M.Sc. Engg. Thesis

A Distributed Optimized Resource Reservation Scheme for Grid Computing

by

Rifat Shahriyar

Submitted to

Department of Computer Science and Engineering
in partial fulfilment of the requirments for the degree of
Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka 1000

December, 2009

The thesis titled '**A Distributed Optimized Resource Reservation Scheme for Grid Computing**', submitted by Rifat Shahriyar, Roll No. 100705037P, Session October 2007, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on December 27, 2009.

Candidate's Declaration

This is to certify that the work entitled 'A Distributed Optimized Resource Reservation Scheme for Grid Computing' is the outcome of the research carried out by me under the supervision of Dr. Md. Mostofa Akbar in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Rifat

Rifat Shahriyar
Candidate

Contents

List of Figures

List of Tables

Acknowledgments

All praises due to Allah, the most benevolent and merciful.

I express my heart-felt gratitude to my supervisor, Dr. Md. Mostofa Akbar for his constant supervision of this work. He helped me a lot in every aspect of this work and guided me with proper directions whenever I sought one. His patient hearing of my ideas, critical analysis of my observations and detecting flaws (and amending thereby) in my thinking and writing have made this thesis a success.

I also want to show my gratitude to Dr. M. Sohel Rahman for his patience and valuable suggestions. It has been a great opportunity to work with him.

I also want to thank the other members of my thesis committee: Dr. Md. Monirul Islam, Dr. M. Kaykobad, and Dr. Hasan Sarwar for their valuable suggestions.

I would like to express my ever gratefulness to my mother, sister, grand mother and wife for their continuous support. Finally, I cannot forget some of my friends (Md. Faizul Bari, Atif Hasan Rahman and Enamul Hoque) for their supports. May Allah reward them all in here and hereafter.

Abstract

The idea of grid computing originated in the scientific community and was initially motivated by processing power and storage intensive applications. The basic objective of grid computing is to support resource sharing among individuals and institutions within a networked infrastructure. Managing various resources in highly dynamic grid environments is a complex and challenging problem. There are works for resource management in different areas of computer science. Some approach uses algorithms for resource management to apply in grid but fails to provide any generalized solutions for grid environment. Most of the approaches are based on a simple architecture considering computer as the main resource in their system. But the real architecture of grid computing is a complex one by considering various resources of any computer during resource management. In grids sometimes assurance is needed for successful completion of jobs on shared resources. Such guarantees can only be provided by reserving resources in advance. So resource reservation is an integral part of resource management system for grid. Moreover the cost for providing resource as services will play a significant role in near future when resource sharing will be popular and inevitable but so far there is no work regarding cost optimization model for grid computing. In this thesis we provide a future reservation supported and cost optimized novel resource management system for grid environment considering its real complex architecture. We demonstrate our claims by conducting a detailed performance evaluation and comparing with the existing system on real workload traces for grid computing.

Chapter 1

Introduction

1.1 What is Grid Computing?

Grid systems have emerged as promising next-generation computing platforms that enable the building of a wide range of collaborative problem-solving environments in industry, science and engineering. Grid environments enable flexible, secure and coordinated resource sharing among dynamic collections of institutions distributed across the world called virtual organizations [13].

The term grid, coined in the mid 90s in the academic world, was originally proposed to denote a distributed computing system that would provide computing services on demand just like conventional power and water grids do. During the last few years, as the technology evolved and the grid concept started being explored on commercial endeavors, some slight but meaningful changes have been made in its original definition. Nowadays, an accepted definition, world-wide, states that a grid is a system that:

- coordinates resources that are not subject to centralized control
- uses standard, open, general-purpose interfaces and protocols
- delivers non-trivial quality of service

Nowadays, most of the interest driven toward the grid concept derives from the fact that, stated as it is, a grid can be regarded as a technology with no boundaries.In fact, if one can integrate all its computing resources, no matter what they are, in a single virtual computing environment, such a system would make possible:

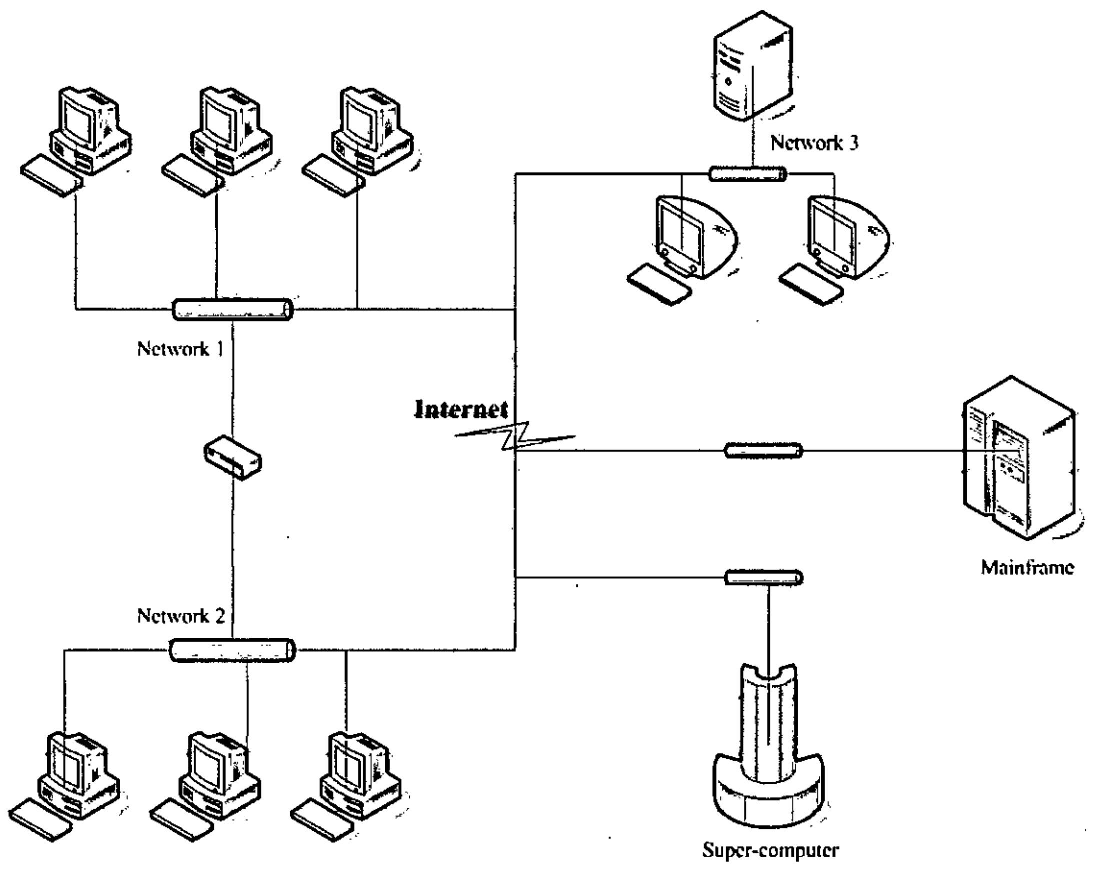

Figure 1.1: Grid : a setup with heterogeneous and independent computing resources

- The effective use of computing resources that otherwise would remain idle for most of the time
- To perform complex and computing-demanding tasks that would normally require large-scale computing resources.

As Web technologies have changed the way that information in shared all over the world, grid computing aims at being the next technological revolution, integrating and making available not only information, but also computing resources such as computing power and data-storage capacity [18]. Figure 1.1 illustrates the way that a grid can be built by means of computing resources that are somehow interconnected by the Internet but that there is no relation among them.

1.2 Use of Grids in Research and Education

This section presents a brief discussion on which types of research and educational activities could benefit from grid computing technologies. So far we have presented what a grid is, but haven't gone into details on what it can do. Actually, it is not difficult to figure out how useful a high-performance computing infrastructure can be, but this is not all of the truth. The fact is that such an infrastructure can be built up from computing resources that are already available, which the reason why grids are so appealing. ~~Briefly stated, a computational grid provides high-performance~~ computing; a data grid provides large storage capacity; and a network grid provides high throughput communication that may be useful for a variety of applications, such as virtual conferences. Having this in mind, we can list the main reasons for using grid computing as follows:

- Improve efficiency/reduce costs
- Exploit under-utilized resources
- Enable collaborations
- Virtual resources and virtual organizations
- Increase capacity and productivity
- Parallel processing capacity
- Support heterogeneous systems
- Provide reliability/availability
- Access to additional resources
- Resource balancing
- Reduce time to results

When these reasons are regarded under the light of scientific research, it is easy to understand why scientists are so keen on grids: they believe that the use of grids will transform the practice of their science.

1.3 What is the Potential for Grids?

~~Because of World Wide Web revolution, networked desktop computers can be found everywhere~~ today. Government, industry, universities, and other research and educational institutions rely on every sort of computer to perform their daily activities, and this *computer-based society* is growing bigger every day. The computational grid has reached a much higher level of maturity than the other types of grid. This may derive from the fact that computing science has always been concerned with computing activities, for obvious reasons, and that distributed computing research and development has been on the road for more than 30 years now. In addition to this, we know that as time goes by and technology evolves, the computers and network connections get faster and more reliable. As a result, the global computing pool becomes more powerful and more strongly coupled, leading to systems that can handle large amounts of data in shorter periods of time. These factors can be summarized in a few words under the grid perspective: A global grid infrastructure is evolving to be readily available in the near future. Knowing that this infrastructure will be somehow available, we should analyze the potential applicability of grid technologies. There are a great number of research and educational areas that could benefit from grid technology; having them in mind, we can say that the potential for grid technology applications depends on the following facts:

- After the World Wide Web, the grid has been regarded as the next natural step towards the evolution of information technology.
- The forthcoming scientific breakthroughs are likely to be brought by the power unleashed by grid computing.
- Such a powerful computing infrastructure can embrace existing and brand new paradigms of application execution.
- Researchers and educators do believe that the grid will come to reality and are looking forward to using it.

For all these reasons, we believe that grid computing will form part of an inexorable future, changing the way that research, education, and even ordinary or everyday tasks are performed. These are some of the possibilities that might arise from the grid world, and there is no doubt that they will definitely change the way that we deal with information in our personal and professional activities.

1.4 Motivation for the Work

The idea of above mentioned grid computing was initially motivated by processing power and storage intensive applications. The basic objective of grid computing is to support resource sharing among individuals and institutions within a networked infrastructure. Managing various resources in highly dynamic grid environments is a complex and challenging problem. There are works for resource management in different areas of computer science. Some approach uses data structures [6] [5] [24] and algorithms [14] for resource management to apply in grid but fails to provide any generalized solutions for grid environment. Most of the approaches are based on a simple architecture considering computer as the main resource in their system. But the real architecture of grid computing is a complex one by considering various resources of any computer during resource management. In grids sometimes assurance is needed for successful completion of jobs on shared resources. Such guarantees can only be provided by reserving resources in advance [21] [27]. So resource reservation is an integral part of resource management system for grid.

Moreover grids are used as a volunteer service now a days. But with the recent improvements in architecture and usage situation will not be the same. Cost for providing resource as services will play a significant role in near future when resource sharing will be popular and inevitable but so far there is no work regarding cost optimization model for grid computing. A complete resource management system for grid computing is required to support all the above mentioned features. So the main motivation of this thesis work is to provide a future reservation supported and cost optimized novel resource management system for grid environment considering its real complex architecture.

1.5 Scope of the Work

The main focus of this thesis work is to design a resource management system for grid computing with the following characteristics:

- It works in a distributed manner for grid architecture which is complex and depicts the real scenario.
- It uses appropriate and efficient data structures to represent the grid architecture.
- It provides support for both instant request acceptation/rejection and future resource reservation for any job.
- It optimizes the overall performance by reclaiming unused resources after a threshold time.

- It optimizes the cost by choosing the appropriate set from a list of possible resource providers by mapping the problem to fractional knapsack, a well known cost optimization problem.

The main outcome of this thesis work is a distributed, future reservation supported and cost optimized resource management system for grid environment. We performed the detailed performance evaluation of our prototype and compared with an existing system using real workload traces. Our proposed claims are justified by the comparative analysis presented on the experimental results on the workload traces.

1.6 Overview of the Thesis

The rest of the chapters are organized as follows. Chapter 2 gives a preliminary description of some terminologies and concepts related to grid computing that may be helpful to understand the context of this thesis. The detailed description of grid resources is presented in this chapter. The related works on resource management for grid computing is also presented in this chapter. Chapter 3, the main chapter of this dissertation, illustrates our proposed resource management scheme and the data structures used by this scheme. The algorithms related to proposed resource management scheme with their message complexity are also given in this chapter. Chapter 4 contains the simulation results of our scheme and a comparative study against existing system in several performance issues. The details of the simulator and simulation results on randomly generated data and real workload are presented here. Chapter 5 draws the conclusion describing the key contributions of this thesis followed by some future research directions related to this topic.

Chapter 2

Preliminaries

2.1 Grid Computing and its Goal

Grid computing can mean different things to different individuals. The grand vision is often presented as an analogy to power grids where users (or electrical appliances) get access to electricity through wall sockets with no care or consideration for where or how the electricity is actually generated. In this view of grid computing, computing becomes pervasive and individual users (or client applications) gain access to computing resources (processors, storage, data, applications, and so on) as needed with little or no knowledge of where those resources are located or what the underlying technologies, hardware, operating system, and so on are [12]. Though this vision of grid computing can capture one's imagination and may indeed someday become a reality, there are many technical, business, political, and social issues that need to be addressed. If we consider this vision as an ultimate goal, there are many smaller steps that need to be taken to achieve it. These smaller steps each have benefits of their own.

Integration of various entities Grid computing can be seen as a journey along a path of integrating various technologies and solutions that move us closer to the final goal. Its key values are in the underlying distributed computing infrastructure technologies that are evolving in support of cross-organizational application and resource sharing across technologies, platforms, and organizations. This kind of virtualization is only achievable through the use of open standards. Open standards help ensure that applications can transparently take advantage of whatever appropriate resources can be made available to them. An environment that provides the ability to share and transparently access resources across a distributed and heterogeneous environment not only requires the technology to virtualize certain resources, but also technologies and standards in the areas of scheduling, security, accounting, systems management, and so on.

Intra and inter organizational grids Early implementations of grid computing have tended to be internal to a particular company or organization. However, cross-organizational grids are also being implemented and will be an important part of computing and business optimization in the future. The distinctions between intra-organizational grids and inter-organizational grids are not based in technological differences. Instead, they are based on configuration choices given: Security domains, degrees of isolation desired, type of policies and their scope, and contractual obligations between users and providers of the infrastructures. These issues are not fundamentally architectural in nature. It is in the industry's best interest to ensure that there is not an artificial split of distributed computing paradigms and models across organizational boundaries and internal IT infrastructures.

Goal as a part of distributed computing Grid computing involves an evolving set of open standards for Web services and interfaces that make services, or computing resources, available over the Internet. Very often grid technologies are used on homogeneous clusters, and they can add value on those clusters by assisting, for example, with scheduling or provisioning of the resources in the cluster. The term grid, and its related technologies, applies across this entire spectrum. If we focus our attention on distributed computing solutions [15], then we could consider one definition of grid computing to be distributed computing across virtualized resources. The goal is to create the illusion of a simple yet large and powerful virtual computer out of a collection of connected (and possibly heterogeneous) systems sharing various combinations of resources.

2.2 Types of Grids

Ideally, a grid should provide full-scale integration of heterogeneous computing resources of any type: processing units, storage units, communication units, and so on. However, as the technology hasn't yet reached its maturity, real-world grid implementations are more specialized and generally focus on the integration of certain types of resources. As a result, nowadays we have different types of grids, which we describe as follows:

- *Computational grid*: A computational grid is a grid that has the processing power as the main computing resource shared among its nodes. This is the most common type of grid and it has been used to perform high-performance computing to tackle processing-demanding tasks.

- *Data grid*: Just as a computational grid has the processing power as the main computing resource shared among their nodes; a data grid has the data storage capacity as its main shared resource. Such a grid can be regarded as a massive data storage system built up from portions of a large number of storage devices [17].

- *Network grid*: This is known as either a network grid or a delivery grid. Such a grid has as its main purpose to provide fault-tolerant and high-performance communication services. In this sense, each grid node works as a data router between two communication points, providing data-caching and other facilities to speed up the communications between such points.

2.3 Terminologies Related to Grid Computing

In ~~this chapter~~ we ~~introduce~~ the ~~terminologies~~ ~~and~~ ~~concepts~~ ~~that~~ ~~are~~ ~~used~~ ~~frequently~~ ~~used~~ ~~in~~ ~~grid~~ computing. A grid is a collection of machines (sometimes referred to as nodes), resources, members, donors, clients, hosts, engines, and many other such terms. They all contribute any combination of resources to the grid as a whole. Some resources may be used by all users of the grid, while others may have specific restrictions. As we mainly focus on grid resources, they are described in more details in the next section.

2.4 Types of Resources

2.4.1 Computation

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are three primary ways to exploit the computation resources of a grid.

- The first and simplest is to use it to run an existing application on an available machine on the grid rather than locally.

- The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors.
- The third is to run an application that needs to be executed many times, on many different machines in the grid.

Scalability is a measure of how efficiently the multiple processors on a grid are used. If twice as many processors makes an application complete in one half the time, then it is said to be perfectly scalable. However, there may be limits to scalability when applications can only be split into a limited number of separately running parts or if those parts experience some other interdependencies such as contention for resources of some kind.

2.4.2 Storage

The second most common resource used in a grid is data storage.A grid providing an integrated view of data storage is sometimes called a data grid. Each machine on the grid usually provides some quantity of storage for grid use, even if temporary.

Primary Storage Storage can be memory attached to the processor or it can be secondary storage, using hard disk drives or other permanent storage media. Memory attached to a processor usually has very fast access but is volatile. It would best be used to cache data or to serve as temporary storage for running applications.

Secondary Storage Secondary storage in a grid can be used in interesting ways to increase capacity, performance, sharing, and reliability of data. Many grid systems use mountable networked file systems, such as Andrew File System (AFS), Network File System (NFS), Distributed File System (DFS), or General Parallel File System (GPFS). These offer varying degrees of performance, security features, and reliability features. Capacity can be increased by using the storage on multiple machines with a unifying file system. Any individual file or database can span several storage devices and machines, eliminating maximum size restrictions often imposed by file systems shipped with operating systems. A unifying file system can also provide a single uniform name space for grid storage. This makes it easier for users to reference data residing in the grid, without regard for its exact location. In a similar way, special database software can federate an assortment of individual databases and files to form a larger, more comprehensive database, accessible using database query functions. More advanced file systems on a grid can automatically duplicate sets of data, to provide redundancy for increased reliability and increased performance. An intelligent grid scheduler can help select the appropriate storage devices to hold data, based on usage patterns. Then jobs can be scheduled closer to the data, preferably on the machines directly connected to the storage devices holding the required data.

Data Striping Data striping can also be implemented by grid file systems. When there are sequential or predictable access patterns to data, this technique can create the virtual effect of having storage devices that can transfer data at a faster rate than any individual disk drive. This can be important for multimedia data streams or when collecting large quantities of data at extremely high rates from CAT scans or particle physics experiments, for example.

Journaling A grid file system can also implement journaling so that data can be recovered more reliably after certain kinds of failures. In addition, some file systems implement advanced synchronization mechanisms to reduce contention when data is shared and updated by many users.

2.4.3 Communications

The rapid growth in communication capacity among machines today makes grid computing practical, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, it should not be a surprise that another important resource of a grid is data communication capacity. This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed, and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid. External communication access to the Internet, for example, can be valuable when building search engines. Machines on the grid may have connections to the external Internet in addition to the connectivity among the grid machines. When these connections do not share the same communication path, then they add to the total available bandwidth for accessing the Internet. Redundant communication paths are sometimes needed to better handle potential network failures and excessive data traffic. In some cases, higher speed networks must be provided to meet the demands of jobs transferring larger amounts of data.

2.4.4 Software and Licenses

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

2.4.5 Special Equipment, Capacities, Architectures, and Policies

Platforms on the grid will often have different architectures, operating systems, devices, capacities, and equipment. Each of these items represents a different kind of resource that the grid can use as criteria for assigning jobs to machines. While some software may be available on several architectures, for example, PowerPC and x86, such software is often designed to run only on a particular type of hardware and operating system. Such attributes must be considered when assigning jobs to resources in the grid. In some cases, the administrator of a grid may create a new artificial resource type that is used by schedulers to assign work according to policy rules or other constraints. For example, some machines may be designated to only be used for medical research. These would be identified as having a medical research attribute and the scheduler could be configured to only assign jobs that require machines of the medical research resource. Others may participate in the grid only if they are not used for military purposes. In this situation, jobs requiring a military resource would not be assigned to such machines. Of course, the administrators would need to impose a classification on each kind of job through some certification procedure to use this kind of approach.

2.4.6 Applications and Jobs

Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing application or job. Usually we use the term *application* as the highest level of a piece of work on the grid. However, sometimes the term *job* and *application* are used equivalently.

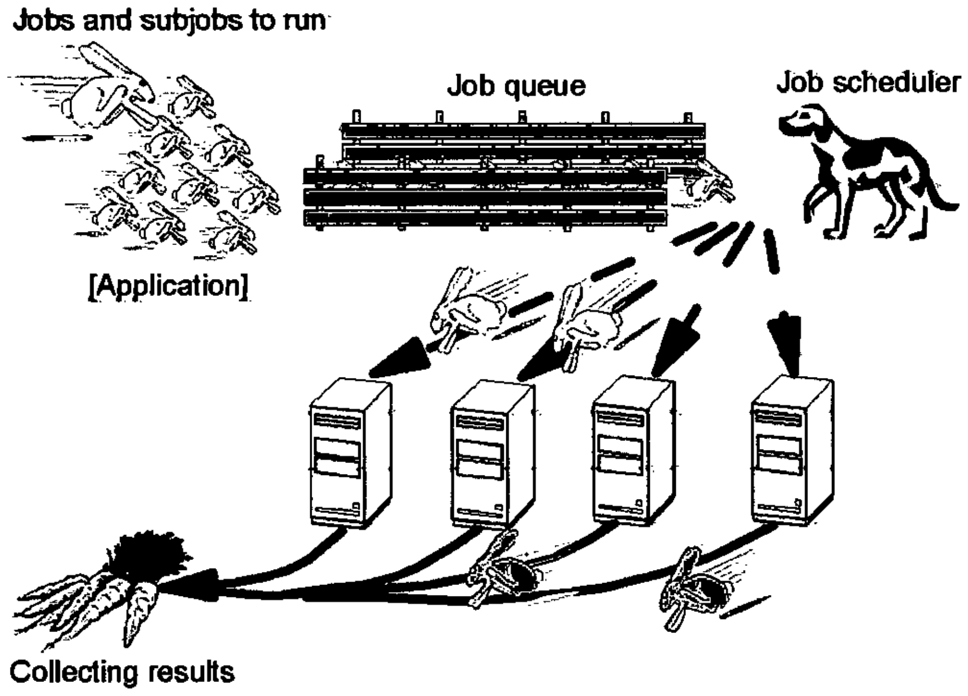

Figure 2.1: An application is one or more jobs that are scheduled to run on grid

Applications may be broken down into any number of individual jobs. Those, in turn, can be further broken down into sub jobs. The grid industry uses other terms, such as transaction, work unit, or submission, to mean the same thing as a job.

Jobs are programs that are executed at an appropriate point on the grid. They may compute something, execute one or more system commands, move or collect data, or operate machinery. A grid application that is organized as a collection of jobs is usually designed to have these jobs execute in parallel on different machines in the grid. The jobs may have specific dependencies that may prevent them from executing in parallel in all cases. For example, they may require some specific input data that must be copied to the machine on which the job is to run. Some jobs may require the output produced by certain other jobs and cannot be executed until those prerequisite jobs have completed executing. Jobs may spawn additional sub jobs, depending on the data they process. This work flow can create a hierarchy of jobs and sub jobs. Finally, the results of all of the jobs must be collected and appropriately assembled to produce the ultimate output/result for the application.

2.4.7 Scheduling, Reservation, and Scavenging

The grid system is responsible for sending a job to a given machine to be executed. In the simplest of grid systems, the user may select a machine suitable for running his job and then execute a grid command that sends the job to the selected machine.

Scheduling More advanced grid systems would include a job scheduler of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid. The term scheduling is not to be confused with reservation of resources in advance to improve the quality of service. Sometimes the term resource broker is used in place of scheduler, but this term implies that some sort of bartering capability is factored into scheduling.

Scavenging In a scavenging grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job whose requirements are satisfied by the machine's resources. Scavenging is usually implemented in a way that is unobtrusive to the normal machine user. If the machine becomes busy with local non-grid work, the grid job is usually suspended or delayed. This situation creates somewhat unpredictable completion times for grid jobs, although it is not disruptive to those machines donating resources to the grid. Grid applications that run in scavenging mode often mark themselves at the operating system's lowest priority level. In this way, they only run when no other work is pending. Due to the performance of modern day processors and operating system scheduling algorithm, the grid application can run for as short as a few milliseconds, even between a user's keystrokes.

Reservation To create more predictable behavior, grid machines are often dedicated to the grid and are not preempted by outside work. This enables schedulers to compute the approximate completion time for a set of jobs, when their running characteristics are known. As a further step, grid resources can be reserved in advance for a designated set of jobs. Such reservations operate much like a calendaring system used to reserve conference rooms for meetings. This is done to meet deadlines and guarantee quality of service. When policies permit, resources reserved in advance could also be scavenged to run lower priority jobs when they are not busy during a reservation period, yielding to jobs for which they are reserved.

Thus, various combinations of scheduling, reservation, and scavenging can be used to more completely utilize the grid. Scheduling and reservation is fairly straightforward when only one resource type, usually CPU, is involved. However, additional grid optimizations can be achieved by considering more resources in the scheduling and reservation process. For example, it would be desirable to assign executing jobs to machines nearest to the data that these jobs require. This would reduce network traffic and possibly reduce scalability limits. Optimal scheduling, considering multiple resources, is a difficult mathematics problem. Therefore, such schedulers may use heuristics. These heuristics are rules that are designed to improve the probability of finding the best combination of job schedules and reservations to optimize throughput or any other metric.

2.5 Grid by Examples

In the following subsections we provide real life examples of grid computing implementations in the areas of research and education.

2.5.1 Scientific Simulation

It is a grid implementation to provide the execution of complex system simulations in the areas of physics, chemistry, and biology. The implementation tackles the problem of intensive calculations, which demands high performance computing and typically requires large computational infrastructures such as clusters. This type of solution has already been set in place in a number of research institutions in Japan including National Institute of Advanced Industrial Science and Technology, AIST [19].

2.5.2 Medical Images

It is a data and computational grid in medical image storage and processing framework. This grid tackles the problem of storing and processing large images, which typically requires large computational infrastructures such as distributed databases and clusters. This type of solution has already been set in place in the eDiaMoND project. This is a collaborative project funded by grants from the Engineering and Physical Sciences Research Council (EPSRC), which is the UK Government's leading funding agency for research and training in engineering and the physical sciences, Department of Trade and Industry (DTI), and IBM. It is strictly a research project which has the ambitious aim of proving the benefits of grid technology to eHealth, in this case for Breast Imaging in the UK. More information about the eDiaMoND project can be found at [10].

2.5.3 Computer-Aided Drug Discovery

It is a grid implementation that tackles the problem of Computer-Aided Drug Discovery (CADD), which demands high performance computing and typically requires large computational infrastructures such as clusters, mainframes, or super-computers. This type of solution has been set in place at the Molecular Modeling Laboratory (MML) at the University of North Carolina (UNC) at Chapel Hill, School of Pharmacy.

2.5.4 Big Science

It is an implementation of a data and computational grid to support government sponsored laboratory types of projects (also known as Big Science). The system accomplishes the problem of storing huge quantities of data, which demands high storage capacity and typically requires large and parallel computational infrastructures. The data grid implementation is based on the IBM General Parallel File System (GPFS). This type of solution has already been set in place in DEISA, a consortium of leading national supercomputing centers in Europe aiming to jointly build and operate a distributed terascale super computing facility. More information about the DEISA project can be found at [11].

2.5.5 E-Learning

It is a grid environment to support many of the educational and research requirements for exchanging information. Knowing the main ways that education can benefit from grid technology, we can deduce the basic technological needs associated with the development of e-learning. The e-learning infrastructure is based on the Access Grid. Access Grid is an ensemble of resources including multimedia large-format displays, presentation and interactive environments, and interfaces to grid middleware and to visualization environments. The AG technology was developed by the Futures Laboratory at Argonne National Laboratory and is deployed by the NCSA PACI Alliance. For more information about the Access Grid, refer to [2].

2.5.6 Visualization

It is a grid implementation to support the field of advanced scientific visualization. The area of visualization is evolving as it addresses emerging and continuing issues, such as interactive and batch rendering of terascale data sets, through remote visualization. At the same time, universities in general have a lot of heterogeneity, using many low-cost resources from different suppliers. This includes running different systems through advanced computing resources, such as super computers, advanced visualization systems, etc. Most of these resources are segregated in specific departments for local access only. This example of grid implementation is inspired by the scientific visualization requirements from the America's largest campus grid, University of Texas at Austin. More information about this campus grid can be found at [8].

2.5.7 Microprocessor Design

It is a computational grid solution that helps to reduce the microprocessors' development cycle and also allows the design centers to share their resources more efficiently. Microprocessor design and microprocessor verification simulation requires massive computational power. This type of solution has been in place in the Microprocessor Design Group at IBM Austin, TX. They design chips for the IBM Eserver high-performance systems, running thousands of simulations to verify timing closure.

2.6 Resource Management in Grid

~~Resource management is a complex task involving security, fault tolerance along with scheduling.~~ It is the manner in which resources are allocation, assigned, authenticated, authorized, assured, accounted, and audited. Resources include traditional resources like compute cycles, network bandwidth, space or a storage system and also services like data transfer, simulation etc.

Grid systems are inter connected collections of heterogeneous and geographically distributed resource harnessed together to satisfy various needs of the users. Resource management is the central component of a grid system. Resource management in grid systems is complex due to various factors like site autonomy, resource heterogeneity etc. Traditional resource management systems work under the assumption that they have complete control on the resource and thus can implement the mechanisms and policies needed for effective use of that resource. But in

Grid systems resources are distributed across separate administrative domains resulting resource heterogeneity, differences in usage, scheduling policies, security mechanisms.

2.7 Related Works on Resource Management in Grid Computing

This section provides an overview of the related works in this topic.

Liang [9] addresses the problem of resource allocation in the GATES (Grid-based Adaptive Execution on Streams) system. They present a resource allocation algorithm that is based on minimal spanning trees. They also evaluate the algorithm experimentally and demonstrate that the results are very close to optimal, and significantly better than most of the other possible configurations. The problem of this system is that it fails to provide any real and generalized architecture for conventional grid computing. It mainly works on multimedia grid based streaming applications. But the conventional grid is much more complex one.

Kun [25] addresses the challenge of providing efficient resource on demand for grid computing from the perspective of network, the living platform of Grid, by providing effective Quality of Service (QoS) mechanisms (both IntServ and DiffServ) inside the Grid networking environment. Specifically, the efficiency of this QoS mechanism is maximized by taking care of the flexible control of QoS parameters/components using policy-based management. It provides solution for on demand resource request but there is no provision for resource reservation for future use.

Yang [28] introduces new admission control strategies by fixing a threshold on resource reservation. Threshold does not allow all requests to have same QoS and to share the entire resources without concerning service priority. Through comprehensive theoretical analysis and extensive simulations, they demonstrate that the strategy with layered threshold is more efficient and flexible than the existing strategies for Grid-based multimedia services systems. This new strategy works using Markov model. It mainly focuses on multimedia systems.

Jianbing [23] introduces a flexible advance reservation for grid applications. Its parameters can be modified according to resource status in order to fill the gaps of resource. Admission control algorithm for this new type of reservation is provided too. Simulation shows that it can improve performance of resource reservation in terms of both call acceptance rate and resource utilization. It mainly focuses on grid of network resource only by using time slots like TDMA.

Jie [26] presents a predictive admission control algorithm to decide whether new advance reservation requests can be accepted according to their QoS requirements and prediction of future resource utilization. It is assumed that once an advance reservation request is accepted, it will definitely be fulfilled. But in practice, it is not always the case. In equipment grid certain special reasons may prevent a confirmed advance reservation from being fulfilled. Examples include resource malfunctions and preemption by more urgent tasks from local schedulers, which are often associated with economic benefits. When confirmed contracts cannot be fulfilled, the reputation of the providers of reserved resources will be ruined and the claimed benefits will be affected. The unfulfillment of accepted advance reservations will cause damages both to the clients and to the equipment grid. They propose a predictive admission control algorithm to avoid such situation by refusing some advance reservation requests which may not be fulfilled according to QoS requirements and historical information. This research mainly focuses on equipment grids which is quite simple in architecture than the conventional grid computing. One major drawback is that their algorithm doesn't work for multiple resources.

Rafael [20] introduces the concept of 0-1 knapsack problem for resource allocation in grid computing. They introduce a utility model for resource allocation on computational grids and formulate the allocation problem as a variant of the 0-1 multichoice multidimensional knapsack problem. They propose a variety of allocation policies for resource in grid. But they do not consider the various hardware or software of a computing node as the resource. They consider computer as the only resource and that is why they used the concept of 0-1 knapsack problem. But we need to consider various hardware or software of a computing node as the resource to represent the actual grid architecture.

Sulistio [22] presents new approaches to advance reservation in order to deal with the limitations of the existing data structures, such as Calendar Queue in similar problems. They propose a Grid advanced reservation Queue (GarQ), which is a new data structure that improves some weaknesses of the aforementioned data structures but highly depends on parameters such as size of interval. Moreover they consider computer as the only resource thus making the system very simple one with no provision for resource utilization.

2.7.1 General Review of Related Works

A general review of the mentioned related works on resource management for grid computing is given below:

- Computer is considered as the only resource. But we need to consider various hardware and software of a computer as the resources.
- Most of the assumed grid architecture is very simple but the actual architecture is quite complex.
- There is no complete system for resource management and no complete cost model for resource sharing so far. But in the near future resource sharing will be popular and inevitable.

Chapter 3

A New Reservation Based Resource Management Scheme

We proposed a new resource management scheme for grid computing environment considering the complex real life grid. In this chapter a detailed description of the system architecture of our resource management scheme is presented with an illustrative example. The proposed data structure is also described with example. But at the beginning we will describe the problem statement in the next section.

3.1 Optimization Problem for the Resource Management Scheme

Grid applications can be broken down into number of jobs. It is the responsibility of the job broker to break down the jobs of the applications. Each job of an application requires some grid resources to perform their operations. The participating computing nodes of grid usually provide the required resources of any job. The computing nodes will provide their resource with the exchange of cost. So each of the resources of any computing nodes will have unit cost associated with it.

Let there be n applications in the grid termed as $A_1, A_2 \cdots A_i \cdots A_n$. Each application consists of a number of jobs. The number of jobs for each application are by $C_{A_1}, C_{A_2} \cdots C_{A_i} \cdots C_{A_n}$. Consider jobs of application A_i are $J_1, J_2 \cdots J_j \cdots J_{C_{A_i}}$. The participating computing nodes of the grids are $N_1, N_2 \cdots N_k \cdots N_l$ and the resource provided by them are $R_1, R_2 \cdots R_r \cdots R_m$. We assume that all the participating nodes will provide all the grid resources. To make the problem description simple, let us consider that the job J_j of the application A_i requires W amount of the resource R_r. The available amount of the resource R_r in the nodes $N_1, N_2 \cdots N_l$ are $w_1, w_2 \cdots w_l$

and the corresponding unit cost are $c_1, c_2 \cdots c_l$. It is not always possible for a single computing node to completely serve a resource request. Most of the cases a number of computing nodes jointly serve a resource request. The serving amount from the nodes are assumed as $s_1, s_2 \cdots s_l$. If a particular node N_σ does not serve the job then the serving amount $s_\sigma = 0$. Now the total cost to serve a resource request is the sum of all individual computing nodes' service cost for their resource. So the total cost of the request will be $\sum_{k=1}^{l} c_k s_k$. The constraints need to be satisfied are as follows:

1. $\sum_{k=1}^{l} s_i = W$, i.e., a particular job gets exactly W amount of resource from the grid.

2. $\sum_{k=1}^{l} w_k > W$, i.e., there is available resource in the grid for a job.

Now the objective is to minimize the total cost $\sum_{k=1}^{l} c_k s_k$ to serve a resource request for a job of an application. Besides the objective of minimizing the cost it is also expected to reduce the number of participating nodes to deliver resource for a particular job. This will reduce the bottleneck for remote communication to the participating nodes. This additional objective can be formulated as follows:

$$\text{minimize } \sum_{k=1}^{l} f(s_k) \text{ where}$$

$$f(s_k) = \begin{cases} 1 & \text{if } s_k > 0 \\ 0 & \text{if } s_k = 0 \end{cases}$$

Here $f(s_k)$ is a boolean function indicating the presence of a node in serving a job.

3.2 Resource Management Scheme

The overall system architecture of our proposed resource management scheme is shown in Figure 3.1. The components of the system, messages and their sequences to run the system have already been described by the caption of the blocks of Figure 3.1. Our proposed resource management scheme consists of the following phases:

3.2.1 Start Phase

The resource management will be controlled and coordinated by a set of computing nodes (computer) termed as Principal Resource Manager (PRM). The PRMs will be selected according to

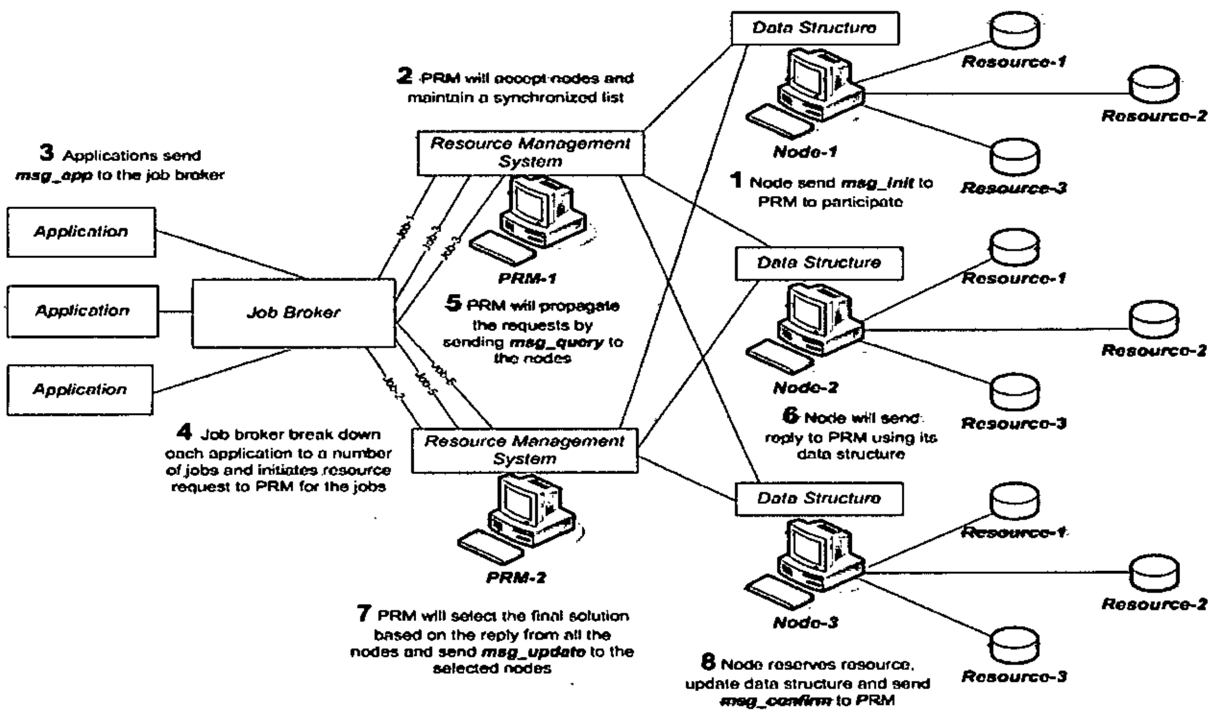
Data Structure
Resource-1
Resource-2
Resource-3
Node-1
1 Node send msg_init to PRM to participate
2 PRM will accept nodes and maintain a synchronized list
3 Applications send msg_app to the job broker
Resource Management System
Application
Application
Application
Job Broker
PRM-1
5 PRM will propagate the requests by sending msg_query to the nodes
Data Structure
Resource-1
Resource-2
Resource-3
Node-2
6 Node will send reply to PRM using its data structure
4 Job broker break down each application to a number of jobs and initiates resource request to PRM for the jobs
Resource Management System
Data Structure
Resource-1
Resource-2
Resource-3
PRM-2
Node-3
7 PRM will select the final solution based on the reply from all the nodes and send msg_update to the selected nodes
8 Node reserves resource, update data structure and send msg_confirm to PRM

Figure 3.1: Architecture of our proposed system

the grid administrators decision. The PRM will be given a list of resources by the administrators that can be provided by the participating nodes of the grid environment. Each resource will be given a unique id named $ResourceId$ for grid environment. This is known as the Start phase.

3.2.2 Initialization Phase

When a node wants to participate in the grid it will send a message named msg_init to any one of the $PRMs$. Each PRM has a list of participating computing nodes that will be synchronized amongst all the PRM. The PRM will accept the node and add it to the list. The node will be given a unique id named $NodeId$ that will help to identify it in the grid environment. Each participating node will have a list of resources to provide service to the grid environment. This list will be a subset of the list maintained by the PRM. This is known as the Initialization phase. The given $NodeId$ for any node is the same as the index of the node in the list maintained by PRM. Thus we can find the reference of any node through any of the PRMs in constant time. Each resource of a node will be given a $ResourceId$ which is also the same as the index of the resource in the node's resource list. Thus we can find the reference of any resource of a given node in constant time. Each node can be considered as its own resource manager (RM). Any participating computing node can be selected as PRM.

3.2.3 Request Phase

Any application on the grid can be broken down into a number of jobs. An application sends a message named msg_app to job broker so that job broker can break it down into a number of different jobs. The jobs usually request resources from the grid. The request will be initiated by the job broker. Job broker will forward the request to any one of the PRMs so that the request processing is distributed among the PRMs. The request mainly contains resource identifier, starting time, and ending time. The PRM will propagate the request to the all the participating nodes. This is known as Request phase. Any single job can issue request for multiple resources. Then the job broker can forward request for each resource to different PRM. So multiple PRMs can process request for a single job.

3.2.4 Search Phase

PRM will forward the request to each participating nodes by sending messages to the nodes. The messages sent to the nodes from PRM are generally named as msg_query. This is a parameterized message and based on the parameter a node replies with specific resource information, unit cost associated with a specific resource and available amount of specific resource in a given time frame. Some of the participating nodes may not provide the searched resource and they will be out of the search immediately. They will ignore the message. The nodes that provide the searched resource will receive the message. For each resource of each computing node, there will be an appropriate data structure to hold the information of used and available amount of resources in specific time frames. Then queries and corresponding updates will be carried out by the node in its own data structures. The data structure maintained by each participating node will not be replicated or copied to the PRM. The PRM will have only the reference of the nodes in the list and through that reference it can virtually have knowledge of the nodes' data structure. In this way multiple PRMs can have access to the most recent state of all of the resources without any space overhead. This is known as Search phase. This is the main computation phase of our proposed resource management scheme.

3.2.5 Reply Phase

After searching, the results will be returned to the PRM by the nodes. Each request will contain a $request\_time$ associated with it. PRM will wait for the result for a specified threshold amount of time from the *request_time*. Job broker will also wait for the replies from PRMs for a specified threshold amount of time from the *request_time* for the job that requires multiple resources. The result contains the notification whether the specific request can be served by this node or not. After getting the search result from all the nodes PRM will have a list of candidate nodes to serve the resource request. Each candidate node will have a unit cost associated with the resource it will provide. So PRM needs to identity the set of nodes that needs to be selected as the provider to minimize the total cost to serve the request. This problem can be mapped to well known fractional knapsack problem. The application of fractional knapsack problem in resource management is a new and novel idea for grid which we introduce here to guarantee cost minimization. This is described in details in the following section. The set of selected nodes will ultimately serve the request. This is known as Reply phase.

3.2.6 Reservation Phase

Once PRM have the list of selected nodes, then it sends a message named *msg_update* to each of the selected node to reserve required resource and update the data structure of the node. This phase is known as Reservation phase. Upon receiving the message the node tries to update its data structure. If the update is successful then it will send a confirmation message named *msg_confirm* to the PRM in reply. But sometimes updating may fail due to unavailability of resource. The PRMs work in distributed manner. So it is possible that a node's resource is not available during the Reservation phase though resource is found available during the Search phase. In the Search phase only available resource is searched but no update is made. So it may happen that another job acquires this specific resource of the node through any other PRM. That is why a confirmation ensures successful resource reservation.

3.3 Fractional Knapsack Problem

In this section we review the fractional knapsack problem and present a greedy algorithm for the fractional knapsack problem.

Consider the following scenario. A thief enters a store and sees the following items:

His Knapsack holds 4 Kgs. What should he steal to maximize profit?

Table 3.1: Store items

Item	Cost in Tk	Weight in Kg
A	100	2
B	10	2
C	120	3

According to the Fractional Knapsack Problem, Thief can take a fraction of an item. So the solution = 2 $Kgs\ of\ item$ A + 2 $Kgs\ of\ item$ C = 100 Tk + 80 Tk = 180 Tk.

3.3.1 Greedy Solution for Fractional Knapsack Problem

Given a set of item I

Table 3.2: List of items - I

weight	w_1	w_2	$\dots$	w_n
cost	c_1	c_2	$\dots$	c_n

Let P be the problem of selecting items from I, with weight limit K , such that the resulting cost (value) is maximum.

1. Calculate $value\ v_i = \frac{c_i}{w_i}\ for\ i = 1, 2 \dots n$
2. Sort the items by decreasing v_i. Let the sorted item sequence be 1,2,3 .. n and the corresponding $value$ and $weight$ be v_i and w_i respectively.
3. Let k be the current weight limit (Initially, $k = K$). In each iteration, we choose Item i from the head of the unselected list.
    - If $k \geq w_i$, we take item i and $k = k - w_i$ and then consider the next unselected item.
    - If $k < w_i$, we take a fraction f of Item i, i.e., we only take $f = \frac{k}{w_i}$ (< 1) of item i, which weights exactly k and the algorithm is terminated.

Knapsack problem and most of its variants are NP-Hard problem and the greedy solution to these problem leads to a suboptimal or approximate solution. But the greedy solution provides the optimal solution to the fractional knapsack problem [16].

We map the optimization problem for the resource management scheme to fractional knapsack problem. A resource of a node can be considered as an *item* and its associated unit cost can be considered as *value*. We need to sort the resources of the nodes by increasing unit cost as we need to minimize the total cost.

3.4 Data Structure to Manage Resources

For each resource of each computing node, there must be an appropriate data structure to hold the information of used and available amount of resources in specific time frames. Each element of the data structure will represent the (starting time, ending time, available amount) information for any resource. For example, we assume a node having *NodeId* N_1 contains the resource having *ResourceId* R_1 with total amount of 100 units. Table 3.3 shows the available unit of resource R_1 of node N_1 in different time periods that need to be maintained by node's data structure.

Table 3.3: Data structure contents of a node

Starting Time	**Ending Time**	**Available amount of resources**
0	25	100
25	50	96
50	150	50
150	∞	100

So we can observe that for the resource management scheme we need to design data structures to perform the following tasks:

- To specify the request (i.e. starting time and ending time of the request) and the amount of specific resource required to complete the request.
- To keep track of available amount of a specific resource at a particular time interval (presented in Table 3.3)
- To store the list of requests with different states (accepted/rejected)

We know that the tree data structures are very much efficient for searching, inserting and deleting of elements. Segment tree is a balanced binary tree data structure that is used for tracking intervals.

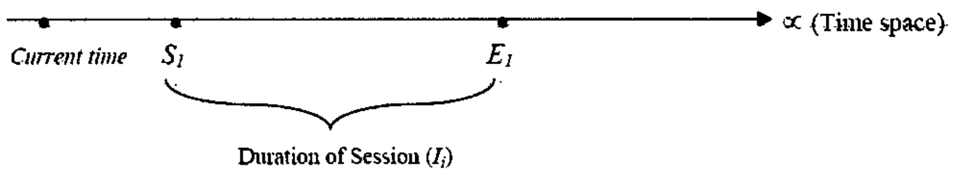

Figure 3.2: Time space segmentation of segment tree

3.4.1 Segments of the Session

Available resource amount at a particular duration specified by a starting and ending point depends on the accepted requests on that particular duration. We can think of a total time duration starting from the current time to ∞. The starting and ending points of a request may divide the total time durations into several parts. Each of these parts is called a segment. A request may consist of one or more segments.

The available amount of a resource is different at different time as the admitted requests are scheduled to start or finish at different times. The available amount of a resource changes with the introduction of a new request at any time. When new requests are accepted the length of the segment are updated and some new segments may be created. Accordingly the available amount of resources will also be updated.

Let there be a session s with starting time S_i, ending time E_i (S_i<E_i) and resource consumed b_i. Total Bandwidth of the system is $B(B \geq \Sigma b_i)$. The sessions may be described as s_1 (S_1, E_1, b_1); s_2 (S_2, E_2, b_2) ... s_n (S_n, E_n, b_n) etc.

If only one session s_1 (S_1, E_1, b_1) is admitted in the server then the time segments will be (*Current time*, S_1, B), (S_1, E_1, B-b_1) and (E_1, ∞, B).

Now, let us consider a new session request s'_1 is submitted to the server shown in Figure 3.3. The starting time, ending time, and resource consumed of the new session s'_1 are described by (S'_1, E'_1, b'_1) respectively where $S_1 < S'_1 < E_1$ and $E_1 < E'_1 < \infty$.

Then the segment will be updated as (*Current time*, S_1, B), (S_1, S'_1, B-b_1), (S'_1, E_1, B-b_1-b'_1), (E_1, E'_1, B-b'_1) and (E'_1, ∞, B).

3.4.2 Segment Tree to Incorporate Grid Job Sessions

The segment tree structure, introduced by Bentley [4], is a balanced binary tree data structure that is used to store segments or intervals. We can map reservation supported resource management

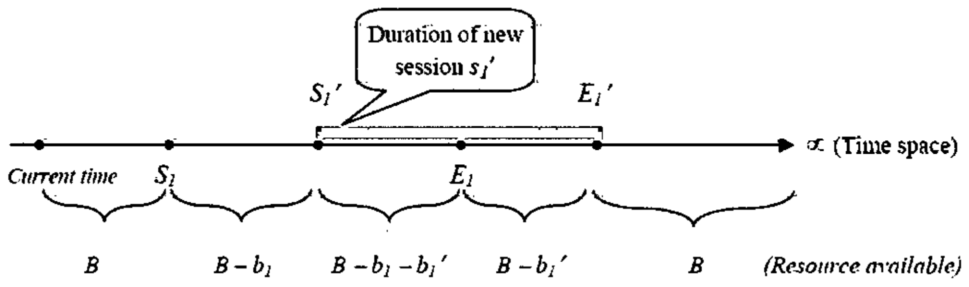

Figure 3.3: Modification of segments

problem of our system to the segment tree with a little modification. The s and t, with $s < t$, of the segment tree $V(s,t)$ can be mapped into the starting time and ending time of a session where starting time < ending time. We modify the traditional segment tree. We add a field (available resource amount of a segment) to each leaf node. Thus the contents of each leaf node in the segment tree with this modification are as follows:

- Starting time
- Ending time
- Amount of specific resource available (between the starting time and ending time)

Leaves of the segment tree contain all the segments and the available resource amount. The internal node of the tree contains only the interval of its child node.

Let us consider a scenario of a segment tree where we assume current time is zero 0 and total available amount of resource is B. A session of starting time 50, ending time 150 and resource request 50 is admitted. Root contains the segment $(0,\infty)$. Root's left node contains $(0,50,B)$ and root's right node contains $(50,\infty)$ segment. Root's right left leaf node contains $(50,150,B-50)$ segment and root's right leaf node contains $(150,\infty,B)$.

Now let us consider another case that a second session of starting time 25, ending time 50 and resource request 4 is admitted. In this case the starting time and ending time of the session lie on one node, Hence the updated segment tree will be as follows-

Now consider a third session of starting time 30, ending time 180 and resource request 5 is admitted. Here the starting time and ending time of the session lie on different node. Now the updated segment tree will be as follows.

If a session arrived with starting point and ending point coincided with the existing segment's starting point and ending point (that means starting point is coincided with the segment's starting

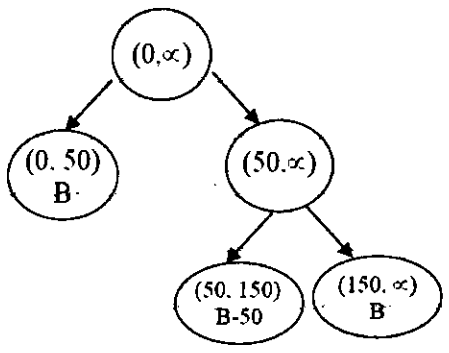

Figure 3.4: Construction of a segment tree

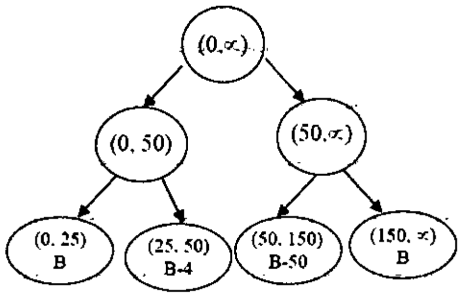

Figure 3.5: Inserting a session into the segment tree

point and ending point is coincided with the segment's ending point) then the height of the tree will not be increased. The corresponding leaf nodes' available amount will be updated only in this case.

3.5 An Illustrative Example

Consider a grid environment where two Principal Resource Managers (PRM) are working named PRM_1 and PRM_2. The provided resource list is $R = \{R_1, R_2, R_3 \ldots R_n\}$. The participating node list is $N = \{N_1, N_2, N_3 \ldots N_n\}$. Figure 3.7 depicts the overall scenario of the system:

Here we can see the resource provided by a specific node N_1. For each resource of N_1 there is a segment tree to maintain the available amount of resources in a specific time frame.

Now consider that Application A_1 contains three jobs termed J_1, J_2 and J_3. J_1 requires 25 units of resource R_1 in time frame of $(25, 45)$. J_1 will send the request to the PRM. PRM have a synchronized list of participating nodes. PRM will forward the query to the participating nodes. Figure 3.8 depicts the scenario of the nodes and corresponding resource R_1. Here separate time intervals and corresponding available amount of resource is shown with the leaf nodes.

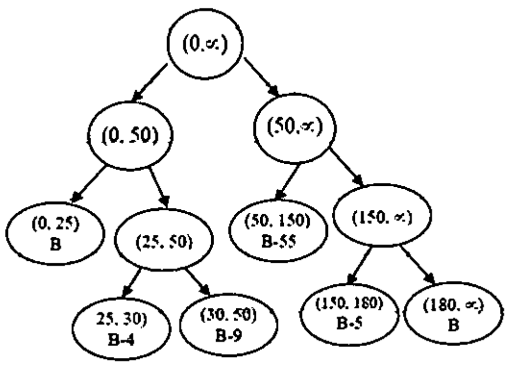

Figure 3.6: Updating a segment tree

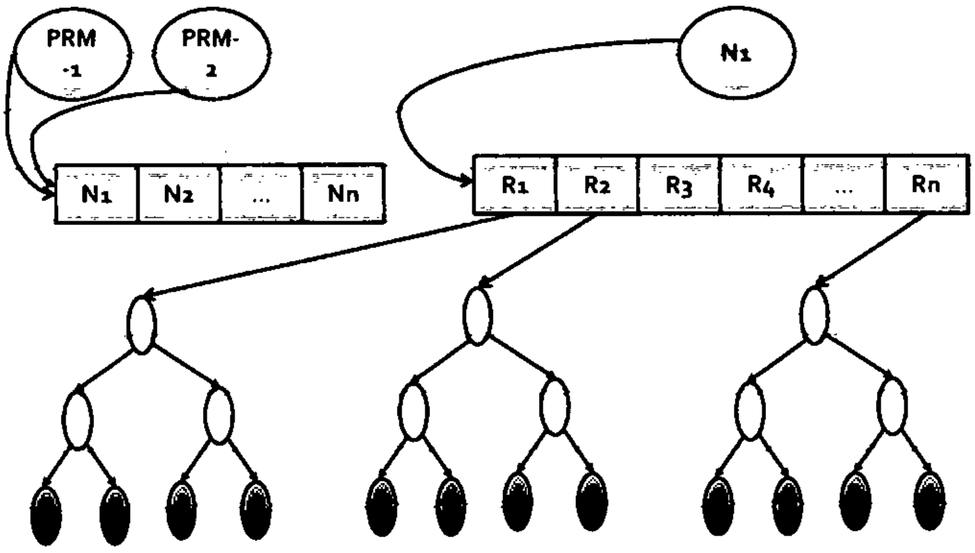

Figure 3.7: Overall system scenario

Here we can see Node N_1, N_2, N_3 and N_4 are providing resource R_1. The corresponding data structure is also shown in the figure. Here the query will be $(R_1, 25, 45)$. If this query is passed to each node data structure, the reply is listed in Table 3.4.

After the search is completed, PRM will receive the above candidate list to serve the request for R_1. If the above candidate list with unit costs are supplied to the fractional knapsack problem then the solution can be presented in Table 3.5

So an add request will be sent to N_3 and N_4 and their corresponding segment tree will be updated as shown in Figure 3.9. The updated resource usages are shown in black shades. This concludes the resource reservation to the job. The application will then start running according to its starting time using these reserved resources.

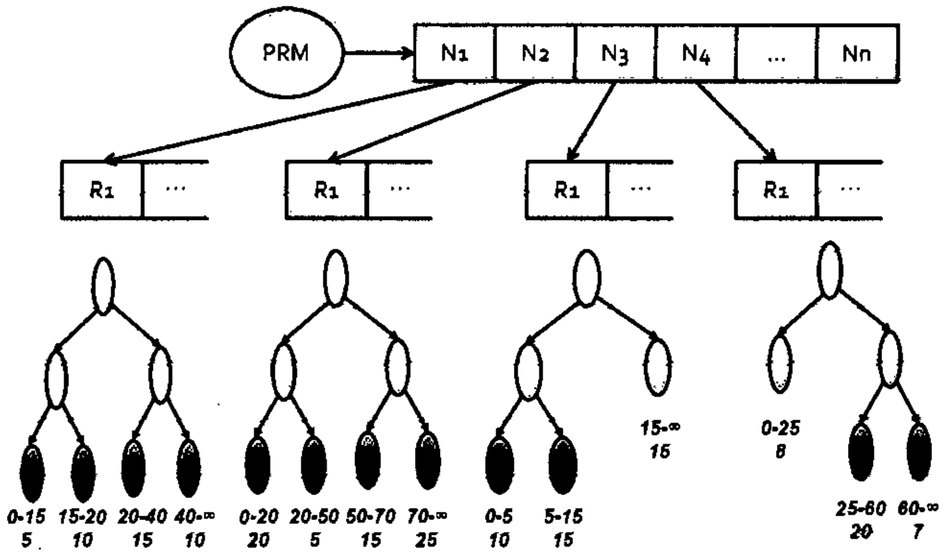

Figure 3.8: Node and its data structure

Table 3.4: Candidate nodes and available amount

Node	Available Amount	Description	Unit Cost
N_1	10	minimum amount of time frame (20, 40) and (40, ∞))	4.0
N_2	5	amount of time frame (20, 50)	4.25
N_3	15	amount of time frame (15, ∞)	3.75
N_4	20	amount of time frame (25, 60)	3.5

Table 3.5: Selected nodes from the candidate list

Node	Amount	Cost
N_4	20	$20 \times 3.5 = 70$
N_3	5	$5 \times 3.75 = 18.75$
	Minimum Total Cost	88.75

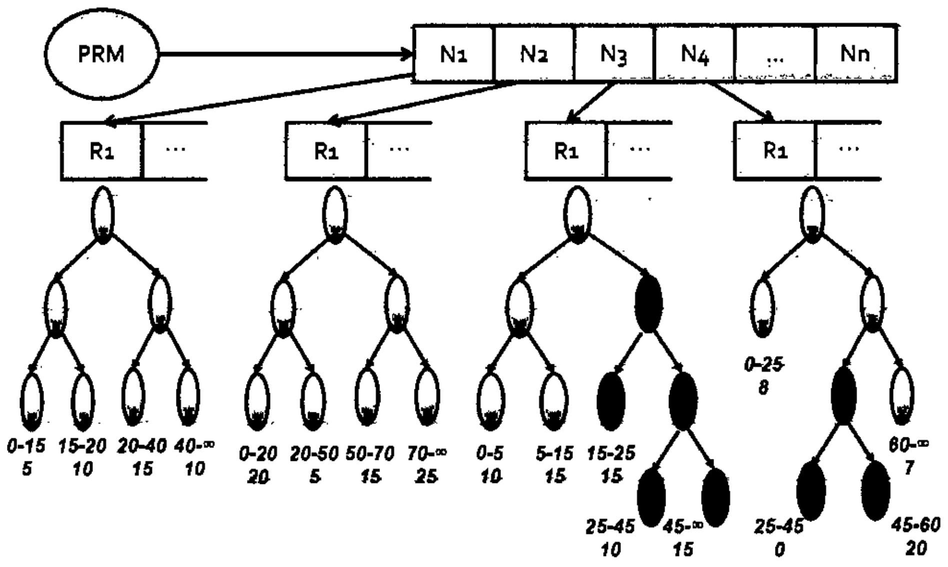

Figure 3.9: Updated system after serving a request

3.6 Major Algorithms and Their Descriptions

3.6.1 Algorithms Related to Data Structure

The list of algorithms related to our data structure are given below with the operations they perform. For details please refer to *appendix D.*

1. *CreateNode(int start, int end)* - Create a single node of segment tree.
2. *CreateSegment(Node n, int pt)* - Create a segment into segment tree to add a resource request.
3. *DeleteSegment(Node n, int start, int end)* - Delete a segment from the segment tree to delete an accepted resource request after its successful completion.
4. *UpdateAmount(Node n)* - Update the available amount of specific resource in the segment tree after adding or deleting of a resource request.
5. *FindAvailable(int start, int end)* - Find the available amount of specific resource in a time frame specified by (start, end) from the segment tree.
6. *AddRequest(Node n, int start, int end, double amount)* - Add a request to the request list of the segment tree.
7. *DeleteRequest(Node n, int start, int end, double amount)* - Delete a request from the request list of the segment tree.

3.6.2 Algorithms Related to Resource Management Scheme

Algorithm 1: GetSelectedNodesFromCandidates(List<Node> candidates, Request request)

```
   Output: Retrieve the selected list of node from the candidate list and send them update
           request to optimize the overall cost for a resource request.
1  $selectedList \leftarrow empty$
2  $m \leftarrow request.amount$
3  $threshold \leftarrow 60\ seconds$
4  while $current\_time \le request.request\_time + threshold$ do
       // Start of Search Phase
       $i \leftarrow GetNextNodeForMinimumCost(candidates, request)$   // Finding node
       with minimum cost
5      $Node\ n \leftarrow candidates[i]$
6      $available \leftarrow$ send_message($msg\_query, AVAILABLE, n.nodeId, request$)
       // Finding the amount of available resource
       // End of Search Phase
7      if $available < m$ AND $available > 0$ then          // Begin of Reply Phase
8          $Reply\ reply(request, n.nodeId, available)$     // Resource available
9          $confirm \leftarrow$ send_message($msg\_update, reply$)
10         if $confirm > 0$ then
11             $m \leftarrow m - confirm$                    // Resource reserved
12         endif
13     endif
14     else if $available \ge m$ then
15         $Reply\ lastreply(request, n.nodeId, m)$         // Resource available
16         $lastconfirm \leftarrow$ send_message($msg\_update, lastreply$)
17         if $lastconfirm > 0$ then
18             $m \leftarrow m - lastconfirm$               // Resource reserved
19             if $m = 0$ then
20                 return                      // Reservation Phase completed
21             endif
22         endif
23     endif
24 endw
```

Algorithm 2: GetNextNodeForMinimumCost(List<Node> candidates, Request request)

Output: Find out the next node to achieve cost optimization while replying a resource request.

```
$min \leftarrow 1$
$size \leftarrow candidates.size$
**for** $i \leftarrow 1$ **to** $size$ **do**
    ~~$Node\ n_i \leftarrow candidates[i]$~~
    $Node\ n_m \leftarrow candidates[min]$
    $r_i cost \leftarrow$ **send_message**$(msg\_query, COST, n_i.nodeId, request)$
    $r_m cost \leftarrow$ **send_message**$(msg\_query, COST, n_m.nodeId, request)$
    **if** $r_i cost < r_m cost$ **then**
        $min \leftarrow i$
    **endif**
**endfor**
**return** $min$
```

Algorithm 3: receive_message(msg_type,msg_parameters)

```
Output: Receive the messages sent to nodes and performs the required operation.
1  $nodeId \leftarrow id\ of\ the\ receiver\ node$
2  if msg_type $= msg\_query$ then
3      $msg\_query\_id \leftarrow retrieve\ the\ id\ from$ msg_parameters
4      $request \leftarrow retrieve\ the\ request\ from$ msg_parameters
5      if $msg\_query\_id = AVAILABLE$ then                // Query for available
       resource
6          $resourceId \leftarrow request.resourceId$
7          $startTime \leftarrow request.startTime,\ endTime \leftarrow request.endTime$
8          return $node.resourceList[resourceId].segmentTree.$
9          $findAvailable(startTime, endTime)$
10     endif
11     else if $msg\_query\_id = COST$ then       // Query for cost of resource
12         $resourceId \leftarrow request.resourceId$
13         return $node.resourceList[resourceId].cost$
14     endif
15 endif
16 else if msg_type $= msg\_update$ then                        // Update request
17     $reply \leftarrow retrieve\ the\ reply\ from$ msg_parameters
18     $resourceId \leftarrow reply.resourceId$
19     $startTime \leftarrow reply.startTime,\ endTime \leftarrow reply.endTime$
20     $amount \leftarrow reply.amount$
21     $actual \leftarrow node.resourceList[resourceId].segmentTree.addRequest$
22     $(startTime, endTime, amount)$
23     if $actual > 0$ then
24         return $actual$                                  // Update successful
25     endif
26     else
27         return 0  // Update not successful as there is no available
           resource
28     endif
29 endif
```

3.6.3 Message Complexity Analysis

The procedure $GetNextNodeForMinimumCost$ requires $2n$ messages where n is the total number of candidate nodes in the grid. For the simplicity of the complexity analysis let us assume that total h nodes among the n candidate nodes of the grid will serve the resource request where $h \approx \frac{n}{2}$. So the average message complexity to reserve a particular resource using $GetSelectedNodesFromCandidates$ procedure will be $h(2n+2) = 2h(n+1) \approx 2\times\frac{n}{2}\times(n+1) \approx n(n+1) \approx O(n^2)$. We assume that a job generally requires k distinct resources. So the average message complexity to reserve all the resource requests of a job is $O(kn^2)$.

PRM will maintain a synchronized list of all participating nodes. It is possible to store the unit cost associated with each resource of each node in the PRM. The space complexity will be $O(lm)$. In that case $2n$ messages will not be needed for $GetNextNodeForMinimumCost$. Then the average message complexity to reserve a particular resource using $GetSelectedNodesFromCandidates$ procedure will be $2h \approx 2 \times \frac{n}{2} \approx O(n)$. So with $O(lm)$ space complexity the average message complexity to reserve all the resource requests of a job is $O(kn)$.

Chapter 4

Simulation Results

In this chapter we present the simulation results of our proposed system. Through simulation we study the behavior of our approach and evaluate its performance based on some performance metrics. We also compare the performance of our approach to an existing system.

4.1 Modules of the Simulator

A simulator of the system is implemented using Java. The main motivations behind using Java are object oriented support and platform independency.

The simulation is run using a computer having Intel Pentium-IV Dual Core 1.60GHz processor, 2 GB of memory and Windows XP operating system.

The major modules of the simulator are described below in the following subsections.

4.1.1 Utility Classes

Utility classes created for the implementation of the simulator are listed as follows:

- *Application*: The *Application* class represents any application for grid computing environment.
- *Job*: Each application consists of a number of jobs. The *Job* class represents any job of an application.
- *Device*: The *Device* class represents the computers that participate in the grid. It can also be termed as *Node*.

- *Resource*: The *Resource* class represents the resource for any device. The resource can be processor, memory, disk etc.
- *RequiredResource*: The *RequiredResource* class represents the required resource of any job for successful completion.
- *RequiredResourceReply*: For each *RequiredResource* for a job, there will be a set of devices that will provide the resource. The *RequiredResourceReply* class represents that information.
- *Request*: The *Request* class is used to store the request in the corresponding data structure.
- *ApplicationJobResponse*: The *ApplicationJobResponse* class is used to store the responses of any resource request.

4.1.2 Data Structure

Data structures developed for simulation are listed as follows:

- *Segment Tree*: As described before, segment tree is used for the proposed system. Segment tree is developed and modified to support reservation for multiple resources.
- *Calendar Queue*: Calendar Queue is developed to build the system with which we compare our proposed system.

4.1.3 Event Simulator

A discrete time event simulator is implemented. The simulator will simulate according to the entry and exit time of a job of an application. For the main scheme of the simulator please refer to *appendix E*.

4.2 Simulation on Randomly Generated Data

To evaluate the performance of the proposed scheme for different policy rules we run the simulation on randomly generated data. The randomly generated data are given below.

- ***DeviceList*** : The list of all the participating computing nodes or devices.
- ***ResourceList*** : The list of all the resources provided by the computing nodes in the grid.

- ***DeviceResourceList*** : The list of all the participating computing nodes or devices with information of their provided resources.
- ***ApplicationJobList*** : The list of all the applications with their corresponding jobs in the grid.
- ***JobResourceList*** : The list of resource requirements of all the jobs in the grid.

Please refer to *appendix A* for the content of the randomly generated data.

4.2.1 Node Selection Rules

The following rules are considered for assigning priority in selecting the next node to serve the request.

- *Max-Res*: This rule prioritizes the nodes that have maximum available resource. In this way number of connection establishment can be reduced.
- *Min-Res*: This rule prioritizes the nodes that have minimum available resource. In this way number of connection establishment can be increased.
- *Min-Cost*: This rule prioritizes the nodes that have the cost to minimize total cost.

4.2.2 Measurement Metrics

The metrics considered for evaluation are *TotalConnection* and *TotalCost*. We have also considered total memory consumption and running time.
TotalConnection: The term *TotalConnection* means the number of nodes required to completely serve a request. The requesting node needs to connect to these nodes. That is why we termed it as *TotalConnection*.
TotalCost: The term *TotalCost* means the total cost required to completely serve a request. This is the summation of all the individual cost of different nodes that serves the request.

4.2.3 Analysis of the Result

From the result shown in Figure 4.1 and 4.2 we observe that our proposed rule Min-Cost for resource allocation always provides optimized cost. But while doing so it also guarantees that the total number of connection will be closely similar to the minimum possible provided by rule Max-Res. We use Min-Cost rule for our proposed resource management scheme.

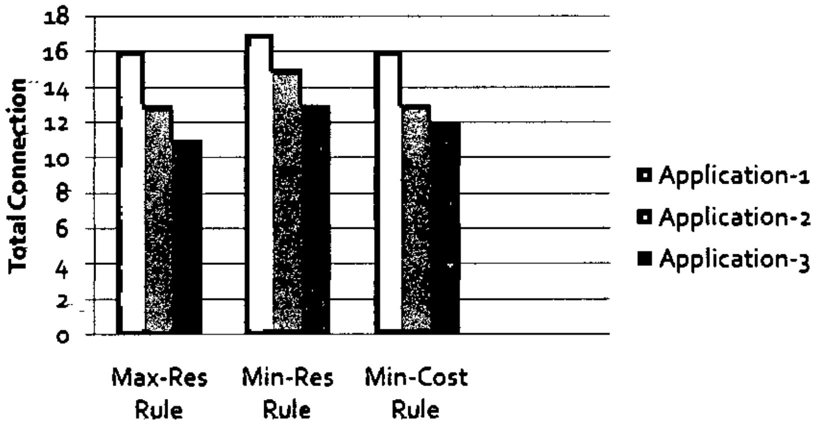

Figure 4.1: Comparison of three rules with respect to TotalConnection

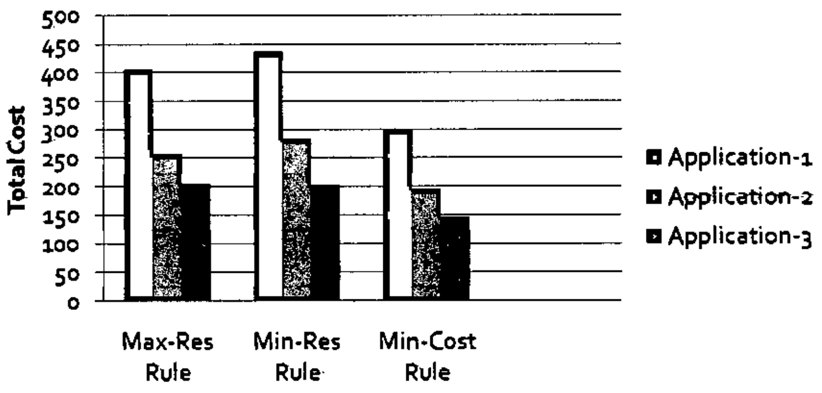

Figure 4.2: Comparison of three rules with respect to TotalCost

4.2.4 Comparison with Sulistio's Resource Management Scheme

We compare our system with an existing system for resource management in grid computing. The work done by Sulistio et al. [22] is the most appropriate to compare. This work provides a new data structure for reservation using the Calendar Queue. In Sulistio's system there is no consideration of cost so a default cost model needs to be assumed. We have implemented Sulistio's System in our simulator and simulation is run on the randomly generated data. The result is presented in Figure 4.3 and 4.4.

As we can see from Figure 4.3 and 4.4, our proposed system outperforms the system developed by Sulistio with a large margin for *TotalCost*. If we have a look at the *TotalConnection* then we see that they differ with a very small margin. We developed Sulistio's system to minimize the *TotalConnection* by giving priority to the next device to be selected according to the rule Max-Res described before. It is guaranteed that Sulistio's system's *TotalConnection* will be minimized.

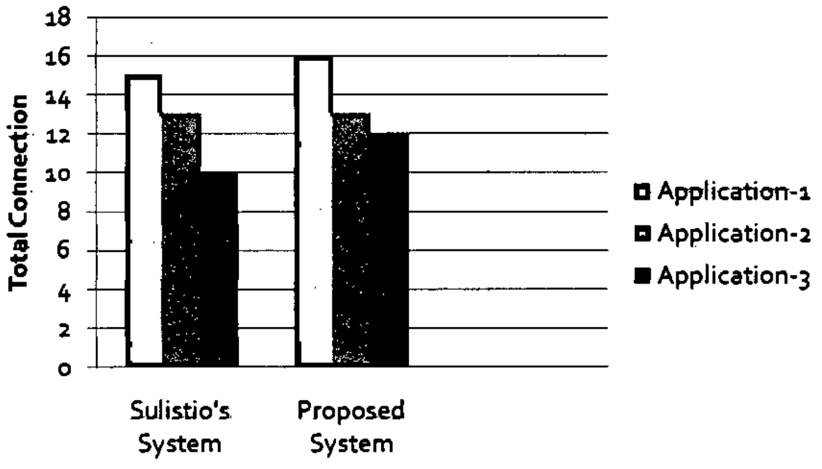

Figure 4.3: Comparison of Sulistio's system with our proposed system with respect to TotalConnection

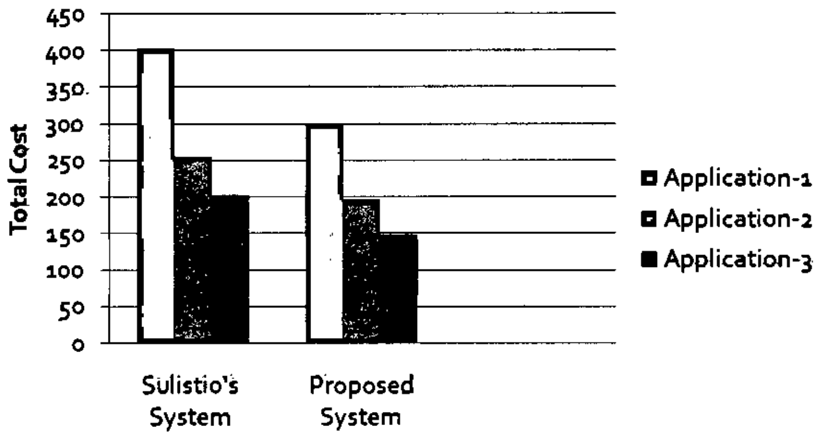

Figure 4.4: Comparison of Sulistio's system with our proposed system with respect to TotalCost

But our system's success is that it can achieve minimum cost solution to serve a grid request thus *TotalConnection* differs from the minimum one by very small values for all the applications. In some cases they are equal. These we observed for all three sample applications.

The memory consumption of our proposed system is also less than Sulistio's system. The main reason of this is to use of an efficient data structure to represent available amount of grid resources over different time intervals. Our proposed system's data structure does not depend on the size interval. But the data structure proposed by Sulistio highly depends on the size of the interval. To make the reservation effective it is required to use small sized interval. Although the initial memory requirement of the Sulistio's system may be slightly better than the newly proposed system but with the increasing number of intervals the memory requirement will increase with the advancement of

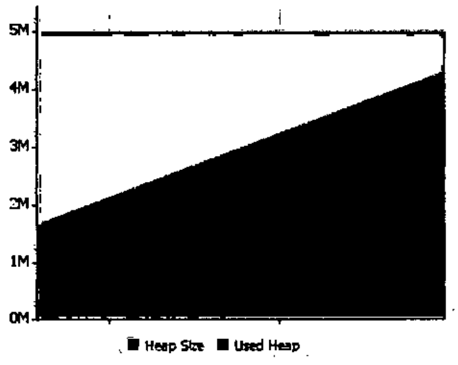

Figure 4.5: Memory consumption of Sulistio's system

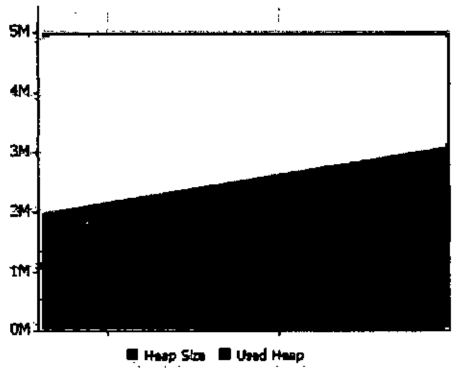

Figure 4.6: Memory consumption of our proposed system

time. Figure 4.5 and Figure 4.6 present the total memory consumption of Sulistio's system and our newly proposed system. The figures are obtained from the memory profiler of NetBeans IDE. In these figures Heap Size represents total available memory and Used Heap represents total used memory. Table 4.1 shows the actual memory consumption with respect to time.

Table 4.1: Memory Consumption

	Initial	**After 5 seconds**	**After 10 seconds**	**After 15 seconds**
Sulistios System	1.8 MB	2.8 MB	3.5 MB	4.2 MB
Proposed System	2 MB	2.3 MB	2.5 MB	3 MB

4.3 Simulation on Real Workloads

In this section we present simulation results using real workload data for grid. Parallel Workloads Archive contains an archive of information regarding the workloads on parallel machines and grids

. It contains raw workload logs from various machines around the world. The goal is to make this information freely available to the researchers interested in the evaluation of parallel and grid systems, and specifically schedulers for such systems. In addition, there is a bibliographical listing of papers related to workload issues, and a list of people working in the field [3].

4.3.1 Chosen Workloads

We choose three workloads. They are :

- ***DAS2-fs0*** : DAS-2 (Distributed ASCI Supercomputer) is a wide-area distributed cluster designed by the Advanced School for Computing and Imaging (ASCI). The DAS-2 machine is used for research on parallel and distributed computing by five Dutch universities. DAS-2 consists of five clusters, located at the five universities. The DAS-2 system is funded by NWO (the Netherlands organization for scientific research) and the participating universities [1].

- ***LPC-EGEE*** : LPC stands for 'Laboratoire de Physique Corpusculaire' (Laboratory of Corpuscular Physics) of University Blaise-Pascal, Clermont-Ferrand, France. LPC is a cluster that is part of the EGEE project (Enabling Grids for E-science in Europe). It is used mostly for biomedical and high-energy physics research.

- ***SDSC-BLUE*** : The San Diego Supercomputer Center (SDSC) is a strategic resource to science, industry and academia, offering leadership in the areas of data management, grid computing, bioinformatics, geoinformatics, high-end computing as well as other science and engineering disciplines. The mission of SDSC is to extend the reach of scientific accomplishments by providing tools such as high-performance hardware technologies, integrative software technologies and deep inter-disciplinary expertise, to the community. SDSC is a pioneer in grid computing and a leader in the national effort to build a comprehensive modern cyber infrastructure [7].

The configurations of these workloads are given in Table 4.1. The main reason behind choosing these three is that these workloads have been considered by Sulistio in their simulation. The major configuration properties of the workloads are given in Table 4.1.

Table 4.2: Chosen workloads and their configurations

Workload	Total Node	Processor	RAM per node	OS
DAS2-fs0	72	Dual 1GHz Pentium-III	1GB	RedHat Linux
LPC-EGEE	70	Dual 3GHz Pentium-IV Xeons	1GB	Scientific Linux
SDSC-BLUE	144	Eight core 375MHz Power3	4GB	RedHat Linux

4.3.2 Standard Workload Format

The standard workload format (swf) of Parallel Workloads Archive was defined in order to ease the use of workload logs. With it, programs that analyze workloads or simulate system scheduling need only be able to parse a single format, and can be applied to multiple workloads. The standard workload format abides by the following principles:

- The files are portable and easy to parse.
  - Each workload is stored in a single ASCII file.
  - Each job (or roll) is represented by a single line in the file.
  - Lines contain a predefined number of fields, which are mostly integers, separated by whitespace. Fields that are irrelevant for a specific log or model appear with a value of -1.
  - Comments are allowed, as identified by lines that start with a ';'. In particular, files are expected to start with a set of header comments that define the environment or model.
- The format is completely defined, with no scope for user extendability. Thus it is guaranteed to be able to parse any file that adheres to the standard, and multiple competing and incompatible extensions are avoided. If experience shows that important attributes have been left out, they will be included in the future by creating an updated version of the standard.

The details of the data fields of the log are presented in *appendix C*. Among the data fields the the important fields for our system are shown in Table 4.2.

The main input for our simulation is ApplicationJobList and JobResourceList that are described before. We need to write parsers for the workload to migrate the data for our system.

For each entry in the workload we insert the following entries:

Table 4.3: Important data fields of workload

Field	Description	Unit
Job Number	This is the job number starting from 1.	integer
Submit Time	The submittal time of the job.	seconds
Run Time	The clock time the job was running.	seconds
Requested Number of Processors	This is the requested number of processors.	integer
Requested Memory	This is the total requested memory.	kilobytes
Application Number	This is the application number.	integer

1. An entry to the ApplicationJobList will be by replacing (ApplicationId, JobId, Time, Type) with (ApplicationNumber, JobNumber, SubmitTime, 'S') of the workload for starting of any job.
2. An entry to the ApplicationJobList will be by replacing (ApplicationId, JobId, Time, Type) with (ApplicationNumber, JobNumber, SubmitTime + RunTime, 'E') of the workload for ending of any job.
3. An entry to the ApplicationJobList will be by replacing (ApplicationId, JobId, ResourceId, Amount) with (ApplicationNumber, JobNumber, 1, RequestedNumberofProcessors) of the workload for processor requirement of any job.
4. An entry to the ApplicationJobList will be by replacing (ApplicationId, JobId, ResourceId, Amount) with (ApplicationNumber, JobNumber, 2, RequestedMemory) of the workload for memory requirement of any job.

4.3.3 Evaluation with Respect to TotalCost and TotalConnection

We simulate our proposed system and Sulistio's system using the above mentioned three workloads. We consider 50 sample applications. The results are shown below:

In Figure 4.7 to Figure 4.9 we observe that the *TotalConnection* of Sulistio's system and our proposed system are equal for most of the applications. There are differences in *TotalConnection* for a few applications [3 applications in Figure 4.7 and 4 applications in Figure 4.9]. Here difference occurs for those applications whose jobs require huge amount of resources compared to the

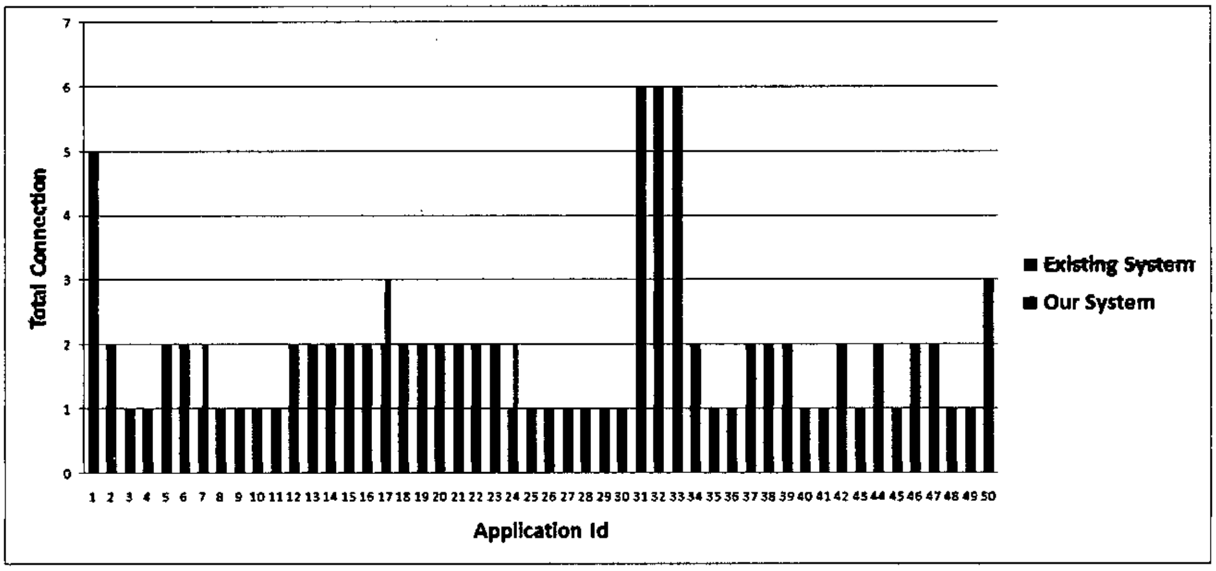

Figure 4.7: TotalConnection required for workload DAS2fs0 using Sulistio's system and our proposed system

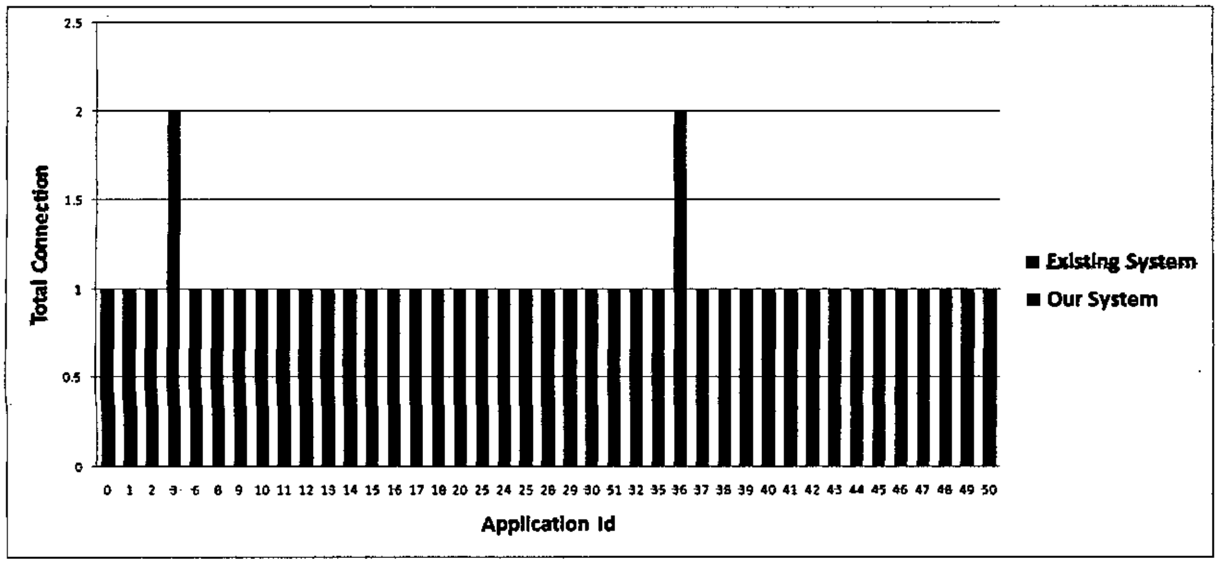

Figure 4.8: TotalConnection required for workload LPC-EGEE using Sulistio's system and our proposed system

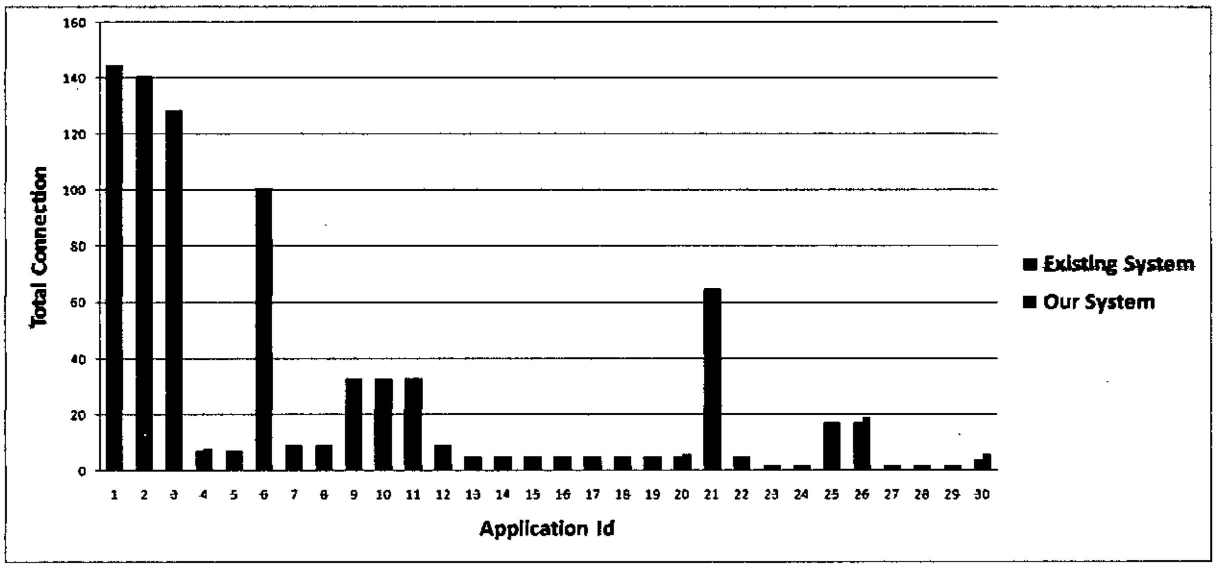

Figure 4.9: TotalConnection required for workload SDSC-BLUE using Sulistio's system and our proposed system

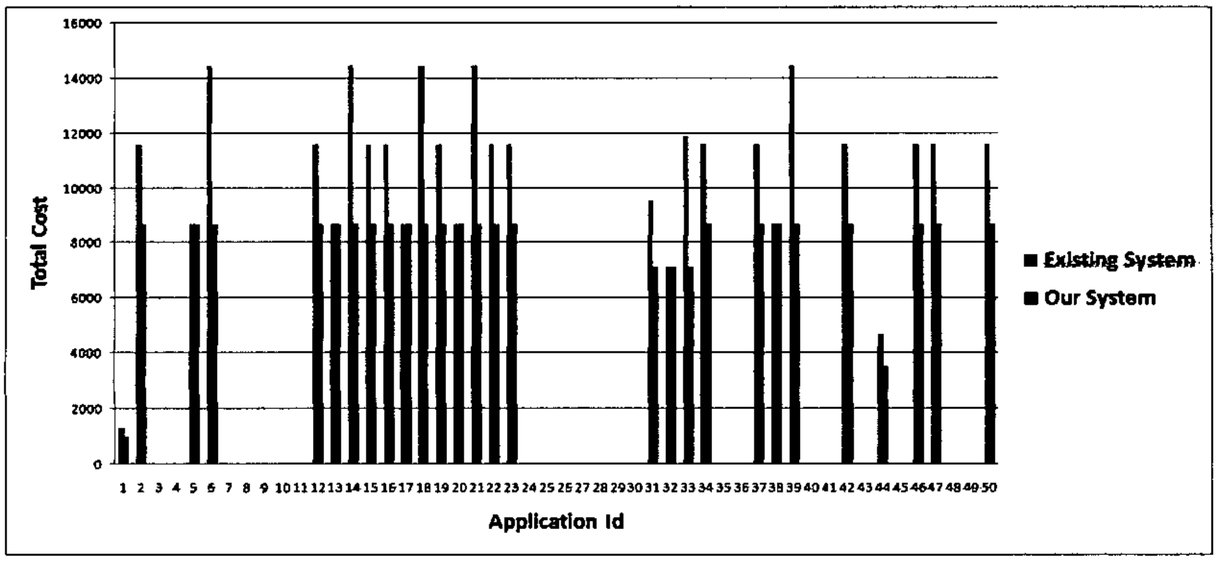

Figure 4.10: TotalCost required for workload DAS2fs0 using Sulistio's system and our proposed system

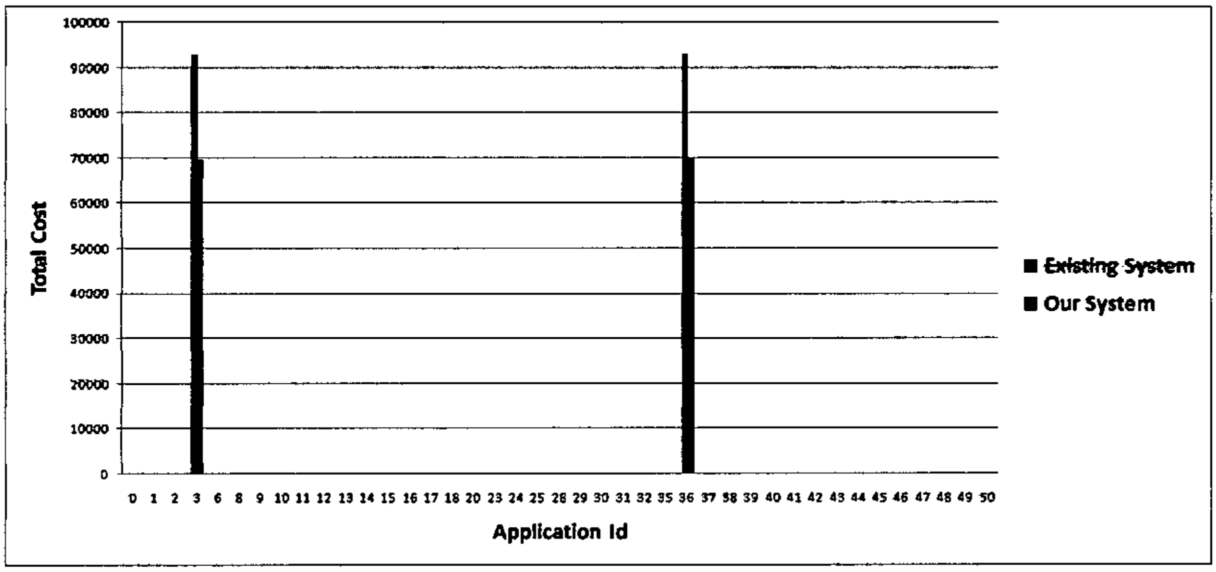

Figure 4.11: TotalCost required for workload LPC-EGEE using Sulistio's system and our proposed system

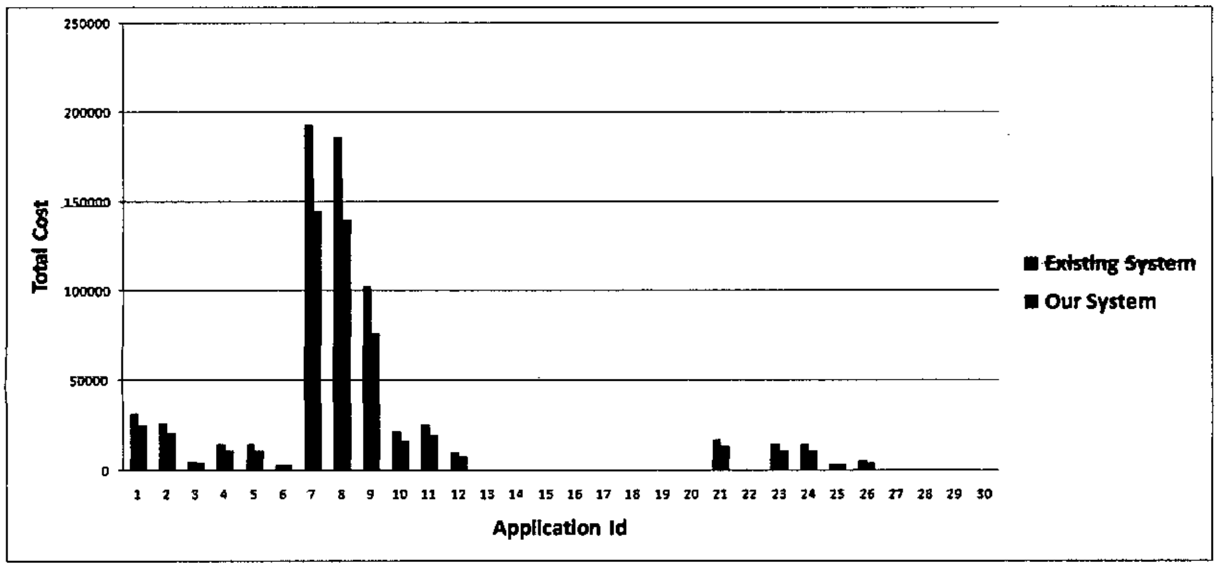

Figure 4.12: TotalCost required for workload SDSC-BLUE using Sulistio's system and our proposed system

other applications of the same grid environment.

We observe that Figure 4.9 requires larger connection compared to the other figures. It is also observed that there are applications with various connection requirements. This justifies the high capacity of SDSC-BLUE and its multipurpose use by high, medium and low profile users in terms of resource requirement.

In Figure 4.10 to Figure 4.12 we observe that the *TotalCost* of Sulistio's system is much greater than our proposed system as we expected. Some of the presented cost data seems to be zero although they are not zero cost. It is due to limitation of presentation space that the smaller cost for smaller application looks zero compared to the larger applications.

4.3.4 Analysis of the Result

TotalCost of our system will be guaranteed minimum as we use fractional knapsack to minimize the total cost. But the interesting point is the margin it varies from Sulistio's system. The *TotalCost* of our proposed system is much less than Sulistio's system for all the workloads. *TotalConnection* of our system will not be minimum because we consider minimizing the *TotalCost*. But we tried to maintain *TotalConnection* as small as possible so that the increasing *TotalConnection* will not be a bottleneck. It is observed from the presented charts that we achieve the goal to maintain the difference as minimum as possible. For almost all the workloads *TotalConnection* for Sulistio's system and our proposed system are same. This is because the nodes that provide the resources in a grid environment are mostly of same configurations and the jobs of the applications in a grid normally requires similar amount of resources. Grid applications are normally broken down into similar type of jobs by the job broker so that the application gets fair share of the resources. The details of how the job broker works is out of our research scope. In the grid environments most of the applications are similar in nature. So the job broker usually breaks down all the applications to same types of jobs where each job requires similar amount of resources. In that cases *TotalConnection* for Sulistio's system and our proposed system are the same. Sometimes there are exceptions in the workloads where a big sized application that requires huge amount of resource is broken down into a single job. This might happen due to the constraint the application cannot to broken down into small jobs. Now consider a scenario where a particular node with the comparatively higher unit cost has the highest available resource. According to our proposed algorithm this particular node will not be chosen for consuming all the resources. But the Sulistio's algorithm will consume this resource to maintain *TotalConnection* minimum. That is why we observe substantial difference in *TotalConnection* in several exceptional cases. But generally it is observed that this difference is negligible.

4.3.5 Running Time Comparison

The running time of Sulistio's system and our proposed system are given in Table 4.3. Here the running time is the total time to incorporate all the sample applications. We can observe that the running time of our proposed system is also better than Sulistio's system. The main reason behind this is the use of appropriate data structures and efficient algorithms in our proposed system.

Table 4.4: Runtime comparison

Workload	Running Time in Sulistio's System	Running Time in Our System
DAS2-fs0	56 seconds	52 seconds
LPC-EGEE	40 seconds	37 seconds
SDSC-BLUE	61 seconds	60 seconds

Chapter 5

Conclusion

In this last chapter, we draw the conclusion of our thesis by describing the major contributions made by the research works associated with the thesis followed by some directions for future research over the issue.

5.1 Major Contributions

The contributions that have been made in this thesis can be enumerated as follows:

- The main contribution of this thesis work is to design a resource management system for grid computing that is able to work in real world complex grid architecture. It works in a distributed manner and uses appropriate and efficient data structures to represent the grid architecture. It has support for both instant request acceptation/rejection and future resource reservation for any job of grid systems. It can optimize the overall performance by reclaiming unused resources after a threshold time. In brief it is a complete resource management system with reservation support for grid computing.

- The resource management system also optimizes the cost by choosing the appropriate set from a list of possible resource providers by mapping the problem to a well known cost optimization problem. Grids are used as a volunteer service now days. But with the recent improvements in grid architecture and usage, situation will not be the same. Cost for providing resource as services will play a significant role in near future when resource sharing will be popular and inevitable. But so far there is no work regarding cost optimization model for grid computing. We here introduce a novel cost optimized model for grid resources.

- We achieve cost optimization by mapping our resource allocation problem to well known fractional knapsack problem. The application of fractional knapsack problem to grid computing is a new idea. We believe that many optimization algorithms can be mapped to grid systems. There is no need to find complex algorithms for grid rather it would be better to work to fit existing algorithms to grid environments.

- We performed the detailed performance evaluation of our resource management system and compared with an existing system using real workload traces like DAS2-fs0, LPC-EGEE and SDSC-BLUE provided by Parallel Workload Archives. The performance of our system in terms of total number of connection established, total cost, memory usage and running time are analyzed and then compared with the existing one.

- After a rigorous simulation based study of various performance issues, we found that our system outperforms the existing system in most of the cases including total cost, memory usage and running time.

5.2 Future Directions of Further Research

Any research on any topic always makes a way to further research. Ours is not an exception. Resource management is one of the core parts of grid computing. It is not a new research area for grid computing but still there are lot of challenges and unsolved problems. Some of future research areas of resource management in grid are given below

- Managing resources with negotiation is one of the open issues in grid resource management. Sometimes effective negotiation for flexible quality of service (QoS) can ensure more accepted jobs in grid system with full resource utilization.

- We introduce a cost optimization model for resource management in grid computing. Future works can be done here to incorporate negotiation for cost between resource provider and resource requester (applications or jobs).

- The computing nodes that provide resource for grid also run local applications in their own operating environment. Works can be done how to optimally balance the distribution of resources for local and grid applications so that the local applications can not be affected by its services provided to the grid.

- Future works can be done on resource management by considering the topology of the grid. In that case communication bandwidth requirement and latency will affect the resource management techniques.

Bibliography

[1] The Distributed ASCI Supercomputer 2(DAS-2). http://www.cs.vu.nl/das2/.

[2] AccessGrid. http://www.accessgrid.org/.

[3] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload.

[4] Jon Bentley. Solution to klee's rectangle problems. *Techical Report, Carnegie-Mellon University, Pittsburgh*, 1975.

[5] Andrej Brodnik and Andreas Nilsson. Static data structure for discrete advance bandwidth reservations on the internet. *Computer Research Repository(CoRR)*, cs.DS/0308041, 2003.

[6] Lars-Olof Burchard. Analysis of data structures for admission control of advance reservation requests. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):413–424, 2005.

[7] San Diego Supercomputer Center. http://www.sdsc.edu/.

[8] Texas Advanced Computing Center. http://www.tacc.utexas.edu/.

[9] Liang Chen and Gagan Agrawal. A static resource allocation framework for grid-based streaming applications: Research articles. *Concurrency and Computation: Practice & Experience*, 18(6):653–666, 2006.

[10] eDiaMoND Grid Computing Project. http://www.ediamond.ox.ac.uk/.

[11] Distributed European Infrastructure for Supercomputing Applications. http://http://www.deisa.org/.

[12] Ian Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, 2002.

[13] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[14] Ian Foster, Carl Kesselman, Craig Lee, Bob Lindell, Klara Nahrstedt, and Alain Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *In Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[15] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6):37–46, 2002.

[16] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms.* Computer Science Press, 1978.

[17] Wolfgang Hoschek, Francisco Javier Jaén-Martínez, Asad Samar, Heinz Stockinger, and Kurt Stockinger. Data management in an international data grid project. In *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 77–90, London, UK, 2000. Springer-Verlag.

[18] J. Joseph, M. Ernest, and C. Fellenstein. Evolution of grid computing architecture and grid adoption models. *IBM Systems Journal*, 43(4):624–645, 2004.

[19] National Institute of Advanced Industrial Science and Technology(AIST). `http://www.aist.go.jp/index_en.html`.

[20] R. Parra-Hernandez, D. Vanderster, and N. J. Dimopoulos. Resource management and knapsack formulations on the grid. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 94–101, Washington, DC, USA, 2004. IEEE Computer Society.

[21] Warren Smith, Ian Foster, and Valerie Taylor. Scheduling with advanced reservations. In *In Proceedings of IEEE International Parallel and Distributed Processing Symposium(IPDPS)00*, pages 127–132, 2000.

[22] Anthony Sulistio, Uros Cibej, Sushil K. Prasad, and Rajkumar Buyya. Garq: An efficient scheduling data structure for advance reservations of grid resources. *International Journal of Parallel, Emergent and Distributed Systems*, 24(1):1–19, 2009.

[23] Jianbing Xing, Chanle Wu, Muliu Tao, Libing Wu, and Huyin Zhang. Flexible advance reservation for grid computing. In *Grid and Cooperative Computing(GCC)*, pages 241–248, 2004.

[24] Qing Xiong, Chanle Wu, Jianbing Xing, Libing Wu, and Huyin Zhang. A linked-list data structure for advance reservation admission control. In *In Proceedings of 3rd International Conference on Networking and Mobile Computing(ICCNMC)*, pages 901–910, 2005.

[25] Kun Yang, Xin Guo, Alex Galis, Bo Yang, and Dayou Liu. Towards efficient resource on-demand in grid computing. *ACM SIGOPS Operating Systems Review*, 37(2):37–43, 2003.

[26] Jie Yin, Yuexuan Wang, Meizhi Hu, and Cheng Wu. Predictive admission control algorithm for advance reservation in equipment grid. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*, pages 49–56, Washington, DC, USA, 2008. IEEE Computer Society.

[27] Lihua Yuan, Chen-Khong Tham, and Akkihebbal L. Ananda. A probing approach for effective distributed resource reservation. In *QoS-IP 2003: Proceedings of the Second International Workshop on Quality of Service in Multiservice IP Networks*, pages 672–688, London, UK, 2003. Springer-Verlag.

[28] Yang Zhang, Jiannong Cao, Xiaolin Chen, Sanglu Lu, and Li Xie. Threshold-based admission control for a multimedia grid: analysis and performance evaluation: Research articles. *Concurrency and Computation: Practice & Experience*, 18(14):1747–1758, 2006.

Appendix A

Randomly Generated Data for Simulation

Table A.1: DeviceList

DeviceId	Name	Description
1	D_1	Device-1
2	D_2	Device-2
3	D_3	Device-3
4	D_4	Device-4
5	D_5	Device-5
6	D_6	Device-6
7	D_7	Device-7
8	D_8	Device-8
9	D_9	Device-9
10	D_{10}	Device-10

DeviceId: a unique identifier of a device.

Name: name of the device.

Description: description of the device.

Table A.2: ResourceList

ResourceId	Name	Description

1	R_1	Resource-1
2	R_2	Resource-2
3	R_3	Resource-3
4	R_4	Resource-4
5	R_5	Resource-5
ReourceId: a unique identifier of a resource.		
Name: name of the resource.		
Description: description of the resource.		

Table A.3: DeviceResourceList

DeviceId	**ResourceId**	**Amount**	**Cost**	**DeviceId**	**ResourceId**	**Amount**	**Cost**
1	1	8	5	6	1	8	4
1	2	8	4	6	2	8	3
1	3	8	3	6	3	8	5
1	4	8	3	6	4	8	3
1	5	8	4	6	5	8	5
2	1	8	5	7	1	8	3
2	2	8	3	7	2	8	4
2	3	8	4	7	3	8	4
2	4	8	4	7	4	8	3
2	5	8	5	7	5	8	5
3	1	8	5	8	1	8	5
3	2	8	5	8	2	8	4
3	3	8	4	8	3	8	3
3	4	8	4	8	4	8	4
3	5	8	4	8	5	8	3
4	1	8	3	9	1	8	5
4	2	8	3	9	2	8	5

4	3	8	4	9	3	8	3
4	4	8	4	9	4	8	5
4	5	8	3	9	5	8	4
5	1	8	5	10	1	8	5
5	2	8	4	10	2	8	5
5	3	8	3	10	3	8	4
5	4	8	4	10	4	8	3
5	5	8	5	10	5	8	4

DeviceId: a unique identifier of a device.

ResourceId: a unique identifier of a resource.

Cost: unit cost associated with the resource.

Table A.4: ApplicationJobList

ApplicationId	JobId	JobName	Type	Time	Status
1	1	A1J1	S	30	P
1	1	A1J1	E	60	P
1	2	A1J2	S	61	P
1	2	A1J2	E	100	P
1	3	A1J3	S	101	P
1	3	A1J3	E	150	P
2	1	A2J1	S	140	P
2	1	A2J1	E	145	P
2	2	A2J2	S	146	P
2	2	A2J2	E	195	P
2	3	A2J3	S	196	P
2	3	A2J3	E	205	P
3	1	A3J1	S	30	P
3	1	A3J1	E	50	P

3	2	A3J2	S	51	P
3	2	A3J2	E	60	P
3	3	A3J3	S	61	P
3	3	A3J3	E	150	P

ApplicationId: a unique identifier of an application.

JobId: a unique identifier on a job of an application.

JobName: name of the job.

Type: type represents the entry or exit of a job. 'E' represents entry and 'S' represents exit.

Time: time represents the time measure of the entry or exit.

Status: status is used to keep track whether the specific row's information is processed or not. 'P' represents Processed and 'NP' represents Not Processed.

Table A.5: JobResourceList

ApplicationId	**JobId**	**ResourceId**	**Amount**
1	1	1	10
1	1	3	20
1	2	1	15
1	2	2	15
1	2	4	20
1	3	1	10
1	3	3	10
2	1	1	5
2	1	2	5
2	1	3	10
2	1	5	5
2	2	2	10
2	2	4	5
2	3	1	10

2	3	3	5
2	3	5	10
3	1	1	4
3	1	2	3
3	1	3	6
3	2	1	5
3	2	5	10
3	3	3	5
3	3	4	6
3	3	5	10

ApplicationId: a unique identifier of an application.

JobId: a unique identifier on a job of an application.

ResourceId: a unique identifier of a resource.

Amount: total amount of specific resource needed by the job.

Appendix B

Results of Simulation on Randomly Generated Data

Table B.1: Evaluation of Max-Res on randomly generated data

ApplicationId	TotalConnection	TotalCost
1	16	404
2	13	257
3	11	204

Table B.2: Evaluation of Min-Res on randomly generated data

ApplicationId	TotalConnection	TotalCost
1	17	436
2	15	283
3	13	202

107491

Table B.3: Evaluation of Min-Cost on randomly generated data

ApplicationId	TotalConnection	TotalCost
1	16	300
2	13	195
3	12	147

Table B.4: Evaluation of the Sulistio's system on randomly generated data

ApplicationId	TotalConnection	TotalCost
1	15	402
2	13	255
3	10	202

Table B.5: Evaluation of our proposed system on randomly generated data

ApplicationId	TotalConnection	TotalCost
1	16	300
2	13	195
3	12	147

Appendix C

Data Fields of the Workload Traces

Table C.1: Data fields of the workload

Field	**Description**	**Unit**
Job Number	Job number starting from 1.	integer
Submit Time	Submittal time the of the job.	seconds
Wait Time	Difference between the job's submit time and the time at which it actually began to run.	seconds
Run Time	Clock time the job was running.	seconds
Number of Allocated Processors	Number of processors the job uses.	integer
Average CPU Time Used	Average over all processors of the CPU time used.	seconds
Used Memory	Total used memory by a job.	kilobytes
Requested Number of Processors	Requested number of processors.	integer
Requested Time	Runtime (measured in clock) or average CPU time per processor.	seconds
Requested Memory	Total requested memory.	kilobytes
Status	Status of the job.	integer
User ID	Identifier of the user of the job.	integer
Group ID	Identifier of the group of the user.	integer
Application Number	Application number.	integer

Queue Number	Queue number.	integer
Partition Number	Partition number.	integer
Preceding Job Number	Number of a previous job in the workload, such that the current job can only start after the termination of this preceding job.	integer
Think Time from Preceding Job	Number of seconds that should elapse between the termination of the preceding job and the submittal of this one.	seconds

Appendix D

Details of the Algorithms Related to the Data Structure

Algorithm 4: CreateNode(int start, int end)

```
Output: Create a single node of segment tree
1 Create a node $n$
2 $n.startTime \leftarrow start$
3 $n.endTime \leftarrow end$
4 $n.left \leftarrow n.right \leftarrow n.parent \leftarrow null$
```

Algorithm 5: CreateSegment(Node n,int pt)

```
Output: Create a segment into segment tree to add a resource request
if $n.left = null$ and $n.right = null$ then
    if $n.startTime < pt$ and $n.endTime > pt$ then
        $n.midPoint \leftarrow pt$
        $n.left \leftarrow CreateNode(n.startTime, pt)$
        $n.right \leftarrow CreateNode(pt, n.endTime)$
    endif
    else
        do nothing
    endif
endif
else
    if $pt < n.midPoint$ then
        $n \leftarrow CreateSegment(n.left, pt)$
    endif
    else if $pt > n.midPoint$ then
        $n \leftarrow CreateSegment(n.right, pt)$
    endif
    else
        do nothing
    endif
endif
return $n$
```

Algorithm 6: DeleteSegment(Node n,int start,int end)

Output: Delete a segment from the segment tree to delete an accepted resource request after its successful completion

```
if ( start < n.startTime and end < n.endTime ) or
(start > n.startTime and end > n.endTime) then
    return
endif
if n.left = null and n.right = null then
    if start ≤ n.startTime and n.endTime ≤ end then
        update available amount
    endif
endif
else
    DeleteSegment(n.left, start, end)
    DeleteSegment(n.right, start, end)
endif
```

Algorithm 7: UpdateAmount(Node n)

Output: Update the available amount of specific resource in the segment tree after adding or deleting of a resource request

```
requestList ← list of all accepted Request
if n = null then
    return
endif
if n is a external node then
    decrease ← 0
    forall the Request r in requestList do
        if n.startTime ≥ r.startTime and n.endTime ≤ r.endTime then
            decrease ← decrease + r.amount
        endif
    endfall
    n.availableAmount ← n.availableAmount − decrease
endif
UpdateAmount(n.left)
UpdateAmount(n.right)
```

Algorithm 8: FindAvailable(int start,int end)

```
Output: Find the available amount of specific resource in a time frame specified by (start,
        end) from the segment tree
1  $min \leftarrow maximum\ integer\ value$
2  $eList \leftarrow list\ of\ all\ external\ or\ leaf\ nodes$
3  forall the Node n in eList do
4      if ( start < e.startTime and end < e.endTime and end ≤ e.startTime) or (start >
       e.startTime and end > e.endTime and start ≥ e.endTime) then
5          do nothing
6      endif
7      else
8          if e.availableAmount < min then
9              $min \leftarrow e.availableAmount$
10         endif
11     endif
12 endfall
13 return $min$
```

Algorithm 9: AddRequest(Node n, int start, int end, double amount)

```
Output: Add a request to the request list of the segment tree
1  $requestList \leftarrow list\ of\ all\ accepted\ Request$
2  $Create\ a\ new\ Request\ r\ with\ startTime \leftarrow start, endTime \leftarrow end\ and$
   $amount \leftarrow amount$
3  $CreateSegment(n, start)$
4  $CreateSegment(n, end)$
5  $UpdateAmount(n)$
6  return $actual\ amount\ of\ the\ added\ request$
```

Algorithm 10: DeleteRequest(Node n, int start, int end, double amount)

Output: Delete a request from the request list of the segment tree

```
1 $eList \leftarrow list\ of\ all\ external\ or\ leaf\ nodes$
2 forall the Node n in eList do
3     if n.startTime = start and n.endTime = end then
4         $n.availableAmount \leftarrow n.availableAmount + amount$
5         break
6     endif
7 endfall
```

Appendix E

Main Scheme of the Simulator

Algorithm 11: MainScheme()

```
Output: Coordinate resource management by taking new event from the event simulator
        and process their resource request
deviceList ← GetDeviceListWithResources()
es ← new EventSimulator()
while es.hasNext() do
    appList ← es.next()
    forall the Application a in appList do
        forall the Job j in a.jobList do
            forall the RequiredResource r in r.resourceList do
                if j.status = 'S' then
                    replyList ←
                    ResourceAllocationScheme.getAllocationCandidates(r, deviceList)
                    forall the RequiredResourceReply rr in replyList do
                        ResourceAllocationScheme.allocate(deviceList, rr)
                    endfall
                endif
                else if j.status = 'E' then
                    ResourceAllocationScheme.deallocate(deviceList, r)
                endif
            endfall
        endfall
    endfall
endw
```