated parameters I_i and O_i , and edge E_{ij} has weight ew_{ij}, where

$ew_{ij} > 0$ if $E_{ij} \epsilon E$

And $ew_{ij} = 0$ otherwise.

In the linear program, x_v represents the amount which a vertex v has been picked and y_{ev} represents how much an edge e has been assigned to one of it's vertices v.

We have following constraints for resource selection:

1. $x_v \geq y_{ev}$ $\quad \forall v \in e \in E$

This means that Job select is more than that of Job assigned.

2. $y_{ev} \geq 0$ $\quad \forall v \in e \in E$

Job assigned to any node is always greater than or equal to zero.

3. $0 \leq x_v \leq 1$ $\quad \forall v \in V$

We have following constraints for resource allocation:

1.$| f(V_i, V_j) / ew_{ij} | <= T$ for $\forall\ E_{ij} \in E$. This is because E_{ij} can transfer ew_{ij} unit of data in one unit of time.

2. $\Sigma\ V_k \in A_i\ [f(V_k,V_i)] >= 0$ for $\forall\ V_i \in V - \{ V_0\}$. This condition says that no intermediate node can generate tasks.

3. $\Sigma\ V_k \in A_i$ & $f(V_i,V_k) >0\ [f(V_i,V_k] <= T * O_i$,for $\forall\ V_i \in V$. This means that V_i cannot send data at a rate higher than what is allowed by its network interface.

4. $\Sigma\ V_k \in A_i$ & $f(V_k,V_i) >0\ [f(V_k,V_i] <= T * I_i$,for $\forall\ V_i \in V - \{V_0\}$. This means that V_i can not receive data at a higher than what is allowed by its network interface.

5. $\Sigma\ V_k \in A_i\ [f(V_k,V_i)/ w_i] <= T$ for $\forall\ V_i \in V - \{V_0\}$.We can see that $\Sigma\ V_k \in A_i\ [f(V_k,V_i)]$ is the total number tasks that V_i has kept locally (tasks received minus tasks sent out). This condition says that no intermediate node should kept more tasks than it can compute.

Objective Function

Maximize $P = p_0 + \Sigma\ (\Sigma_{V_k \epsilon V} f(V_k,V_i))$----------(1)

$V_i \epsilon V - \{V_0\}$

And $\Sigma\ x_v$ such that, $v \in V$

The first step of the algorithm is to look at all capacity one vertices and see if they are required for the solution, this is tested by creating a special form of network diagram problem in which a bipartite graph is formed in which one set of vertices represents the edges of the original graph and the other set represents the vertices of the original graph. Edges are placed connecting the vertices representing edges to the vertices representing their endpoints with capacity for 1 unit of flow. The source is connected to all of the vertices representing edges with edges that can support 1 unit of flow, and finally the sink is connected to each vertex representing a vertex from the original graph with the same capacity as the vertex had in the original node. If at this point the maximum flow through the newly constructed flow network is greater than or equal to |E| there is a feasible solution.

In this way each capacity one vertex can be "removed" by setting it's capacity to 0 and then the graph can be tested for feasibility. If the graph has a feasible solution that vertex's capacity is left at 0. The capacity one vertices, which are required for feasibility, are automatically picked by setting the corresponding x_v variable to 1. Once this preprocessing is done the LP is constructed and solved. At this point the vertexes whose corresponding x_v variable are greater than or equal to ½ are included in the cover automatically. This does not guarantee a solution, so the remaining vertices are included with probability equal to $2x_v$, for the corresponding x_v variable. This still does not guarantee a solution, so in a relatively complicated manner a few more vertices are selected for the cover. These vertices are paid for out of the excess that was paid in order to include all of the vertices with x_v larger than ½, or $\Sigma\ v \in V\ (2\ x_v - 1)$. At this point we have a feasible solution in which the vertices are integrally picked and assigned to the cover or not, but the edges may be fractionally assigned to vertices.

Finally the same flow network used in the feasibility checking is constructed and the edges are integrally assigned to vertices using the integrality of flows property which guarantees a solution in which the constraints are still satisfied, but that each edge carries an integral amount of flow.

5.2 Resource Allocations to Maximize System Performance

From equation 1 we can shows that p_0 is an additive constant to the system throughput; hence, we can ignore p_0 and maximize $\Sigma\ V_i \in V - \{V_0\}\ (\ \Sigma\ V_k \in V\ f\ (V_k,V_i))$ .System performance is maximized when V_0 perform tasks at the highest rate possible. Formally we have the following proposition.

$\Sigma\ V_i \in V -\{V_0\}\ (\ \Sigma\ V_k \in V\ f(V_k,V_i)) = \Sigma\ V_k \in V\ f\ (V_0,V_k)$

5.2.1 A Linear Programming Formulation:

We first transform the base model to include the constraints enforced by I_i and O_i .The transformation is performed using the following procedure

Procedure 1:

1. In place of V_i put nodes V_i^1, V_i^{11} .The weights of these new nodes are w_1, w_2, w_3 and w_4 respectively. Add directed edge of weight I_i for input node. And another directed edge of weight O_i for outgoing edge.

2. Assigns weight to each between two nodes. Weights for these edges are $ew_{ij.}$ Where $(i,j) \in 1\ldots 4$.

d[s] = 0.

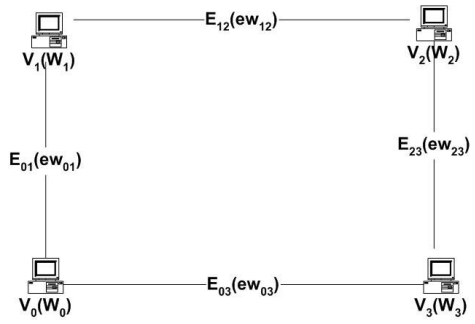


Figure: 5.2: The base model of a sample system

Linear programming formulation is converted into network diagram and then Bellman - Ford method is applied. It helps in finding shortest path. When this algorithms terminates,

it has computed, for each vertex V, a value of d[V] such that for each edge (u,v) ϵ E, we have d[v] <= d[u] +f(u,v).The source vertex initially receives a value d[s] =0 , which is never changed. Thus we obtain the following linear program to compute the shortest weight from S to T.

Objective: Maximize d[t]

Subject to: d[v] <= d[u] + ώ (u, v) for each edge (u, v) € E,

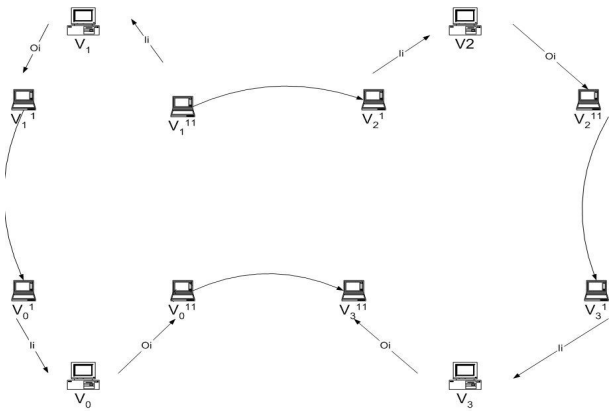


Figure 5.3: The Base model and its intermediate representation

In this linear program, there are |V | (n) variables d[v], one for each vertex v ϵ V. There are |E|+ 1(m+1) constraint, one for each edge plus the additional constraint that the source vertex always has the value 0.

We start with f (u, v) =0 for all u, v ϵ V, giving an initial flow of value 0. At each iteration, we increase the flow value 0. At each iteration, we increase the flow value by finding an augmenting path, which we can think of simply as a path from the source s to the sink t along which we can send more flow, and then augmenting the flow among this path. We repeat this process until no augmenting path can be found.

5.2.2 A Network Diagram Representation:

From equation 1 we notice that the system performance is the sum of V_0's compute power and the rate with which tasks flow out of V_0 After the data (tasks) flow out of V_0, it will be transferred in the system and finally be consumed by some nodes. If we model

these data consumptions as a special type of data flow to generator nodes, then the performance of the system is exclusively defined by the rate with which data flow out of V_0. This leads to our network representation for the system performance problem. The following procedure transforms the intermediate representation top network representation. [51]

Procedure 2:

1. Now, create a corresponding intermediate node V_i^1 in the Network Diagram representation. Set the weight of V_i^1 as w_1.

2. For each edge E_{ij} in the intermediate representation that goes from V_i to V_j, create an edge E_{ij}^1 in the representation that goes from V_i^1 to V_j^1. Set the weight of E_{ij}^1 as that of E_{ij}.

3. Add a node S to the network diagram .S has weight 0.

4. For each node V_i^1 in the network representation, if the weight of V_i(v_i^1 's corresponding node in the intermediate representation) is greater than 0, add an edge E_{ij}^1 that goes from V_i^1 to S. Set the weight of E_{ij}^1 as the weight of node V_i , in the intermediate representation.

We call the new node S sink node of the Network representation.

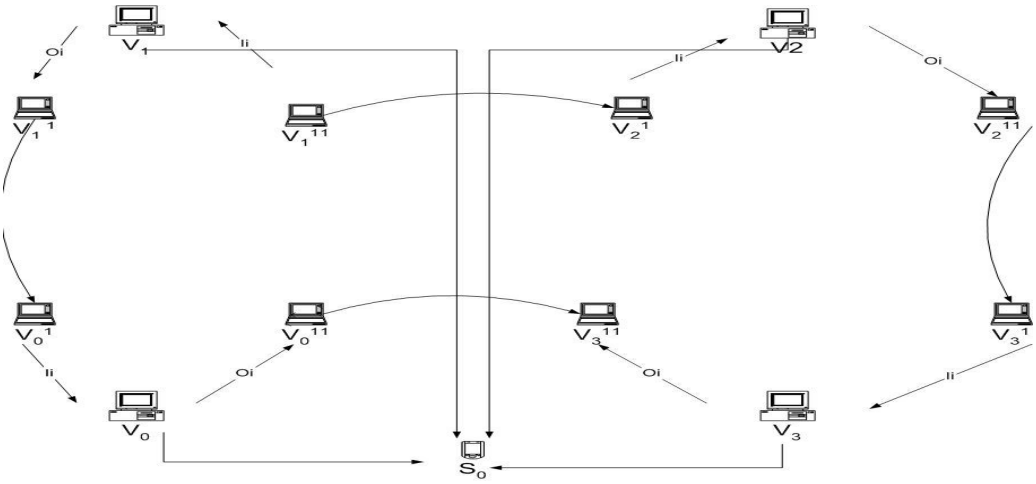


Figure 5. 4: The Network representation of the system in Figure 5. 3

Above figure shows an example of network diagram when applying procedure 2 to intermediate representation. Node S is the newly added sink node. The weight of the nodes is marked in the parenthesis after the node names. The edge names are omitted; only the edge weights are marked in the parentheses.

5.2.3 Solving Problem with Simplex Method

To solve a linear formulation we here use simplex method. Steps of simplex methods are as:

Step 1. Convert the linear program into standard form. This means making the linear program the minimizing variety and changing inequality constraints to equality constraints.

Step 2. Find a first basic feasible solution. (Iterate through these steps with a slightly expanded form of the problem.)

Step 3. Calculate the Reduced costs.

Step 4. Test for optimality.

Step 5. Choose the entering variable.

Step 6. Calculate the Search Direction.

Step 7. Test for unboundedness.

Step 8. Choose the leaving variable by the Min Ratio Test.

Step 9. Update the solution.

Step 10. Change the basis.

For finding shortest path in the network Bellman Algorithm is used with complexity O (|V|.|E|).

Bellman (G, w, s)

Instruction 1. Initializes single source (G,s)

Instruction 2. **for** i ← 1 to |V[G] | -1

Instruction 3. **do for** each edge (u,v) ϵ E{G}

Instruction 4. **do** RELAX (u,v, w)

Instruction 5. **for** each edge (u, v) ϵ E (G)

Instruction 6. **do if** d[v] > d[u] + w (u,v)

Instruction 7. **then return** FALSE

Instruction 8. **return** TRUE

In this section we analyze the performance of the presented approach in the context of computational costs. Our Algorithms runs in time Ω(VE), since the initialization in line 1 takes Θ (V) time, each of the |V| -1 passes over the edges in line 2-4 takes Θ(E) time, and the for loop of line 5 -7 takes Ω(E) time.

Chapter 6. Implementation Details and Experimental Setup

In order to implement our proposed resource allocation model, we developed an application, which simulates on the Grid environment i.e. on Grid portal. We developed the application using J2EE, Java Applets and MySQL database server.

6.1 Technologies Used

In this section we give a brief introduction about the technologies we used in developing our project. We used J2EE to program the client side architecture and used applets to program the sever side architecture.

6.1.1 Java

Java language offers several features that ease the development and deployment of a software environment for Grid computing. Its network-centric approach and its built-in support for mobile code enable the distribution of computational tasks to different computer platforms.

Java runtime systems are available for most hardware platforms and operating systems. Because of the heterogeneity of the hardware and of operating systems employed by Internet users, it is crucial that a platform for large-scale Grid computing be available for a large variety of different computer systems. Consequently, a Java- based platform potentially allows every computer in the Internet to be exploited for distributed, large-scale computations, while at the same time the maintenance costs for the platform are minimal (”write once, run everywhere”). Apart from its portability and compatibility, language safety and a sophisticated security model with flexible access control are further frequently cited advantages of Java. As security is of paramount importance for the acceptance of a platform for Grid computing, the security and safety features of Java are highly appreciated in this context.

6.2 System Model

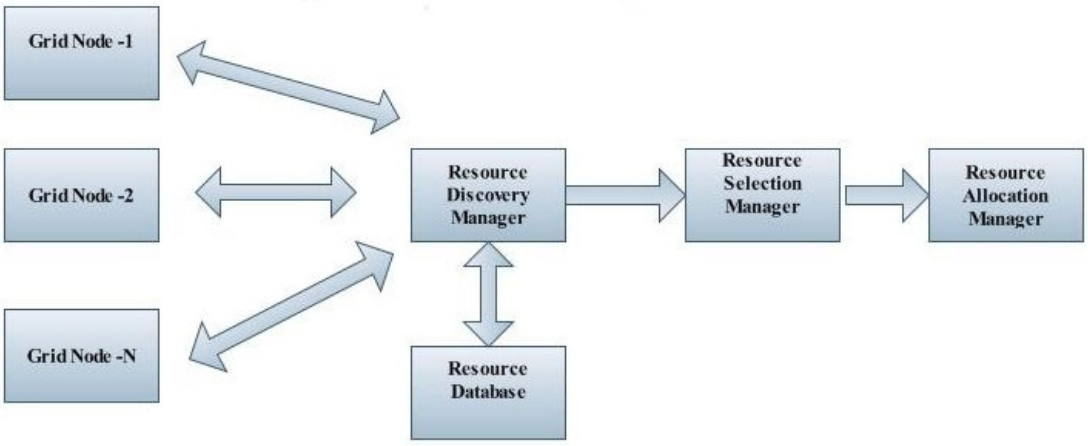


Figure 6.1: Proposed System Architecture

6.3 Process Flow Diagram

In Grid, resources are distributed at different locations. As shown in figure there are N different locations. When user queries for any resource,

- Resource Discovery manager searches for resources using Query Based Approach in Resource Database.
- Sends results to resource selection module.
- By applying different constraints criteria and simplex method, selection manager select best available resource location.
- Based on the result, user get the system name and IP address from where he can get resources of minimized cost or maximized performance.

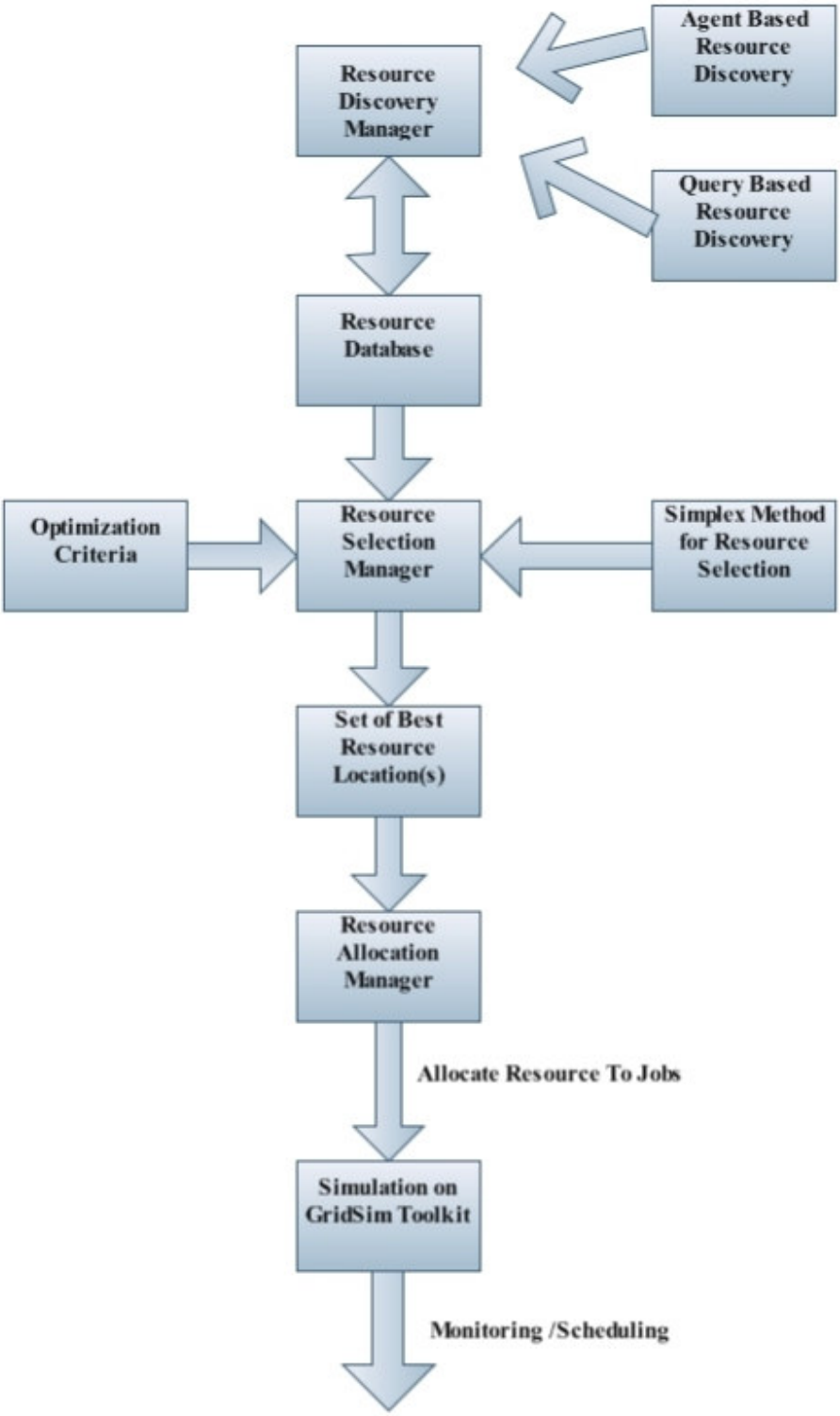


Figure 6.2: Process Flow Diagrams for Resource Allocation

6.4 Experimental Setup

The main aim of experiment is to demonstrate the effectiveness of Linear Programming based allocation strategy whose performance needs to be evaluated under different scenarios such as varying the number of users with different requirements. It is hard to perform performance evaluation involving multiple users in a repeatable and controllable manner for different scenarios due to dynamic nature of Grid environment. Therefore, this work simulates a Grid environment based on a Java-based discrete-event Grid simulation toolkit called GridSim [52]. The toolkit provides facilities for modeling and simulating Grid resources and Grid users with different capabilities and configuration.

To simulate application scheduling in GridSim environment requires the modeling and creation of GridSim resources and applications that model tasks. For the sake of simplicity in analyzing the result, it is assumed that all the users and resources stochastically similar within the respective groups with very small variances in their characteristics.

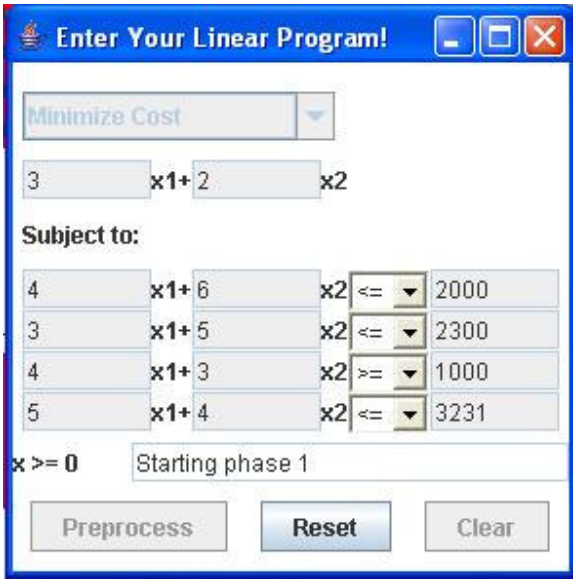
(1) Resource Modeling. Grid resources are modeled and simulated as many as different characteristics such as speed of processing, time zone, etc. The resources capability is defined as a MIPS rating. It varied with normal distribution from 0.5 to 1.5. For every Grid resource, local workload is estimated based on typically observed load conditions. The network communication speed between user and resource are defined in terms of data transfer bandwidth rate 100Mbps.

(2) Application Modeling. Grid users are modeled as many as 20 that are competing for resource. Each user consists of 4 tasks with variation of ± 1. Each task is heterogeneous in terms of task length and input files size. Task length is expressed in such a way that they are expected to take at least 20,000 MI (Million Instructions) with a random variation of 0 to 10% on the positive side. This 0 to 10% random variation in task length is introduced to model heterogeneous in different tasks. The user's budget varied with normal distribution from 90G$/s to 180G$/s. Here, G$ denotes Grid dollar.

As shown in the system architecture, the system is implemented in java and following are screenshots of the system.

Screenshot 1: Entry of resource location available and User needs

First screen displays resource location searched by resource discovery manager and number of resources that user entered at the time of discovery.



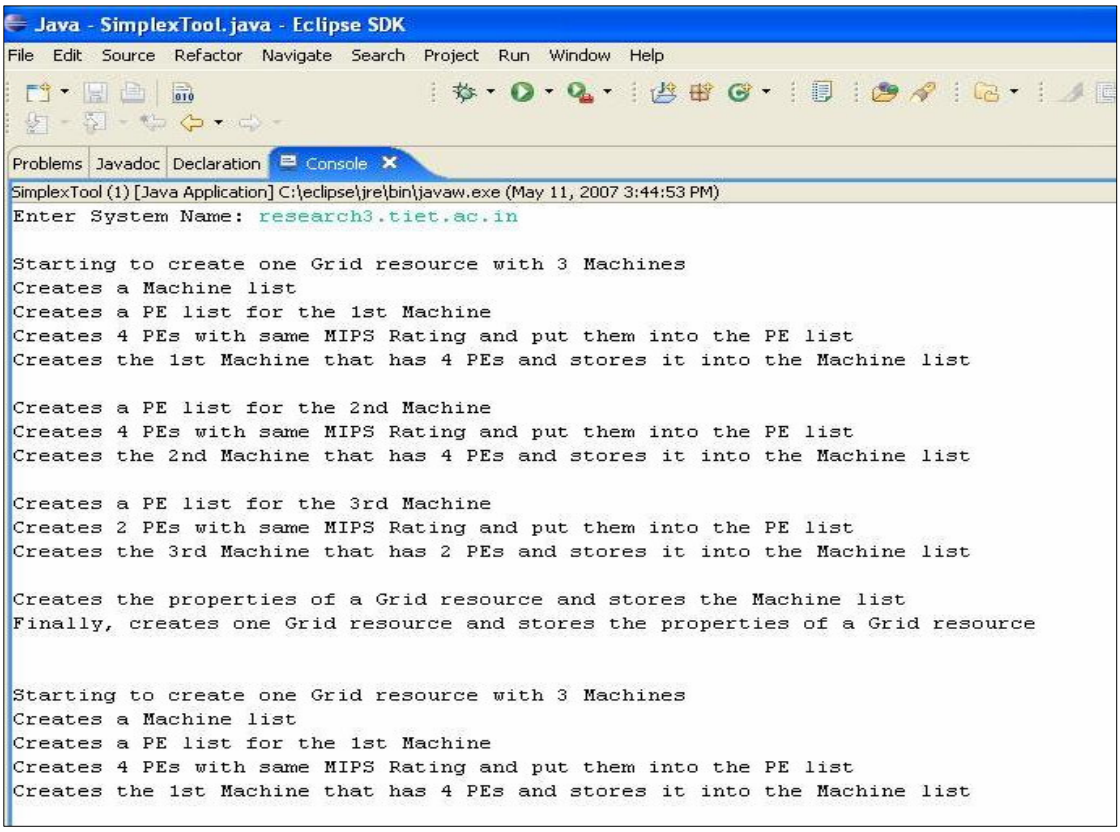
Screenshot 2: Resource Matrix

In Screenshot 3 first field displays the user's objective i.e. whether he wants to maximize performance or minimize cost. Second row displays type of resource user needs and in how much quantity.

Finally numbers of location where the resources are discovered is displayed in equations. Right hand side of equation displays maximum cost or maximum CPU speed and in LHS of equation number of resources available at different location are displayed.

Screenshot 3: Solved Resource Matrix Optimized Values

In above screen, resource matrix is solved using simplex method and based on this optimized result next step is performed i.e. allocate the resource from optimized location.



Screenshot 4: System Name with best resource available

When click on resource allocate than resource is allocated to job where best resources are available.

6.5 Experiment Results

Figure 6.3 illustrates how resource capacity and budget affect job execution time. As shown in Figure 6.3, for a low resource capacity (e.g. 75MIPS~150MIPS), the job execution time decreases 60% with the increase of resource capacity. Alternatively, when supply with a high resource capacity (e.g. 339MIPS~414MIPS), the job execution time decreases very slowly with the decrease of resource capacity. This is because when a higher resource capacity is available, the execution time of the task running on the resource becomes very small which cannot affect the job execution time. The Figure 6.4 and compare the job execution time under different range of resource capacity variation for the Round Robin and Linear Programming strategies. The X-axis shows a change in

resource capacity (total PE speed). The job execution time using Linear Programming allocation strategy can be as much as 36% shorter than that using the Round-Robin allocation when the resource capacity varies from 75MIPS~188MIPS. The reason for the performance improvement is that our optimal allocation considers user requirements while Round Robin does not. Figure 6.3 shows Job execution time of user i for different resource Capacity limits with a fixed budget for each of the user.

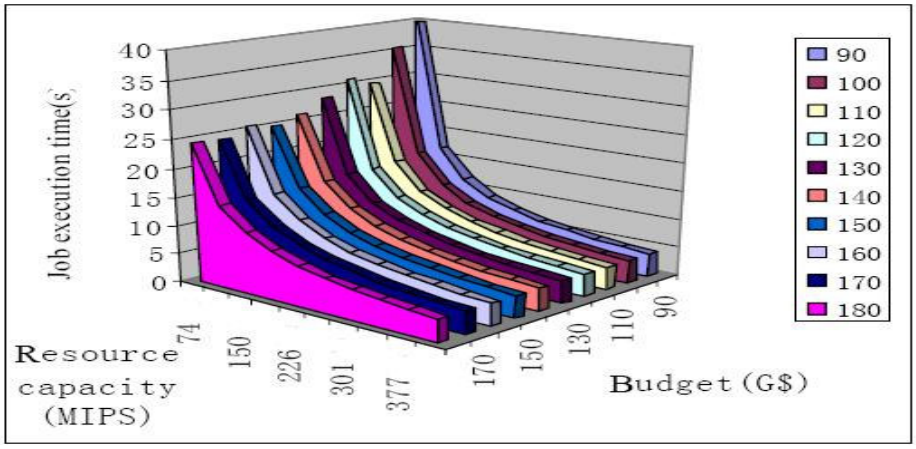


Figure 6.3: Job Execution Time V/s Resource Capacity and Fixed Budget

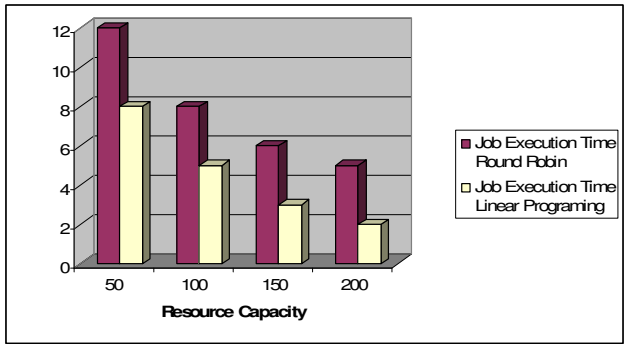


Figure 6.4: Comparison of job execution time with resource capacity values

6.6 Performance Analysis

To this end, we constructed several sets of random graphs of varying size and ran them through the algorithm to gather a significant amount of information in which to search for patterns. From the smaller graphs, which we had generated at first, we had conjectured that in the linear program solution x_1, x_2 for all edges assigned to a vertex, but in the

larger test graphs this was not always the case. For example we consider one linear programming problem in this example x and y are 2 resources.

Objective Function

Maximize $z = 2x + y$

Subject to the constraints:

1. $x \geq 0$,

2. $y \geq 0$,

3. $x + 2y \leq 5$,

4. $x - y \leq 2$. Then

We begin by graphing the region in quadrant I ($x \geq 0, y \geq 0$) formed by the constraints. Now we evaluate the objective function at the four vertices of this region.

Corners	**Objective Function** $z = 2x + y$
(0, 0)	$z = 2 \bullet 0 + 0 = 0$
(2, 0)	$z = 2 \bullet 2 + 0 = 4$
(3, 1)	$z = 2 \bullet 3 + 1 = 7$ **The maximum value of z.**
(0, 2.5)	$z = 2 \bullet 0 + 2.5 = 2.5$

Table6.1: Calculation of value of Z

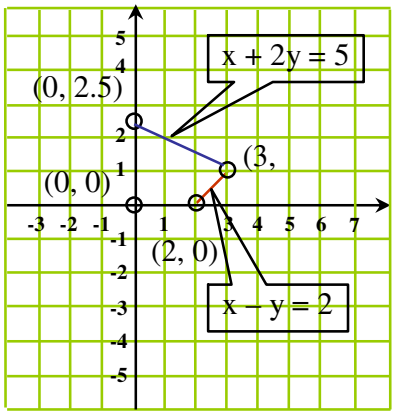


Figure 6.5: Graph of the constraints

Thus, the maximum value of z is 7, and this occurs when $x = 3$ and $y = 1$.

Chapter 7. Conclusion and Future Scope of Work

In this thesis, issues and challenges involved in dynamic resource allocation in Grid, have been addressed. In order to support resource allocation in Grid environment, the proposed solution is based on an intelligent technique, named linear programming for performance management. The effectiveness of the proposed solution has been verified on open source GridSim Toolkit version 4.0, for simulation of heterogeneous, controllable and repeatable test environment.

Main highlights of the thesis work are:

- Allocation of best resources using linear formulation discovered by resource discovery approaches.
- The allocation approach fine-tunes the cost and performance optimization tradeoff.
- Use of Simplex Approach [20] of linear programming problem to check complexity bounds.
- Experimental simulation results for the proposed approach have been compared with the Round Robin Allocation Approach.
- The proposed approach has been completely implemented in JAVA for complete platform independence.

7.1 Future Scope of Work

- The current implementation is done within a LAN with limited number of PCs; this can be ported to global Grid environment.
- The Approach proposed in this thesis has been only simulated on a virtual Grid Environment, before all the design goals can be verified and validated this Approach should be implemented in a Third Party Grid Resource Broker like GridBus Broker or Nimrod-G or Condor-G for testing in a Real Grid Environment.

References

[1]. B. veeravali and G.Barlas, Scheduling Divisible Loads with Processor Release Times and Finite Buffer Capacity Constraints in Bus Networks, Special Issue on Divisible Load Scheduling ,Cluster Computing ,2003.

[2]. B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel & S. Tuecke, "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing", In Mass Storage Conference, 2001.

[3]. BOINC project, Berkeley Open Infrastructure for Network Computing: http://boinc.berkeley.edu (Last visited on 03/11/2006)

[4]. B. Tierney, W. Johnston, J. Lee & G. Hoo, "Performance Analysis in High-Speed Wide Area IP over ATM Networks: Top-to-Bottom End-to-End Monitoring", IEEE Networking, 1996.

[5]. C. Baru, R. Moore, A. Rajasekar, & M. Wan, "The SDSC Storage Resource Broker", In Proceedings CASCON'98 Conference, 1998.

[6]. DataGrid: http://eu-datagrid.web.cern.ch/eu-datagrid/ (Last visited on 03/11/2006)

[7]. D. Abramson, R. Sosic, J. Giddy & B. Hall, "Nimrod: A Tool for Performing Parameterized Simulations Using Distributed Workstations", In Proceedings 4th IEEE Symposium On High Performance Distributed Computing, 1995.

[8]. D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini, Economic models for allocating resources in computer systems, 1996.

[9]. D. Kuokka and L. Harada, Matchmaking for information agents, Proceedings of the 14th International Joint Conference on Artificial Intelligence, 1995, pp. 672{678}.

[10]. D.Thain , T.Tannenbaum and M.Linvy, Condor and the grid , in F.Berman, A.J.G. Hey and G.Fox(Eds), Grid Computing :Making The Global Infrastructure a reality.

[11]. D. Thain, T. Tannenbaum, and M. Livny, Distributed computing in practice: The condor experience, 2004.

[12]. E. Gabriel, M. Resch, T. Beisel & R. Keller, "Distributed Computing in a Heterogeneous Computing Environment". In Proceedings EuroPVM/MPI'98, 1998.

[13]. F. Berman, “High-Performance Schedulers”, In I. Foster & C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999, 279-309.

[14]. F. Berman, R. Wolski, S. Figueira, J. Schopf & G. Shao, “Application-Level Scheduling on Distributed Heterogeneous Networks”, In Proceedings Supercomputing '96, 1996.

[15]. Gangadhar Mahesh kandru, Economy Based Resource Allocation in Grid”, 2006

[16]. H. Casanova, G. Obertelli, F. Berman & R. Wolski, “The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid”. In Proceedings SC'2000, 2000.

[17]. H. Casanova, J. Dongarra, C. Johnson & M. Miller, “Application-Specific Tools”, In I. Foster & C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999, 159-180.

[18]. H. Casanova & J. Dongarra, “NetSolve: A Network Server for Solving Computational Science Problems”, International Journal of Supercomputer Applications and High Performance Computing, 11(3):212-223. 1997.

[19]. Helmut Hlavacs and Christoph W. Ueberhuber, Dynamic load balancing on heterogeneous workstation clusters with irregularly fluctuating capacity.

[20]. http://people.hofstra.edu/faculty/Stefan_waner/RealWorld/tutorials4/frames4_3.htm

[21]. Ian Foster and Carl Kesselman, Computational Grids, Chapter 2 of “The Grid: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann Publishers, 1999. ISBN:1558604758

[22]. Ian Foster and Steve Tuecke “The Anatomy of the Grid”, in 2000.

[23]. I. Foster, C. Kesselman & S. Tuecke. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”. International Journal of High Performance Computing Applications, 2001.

[24]. I. Foster, C. Kesselman, G. Tsudik & S. Tuecke, “A Security Architecture for Computational Grids”. In ACM Conference on Computers and Security, 1998, 83-91.

[25]. I. Foster & N. Karonis, “A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems”. In Proceedings SC'98, 1998.

[26]. J. Beiriger, W. Johnson, H. Bivens, S. Humphreys & R. Rhea, “Constructing the ASCI Grid”, In Proceedings 9th IEEE Symposium on High Performance Distributed Computing, 2000, IEEE Press.

[27]. James F. Kurose and Rahul Simha, A microeconomic approach to optimal resource allocation in distributed computer systems, IEEE Trans. Comput. 38 (1989), no. 5,705{717}.

[28]. Junwei Cao, Darren J. Kerbyson, and Graham R. Nudd, Performance evaluation of an agent-based resource management infrastructure for Grid computing, CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid (Washington, DC, USA), IEEE Computer Society, 2001, p. 311.

[29]. J. Frey, I. Foster, M. Livny, T. Tannenbaum & S. Tuecke, “Condor-G: A Computation Management Agent for Multi-Institutional Grids”, University of Wisconsin Madison, 2001.

[30]. J. Howell & D. Kotz, “End-to-End Authorization”, In Proceedings 2000 Symposium on Operating Systems Design and Implementation, 2000, USENIX Association.

[31]. J.P. Goux, S. Kulkarni, J. Linderoth & M. Yoder, “An Enabling Framework for Master-Worker Applications on the Computational Grid”, In Proceedings 9th IEEE Symp. On High Performance Distributed Computing, 2000, IEEE Press.

[32]. J. Leigh, A. Johnson & T.A. DeFanti, “CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments”, Virtual Reality: Research, Development and Applications, 2(2):217-237. 1997.

[33]. K. Czajkowski, S. Fitzgerald, I. Foster & C. Kesselman, “Grid Information Services for Distributed Resource Sharing”, 2001.

[34]. Karl Czajkowski and Ian Foster and Nick Karonis and Carl Kesselman and Stuart Martin andWarren Smith and Steven Tuecke, A Resource Management Architecture for Metacomputing Systems, Lecture Notes in Computer Science 1459 (1998), 62

[35]. Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, A taxonomy and survey of Grid resource management systems for distributed computing, Software Practice and Experience 32 (2002), no. 2, 135{164}.

[36]. M. Beynon, R. Ferreira., T. Kurc, A. Sussman & J. Saltz, "DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems", In Proceedings 8th Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems, 2000, 119-133.

[37]. M. Gasser& E. McDermott., "An Architecture for Practical Delegation in a Distributed System", In Proceedings 1990 IEEE Symposium on Research in Security and Privacy, 1990, IEEE Press, 20-30.

[38]. M. Preist C., Van Tol, Adaptive agents in a persistent shout double auction, Proc.of 1st International Conference on the Internet Computing and Economics, 1998, pp. 11-17.

[39]. L. Childers, T. Disz, R. Olson, M.E. Papka, R. Stevens & T. Udeshi, "Access Grid: Immersive Group-to-Group Collaborative Visualization". In Proceedings 4th International Immersive Projection Technology Workshop, 2000.

[40]. LHC@Home: http://lhcathome.cern.ch/ (Last visited on 03/11/2006)

[41]. M. Maheswaran, S.Ali, H.j.Siegel, Hensgen, Dynamic Mapping of a class of Independent Tasks onto a Heterogeneous Computing Systems ,Journal of Parallel and Distributed Computing.

[42]. M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson & A. Essiari, "Certificate-based Access Control for Widely Distributed Resources". In Proceedings 8th Usenix Security Symposium, 1999.

[43]. M. Livny, "High-Throughput Resource Management". In I. Foster & C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999, 311-337.

[44]. Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, Eduardo Gómez-Sánchez,"Grid Characteristics and Uses: A Grid Definition." European Across Grids Conference 2003: 291-298

[45]. O.Ibarra and C. Kim,Heuristic Algorithms for scheduling Independent tasks on Non identical Processors, Journal of ACM ,24(2),1977

[46]. Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo, Asyn chronous complete method for general distributed constraint optimization, Proceedings of the first international joint conference on Autonomous agents and multiagent Systems Part I, July 2002.

[47]. R.Buyya, D.Abramson, and J.Giddy, Nimrod/g: architecture for a resource management and schedulling system in a global computational Grid, Proceedings of the HPC ASIA 2000, 4th International Conference on High Performance Computing in Asia- Pacific Region, IEEE Computer Society Press, USA, 2000.

[48]. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, Economic models for resource management and scheduling in Grid computing, 2002.

[49]. R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer & V. Welch, "Design and Deployment of a National-Scale Authentication Infrastructure", IEEE Computer, 33(12):60-66. 2000.

[50]. T.D. Braun, H.J. Siegel and N.Beck ,A comparison of Eleven Static Heuristic for Mapping a class of Independent Tasks onto Heterogeneous Platforms in IPDPS,April 2003.

[51]. Roura. An improved master theorem for divides and concurs recurrence .In Proceeding of Automata, Language and Programming.

[52]. R. Buyya, M. Murshed, "GridSim: A Toolkit forModeling and Simulation of Grid ResourceManagement and Scheduling", Journal of Concurrency and Computation: Practice and Experience, Vol. 14, No. 13-15, pp. 1175–1220, 2002.

[53]. Steve J. Chapin, Dimitrios Katramatos, John Karpovich, and Andrew Grimshaw, Resource management in legion, Future Gener. Comput. Syst. 15 (1999), no. 5-6, 583{594}.

[54]. Thomas Corman, Rivest ,Introduction to Algorithms ,Second Edition, PHI New Delhi 2004.

[55]. The globus project: a status report, Future Generation Computer Systems 15 (1999), no. 5{6, 607{621.}

[56]. W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger & K. Stockinger, "Data Management in an International Data Grid Project", In Proceedings 1st IEEE/ACM International Workshop on Grid Computing, 2000, Springer Verlag Press.

[57]. www.gnutella2.com. .(Last visited on 30/04/2007)

[58]. www.jxta.org.(Last visited on 30/04/2007)

[59]. www.freenet.sourceforge.net/ : For Freenet, P2P system

[60]. www.kazaa.com/us/index.htm: For Kazaa, P2P system

[61]. www.napster.com/ : For Napster, P2P system (Last visited on 30/04/2007)

[62]. www.p-grid.org: For P-Grid, P2P system (Last visited on 30/04/2007)

[63]. Xiaotong Shen and Jianming Ye, Adaptive model selection, Journal of the American Statistical Association 97 (2002), no. 457, 210

Paper Accepted /Published

1. Anurag Gothi, Seema Bawa, "Performance Optimization Oriented Resource Allocation in Grid Computing", Accepted in International Conference on Emerging Trends In High Performance Architecture, Algorithms & Computing, **HiPAAC 2007,** Chennai (July11-13, 2007).

2. Anurag Gothi, Seema Bawa, "Grid Resource Allocation: Challenges and Processes", Accepted in National Conference on Digital Information Management-NCDIM"07, Mumbai (March 23-24, 2007).

3. Anurag Gothi, Seema Bawa, "Features of Grid Resource Allocation Patterns" in Proceedings of National Seminar on Business Transformation Through Technological Integration ISTE Sponsored, PIMT Mandi Gobindgarh, March 30, 2007.

4. Anurag Gothi, Seema Bawa, "Grid Resource Allocation: Challenges and Processes", Presented in National Conference on Information Technology: Emerging Engineering Prospective and Practices-ITEEPP"07, Thapar University, Patiala (April 06-07,2007).