M.Sc. Engg. Thesis

A Resource Reservation Scheme for Workflow-based Applications in Grid

by

Md. Abu Sayeed Mondol

Submitted to

Department of Computer Science and Engineering
in partial fulfilment of the requirements 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

April, 2011

The thesis titled ‘**A Resource Reservation Scheme for Workflow-based Applications in Grid**’, submitted by Md. Abu Sayeed Mondol, Roll No. 040805040P, Session April 2008, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfilment 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 April 26, 2011.

Board of Examiners

1. ______________________	
Dr. Md. Mostofa Akbar	Chairman
Professor	(Supervisor)
Department of Computer Science & Engineering	
BUET, Dhaka 1000	
2. ______________________	
Dr. Md. Monirul Islam	Member
Professor & Head	(Ex-officio)
Department of Computer Science & Engineering	
BUET, Dhaka 1000	
3. ______________________	
Dr. M. Kaykobad	Member
Professor	
Department of Computer Science & Engineering	
BUET, Dhaka 1000	
4. ______________________	
Dr. Reaz Ahmed	Member
Associate Professor	
Department of Computer Science & Engineering	
BUET, Dhaka 1000	
5. ______________________	
Dr. Mohammad Nurul Huda	Member
Associate Professor	(External)
Department of Computer Science & Engineering	
United International University, Dhaka	

Candidate's Declaration

This is to certify that the work entitled 'A Resource Reservation Scheme for Workflow-based Applications in Grid' 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.

Md. Abu Sayeed Mondol
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 thank the other members of my thesis committee: Dr. Md. Monirul Islam, Dr. M. Kaykobad, Dr. Reaz Ahmed and Dr. Mohammad Nurul Huda for their valuable suggestions.

I would like to express my ever gratefulness to my parents, sisters and brothers for their continuous support. Finally, I cannot forget some of my colleagues (Rifat Shahriar, Rajkumar Das, Anindya Tahsin Prodhan and Shahriar Iqbal) for their supports. May Allah reward them all in here and hereafter.

Abstract

Grid computing, a parallel and distributed computing infrastructure via wide-area sharing of computational resources, has evolved to be a mainstream technology enabling large-scale virtual organizations. Since the main objective of grid computing is to support resource sharing within a networked infrastructure, managing resources properly in grid environment is very important. In most grid systems where submitted tasks are initially placed into a queue due to unavailablity of required resources, there is no guarantee as to when these tasks will be executed. This policy may cause problems for time-critical applications. This policy is also problematic for workflow-based applications where tasks have inter-dependencies. Using Advance Reservation (AR) in grid systems allows users to secure or guarantee resources prior to executing their jobs. The resource reservation is a scheduling process that maps tasks on the distributed resources. One of the major challenges of resource reservation for a workflow-based application is to minimize the delay of execution of the overall application. In general, the problem of mapping a set of interdependent tasks on distributed services belongs to a class of problems known as NP-Complete problems. Thus, in practice, heuristics are most often used to schedule workflow-based applications in grid environments. In this thesis some properties like slack time of tasks, critical paths etc. of a workflow-based application have been exploited to provide a resource reservation scheme that gives better results and supports advance reservation. We demonstrate our claims by conducting a detailed performance evaluation and comparing with existing system for grid computing.

Chapter 1

Introduction

1.1 Grid Computing

Grid computing 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, such as processors, storage, data, applications etc., 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.

Grid is a parallel and distributed system that enables sharing, selection and aggregation of geographically distributed computing resources, owned and controlled by multiple organizations or individuals. Grid computing may be described as the virtualization and pooling of computational resources into a single set of shared services [17]. The main objective of grid computing is to utilize the unused computational resources of multiple organizations or individuals to provide cost effective computational facilities. Since the resources of the grid environment is owned and controlled by multiple organizations and individuals, there needs a centralized system to manage and coordinate the resources for the processing of the applications efficiently. Figure 1.1 compares the difference in infrastructures between traditional computing and grid computing.
Grid computing operates on three basic technology principles:

- Standardize hardware and software components to reduce incompatibility and simplify configuration and deployment

- Virtualize computing resources by pooling hardware and software into shared resources

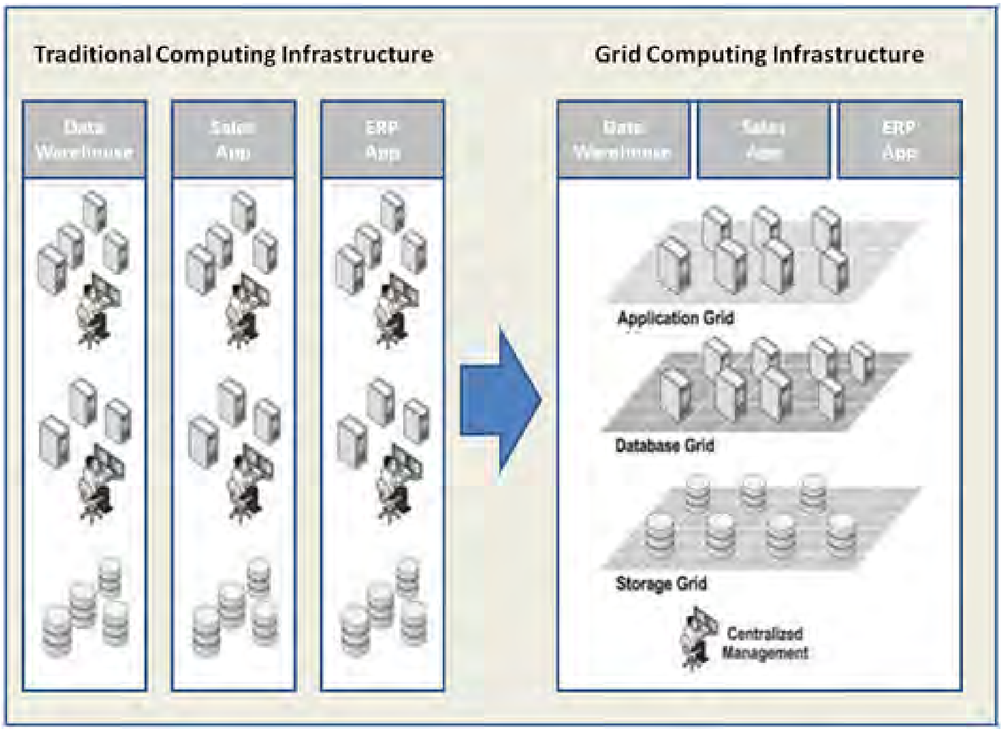

Figure 1.1: From Traditional Computing Infrastructure to Grid Computing Infrastructure

- Automate systems management, including resource provisioning and monitoring

Virtualizing computing resources means that applications are not tied to specific server, storage, network components or any other computing resources and can use any virtualized computing resource. Virtualization occurs through a sophisticated software layer that hides the underlying complexity of resources and presents a simplified, coherent interface used by applications and other computing resources.

Grid can be regarded as a technology with no boundaries. Due to the integration of a large number of computing resources, no matter what they are, in a single virtual computing environment, grid makes possible:

- 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

Like Web Technology that has brought revolution in the world of information sharing, Grid Computing is going to be the next technological revolution by integrating and making available not only information, but also computing resources such as computing power and data-storage

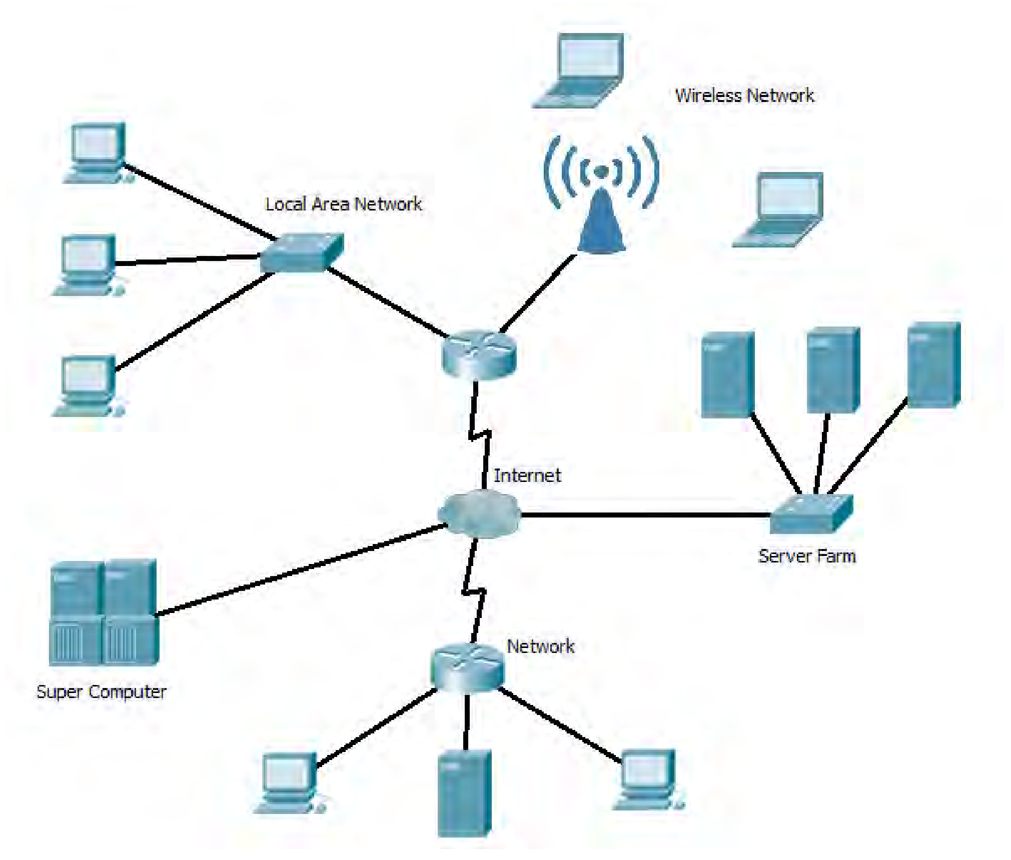

Figure 1.2: Grid : a setup with independent computing resources

capacity [21]. Figure 1.2 illustrates a way that a grid can be built by means of computing resources that are somehow interconnected by the Internet but there is no other relationship among them.

1.2 Benefits of Grid Computing

1.2.1 Exploiting underutilized resources

Grid computing is used to run applications on machines which are idle or underutilized. In most organizations as well as individual usage, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5 percent of the time over a business day [20]. Even server machines can often be relatively idle. Similarly other resources, like storage capacity, network bandwith and so on, remain underutilized. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

1.2.2 Parallel CPU capacity

The potential for massive parallel CPU capacity is one of the most common visions and attractive features of a grid. The application to run is partitioned into independently running parts that are executed in parallel on different machines in the grid. A perfectly scalable application will, for example, finish in one tenth of the time if it uses ten times the number of processors. Barriers often exist to scalability since applications may not be transformed to run in parallel on a grid.

1.2.3 Virtual resources and virtual organizations for collaboration

Grid computing provides environment and standards that enable heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of resources. The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger grid.

1.2.4 Access to additional resources

As already stated, in addition to processing and storage resources, a grid can provide access to other resources as well. The additional resources can be provided in additional numbers and/or capacity. For example, some machines may have expensive licensed software installed that users require. Users' jobs can be sent to such machines, more fully exploiting the software licenses. All of these will make the grid look like a large system with a collection of resources beyond what would be available on just one conventional machine or within an organization.

1.2.5 Resource balancing

A grid incorporates a large number of resources contributed by individual machines into a large virtual single-system. For applications that are grid-enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization. During peak load of a system, extra load can be routed to other idle resources of the grid infrastructure. As a result, the scalability of the system increases to a great extent although the computing infrastructure of the system itself remains same.

1.2.6 Reliability

In the conventional computing infrastructure, reliability is increased through the use of expensive hardware, often in a redundant way. Grid provides a complementary approach to reliability which is cost effective. For example, in critical and real-time situations, multiple copies of important jobs can be run on different machines throughout the grid. So grid can provide cost effective reliability that conventional computing infrastructure may not provide or may provide with the result of larger cost.

1.2.7 Economy

Generally idle resources are used to provide computing facilities in grid. So the cost of services in grid environment is very low and in many cases it is free. Grid computing can provide more scalable and reliable system with relatively very low cost. Sharing of costly software license and other resources also reduces the cost of operations of the users. Since the concept of grid is based on resource sharing, a user not only consumes services from grid but also can provide his resources as service to others. Thus grid brings economic benefit to its users.

1.2.8 Environment

Nowadays, the use of electric and electronic devices is increasing at a pace that has generated a lot of electronic waste. Though a little portion of the waste is being recycled, a major portion of the waste is being remained unhandled. As a result, it causes a great threat for the environment and thus for the living creatures on the earth. Since existing resources are used in grid, it will help to reduce the overall production of electronic devices that will result less pollution due to less electronic waste and less production.

1.3 Grid by Examples

In the following subsections we provide some real life examples of grid computing implementations in different areas.

1.3.1 Search for Extraterrestrial Intelligence

SETI (Search for Extraterrestrial Intelligence) is a scientific area whose goal is to detect intelligent life outside earth. One approach, known as Radio SETI, uses radio telescopes to listen for narrow-bandwidth radio signals from space. Radio SETI project analyze the data digitally. More computing power enables searches to cover greater frequency ranges with more sensitivity. Radio SETI, therefore, has an insatiable appetite for computing power. A project named the SETI@home [27] at University of California at Berkley uses a virtual supercomputer composed of large numbers of internet connected computers in form of grid.

1.3.2 Scientific Simulation

It is a grid implementation to provide the execution of complex system simulations in different areas of science. 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 around the world including National Institute of Advanced Industrial Science and Technology of Japan [25].

1.3.3 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 [14]. It is 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. .

1.3.4 Big Science

It is an implementation of a data and computational grid to accomplish the problem of storing huge quantities of data for a system which demands high storage capacity and typically requires large and parallel computational infrastructures. This type of solution has already been set in place in DEISA [15], a consortium of leading national supercomputing centers in Europe aiming to jointly build and operate a distributed terascale super computing facility.

1.3.5 E-Learning

It is a grid environment to support many of the educational and research requirements for exchanging information. The e-learning infrastructure is based on the Access Grid [3], which is an ensemble of resources including large-format displays, presentations etc.

1.4 Motivation for the Work

1.4.1 The Need for Advance Reservation

In most Grid systems, submitted jobs are initially placed into a queue if there are no available resources. Therefore there is no guarantee as to when these jobs will be executed. To address these issues and to ensure that the specified resources are available for applications when required, several researchers have proposed the need for advance reservation [23][26][29]. Common resources that can be reserved or requested are processing power, storage elements, network bandwidth or a combination of any of those. In general, reservation of the aforementioned resources can be categorized into two: immediate and advance. However, the main difference between these two reservations is the starting time. Immediate reservation acquires the resources to be utilized straight away, whereas advance reservation defers their usage later in the future. Advance reservation can be useful for several applications, as described below:

- Parallel applications, where tasks require multiple computing nodes simultaneously for execution.
- Workflow-based applications, where each task may depend on the execution of other tasks in the application. Hence, it needs to wait for all of the dependencies to be satisfied before it can be executed.
- Multimedia or soft real-time applications, such as video conferencing and player, where they need to have a certain amount of bandwidth to ensure a smooth broadcast of video and audio over the network. Therefore, any dropouts in a network transfer are not tolerable.

1.4.2 Why Workflow-based Applications?

Applications can be divided into two major categories: single task applications and multi task applications. In a single task application, the task itself is considered to be the application. On the other hand, a multi task application consists of more than one task. The multitask applications can be of two types namely Bag-of-Tasks (BoT) applications and Workflow-Based applications. BoT applications are those applications whose tasks are independent of each other and so they can run parallel. In workflow-based applications, there are dependencies between the tasks of the application. So, scheduling and managing resources for workflow-based applications in grid is relatively complex and challenging. Resource reservation system for single task or BoT applications cannot deal with workflow-based applications but those for workflow-based applications can deal with the others since they are considered to be a subset of workflow-based applications. So a system for workflow-based application is a generalized one which can deal with all type of applications.

Many of the real world problems can be best represented by a set of interdependent tasks rather than a single task or a set of independent tasks. Decomposing an application into different smaller tasks result faster execution of the application in grid. The more independent the tasks are of each other, the faster it will be, considering availability of resources. But in many cases it may not be possible to effectively decompose an application into a set of independent tasks. Decomposing such an application into sets of interdependent tasks i.e. into a workflow-based application makes it possible to run multiple tasks parallelly in grid.

1.5 Challenges

There are challenges in adopting advance reservation for workflow-based applications into Grids. The major challenges are briefly described below.

- Workflow-based application is represented by Directed Acyclic Graph (DAG). Scheduling DAGs with different node and edge weights into different resources is generally an NP-complete problem [12]. So it is computationally infeasible to derive an optimal scheduling solution for such applications. Rather heuristics are most often used to compute optimized (but not optimal) schedules. So, it is challenging to derive an algorithm for scheduling workflow-based applications in gird resources which is computationally feasible and produces better result.

- One of the major goals of the scheduling algorithms is to minimize the overall time required to complete an application. Since resources are shared and generally idle resources are used in grid, it may not be possible to schedule the applications with minimum delay. So the challenge is to reduce the total time required to complete an application.

This thesis addresses the above challenges by modeling a resource reservation scheme that exploits features like slack time of tasks and critical paths of an application.

1.6 Scope of the Work

The main focus of this thesis work is to design a resource reservation 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 provides support for both immediate and future resource reservation for any application.
- It reduces average delay to complete an application and increases the number of zero delayed applications.
- It is relatively a faster algorithm.
- It optimizes the overall performance using heuristics, effective algorithms and data structures.

The main outcome of this thesis work is a resource reservation system for workflow-based applications in grid environment with the support of immediate/future reservation, and reduced completion time with improved Quality of Service (QoS). We performed the detailed performance evaluation of our prototype and compared with an existing system. Our proposed claims are justified by the comparative analysis presented on the experimental results.

1.7 Thesis Organization

The rest of the chapters are organized as follows. Chapter 2 gives a preliminary description of some terminologies and concepts related to grid computing and workflow-based applications that might be helpful to understand the context of this thesis. The detailed description of grid resources, representation and features of workflow-based applications are presented in this chapter. The related works on resource reservation and management for grid computing is also presented in this chapter. Chapter 3, the main chapter of this dissertation, illustrates our proposed resource reservation scheme and the data structures used by this scheme. The algorithms related to the proposed resource reservation scheme with their 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 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 Resources

Grids aggregate various networked resources for solving large-scale data-intensive or compute-intensive applications [17]. Various types of resources are used to provide services in a grid infrastructure. Some resources may be used by all users of the grid, while others may have specific restrictions. This section describes the resources generally used in grid.

2.1.1 Computation

Computing cycles, provided by the processors of the machines on the grid, are the most common type resource. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage and connectivity. The computing resources may be used in many ways. The simplest way is to use it to run an existing application on an available machine on the grid rather than locally. Another approach to exploit the computing resources 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. If an application needs to be executed many times, the many different machines in the grid can be used to run the application. 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.

2.1.2 Storage

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

2.1.3 Communication

Communication technology is becoming more sophisticated and high performance communication infrastructure with more capacity is ubiquitous. Therefore, data communication capacity is emerging as an important resource to be used in grid. 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, they add to the total available bandwidth for accessing the internet.

2.1.4 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.

2.1.5 Other resources

Platforms on the grid will often have different architectures, operating systems, devices, capacities, and equipments. Each of these items represents a different kind of resource that the grid can use as criteria for assigning jobs to machines. There may be some software and applications that will run on certain type of machines or devices. Such attributes must be considered when assigning jobs to resources in the grid. Even sensors, surveillance cameras etc. can be used in grid as resource to perform some specific jobs. So, the number of resources in grid is not limited to some few although some resources are more frequently used than others.

2.2 Resource usage policies

In the early days of grid, not enough attention was paid to issues surrounding the description and enforcement of policies for the control and management of a grid. Due to tremendous development and use of grid computing, resource usage policy for grid is becoming more important. These policies define the permitted or desired usage scenarios allowed by resource providers, virtual organizations, or even the governing body for the entire 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.

Policies may be imposed for usage rate or time. Some resource provider may impose policy to use some or all of its resources up to some percentage or threshold of its capacity. This rate may vary for different time of the day, week, month or year. For example, the resources may be used upto 80% of its capacity during the weekend holidays, 60% during daily night and so on. The policies may be applicable for individual or some or all users, resources or applications of the grid.

2.3 Tasks and Applications

Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing application or task.

2.3.1 Tasks

Tasks 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. The instructions of a task are generally executed in a single machine sequentially. The grid industry uses other terms, such as transaction, work unit, job or submission, to mean the same thing as a task.

2.3.2 Applications

Usually we use the term application as the highest level of a piece of work on the grid. An application is the entity that means to be executed on the grid. Depending upon the number of tasks, applications can be divided into two major categories: single task application and multitask application.

Single task application

If the entire application is not divided into more than one task then the application is called to be a single task application. The task itself is considered to be the application. Sometimes the terms task and application are used equivalently.

Multi task application

A multi task application consists of more than one task. Applications are broken down into multiple tasks so that tasks can be executed parallel on different machines in the grid. But interdependency between the tasks may limit the degree of parallelism. The multitask applications can be of two types namely Bag-of-Tasks (BoT) application and Workflow-based (WBA) application.

Bag-of-Tasks Applications are those parallel applications whose tasks are independent of each other. Despite their simplicity, BoT applications are used in a variety of scenarios, including data mining, massive searches (such as key breaking), parameter sweeps, simulations, fractal calculations, computational biology, computer imaging and so on. Due to independence of the tasks, BoT applications can be successfully executed over widely distributed computational grids, as has been demonstrated in [4].

Workflow-based Applications are those in which there are dependencies between the tasks of the application. Although BoT applications are more suitable for computational grid, there are many applications that cannot be broken down into a set of independent tasks. For example, in astronomy, workflows with thousands of tasks are needed to identify galaxy clusters within the Sloan Digital Sky Survey [5]. Figure 2.1 shows an application with dependencies between the tasks. Each node in the graph represents a task whereas dependencies among the tasks are represented by the directed edges. Workflow-based application has been described in details in the following section.

2.4 Workflow-based Application

A Workflow-based Application (WBA) is a model of a parallel program that consists of many interdependent tasks. The dependency between the tasks limits the degree of parallelism that can be achieved by executing multiple tasks simultaneously on different Compute Nodes (CN) or

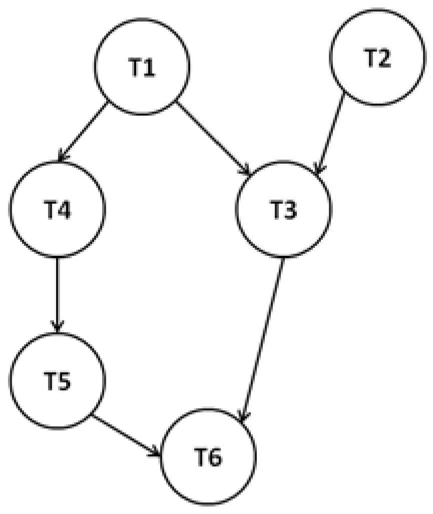

Figure 2.1: An application with dependencies between the tasks

Processing Elements (PE). Subtasks exchange data via an interconnection network. The dependencies between tasks are described by means of a Directed Acyclic Graph (DAG) called a Task Graph (TG). Executing a WBA is determined by two factors: a node weight that denotes the computation time of each subtask, and an edge weight that corresponds to the communication time between dependent subtasks [18]. Thus, to run a WBA, we need a target system that is tightly coupled by fast interconnection networks. Typically, systems such as grid computing provide an appropriate infrastructure for running parallel programs.

A Task Graph (TG) is a graph in which each node represents a task to be performed and a directed arc from a task to another indicates that the former task must be completed before the latter task begins. Each TG can be represented in a Standard Task Graph (STG) format [22], as illustrated in Figure 2.2. There is a number in the first row of the STG representing the total subtasks of the application. In this figure, a TG consists of 9 subtasks ($T0 - T8$). Then, a specification of individual subtask is described in a new row. Each row consists of three integers, denoting the subtask index or id, its node weight and number of parents. If the subtask has dependencies or parents, each of the following rows contains two numbers for a parent, specifying its parent id and the edge weight. For example, a subtask with index number 4 or $T4$ has two parents. Then, the next lines mention parents of $T4$, i.e. $T1$ and $T2$, and their edge weights of 1 and 2 time units respectively. Finally, # denotes a single line comment in the STG format. Figure 2.3 shows the structure of the TG, by using the example illustrated in Figure 2.2. In this figure, an edge weight is represented by a number next to the arrow line.

Scheduling the TG in a non-dedicated environment is a challenging job because each subtask

```
9                                   # total subtasks
0     5     0                       # subtask index, node weight, and num of parents
1     7     0
2     5     1
                  0     2           # parent index and edge weight
3     9     1
                  0     1
4     6     2
                  1     1
                  2     2
5     7     1
                  3     1
6     5     2
                  4     2
                  5     2
7     8     2
                  4     1
                  6     1
8     6     2                       # subtask index, node weight, and num of parents
                  1     2           # parent index and edge weight
                  7     2           # parent index and edge weight
```

Figure 2.2: Standard Task Graph (STG) format

needs to wait for its parent subtasks to finish executing in order to satisfy the required dependencies, as depicted in Figure 2.3. Therefore, Advance Reservation (AR) is needed to secure or guarantee resources prior to the execution of the subtasks.

2.5 Critical Path Modeling

The Critical Path(CP) is the longest-duration path in a TG. The significance of the critical path is that the activities that lie on it cannot be delayed without delaying the overall completion time of the application.

The critical path can be identified by determining the following parameters for each task or activity:

- *Earliest Start Time (EST)*: The earliest time at which the task can start given that its precedent tasks must be completed first.
- *Latest Start Time(LST)*: The latest time at which the task can be started without delaying the overall completion time of the application.
- *Earliest Finish Time(EFT)*: The earliest start time for the task plus the time required to complete the task.

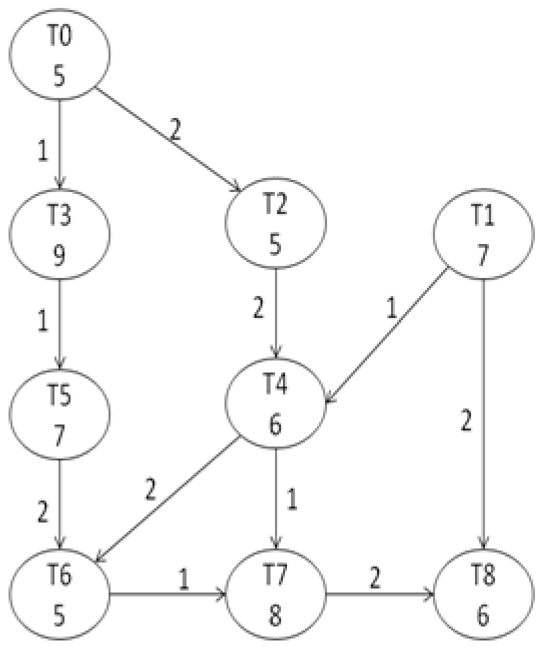

Figure 2.3: Structure of a task graph

- *Latest Finish Time(LFT)*: The latest time at which the task can be completed without delaying the overall completion time of the application.

Slack Time (SLT) of a task is the amount of time that the task can be delayed past its earliest start or earliest finish without delaying the application. The slack time for an activity is the time between its EST and LST, or between its EFT and LFT. So, $SLT = LST - EST = LFT - EFT$

For a TG, the critical path is the path through the TG in which all the tasks have zero slack time, that is, the path for which $EST = LST$ and $EFT = LFT$ for all tasks in the path. A delay in the critical path delays the application.

Since the network bandwidth available now a days is higher and we are focusing on computing intensive tasks, we can consider that the time required to send the output from a task to its dependent tasks is relatively very low compared to the average computation time of a task though the tasks are being executed on different machines in the grid. So, we can ignore the data transmission time from a task to another, that is the edges of the task graph can be considered to be of zero weight. But to provide more accuracy with preserving simplicity, the output transfer time from a task to another i.e. the output transfer time from any machine to another in grid can be considered to be same for all. So, all the edges of the task graph can be considered of same weight. If all the edges of a task graph are of same weight, we can simplify the graph by adding the edge weight

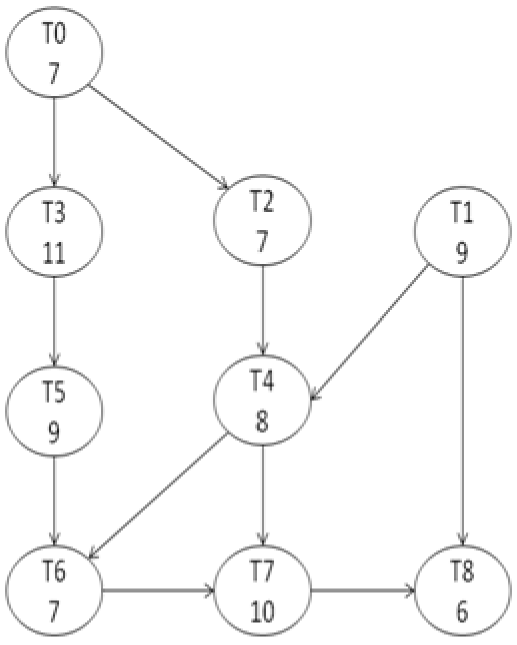

Figure 2.4: TG without edge weight

TN	*D*
EST	*LST*

Figure 2.5: Structur of a node of a TG to show Slack Time and Critical Path

to each task's processing time except for the tasks which have no outgoing edge i.e. no dependent task. Thus we are to deal with a TG without edge weight which will provide simplicity with reasonable accuracy. If we consider the weight of the edges of the task graph of Figure 2.3 to be of 2, then the task graph can be represented by the Figure 2.4.

To compute the slack time of the tasks and critical paths in the TG, we represent each node of TG as shown in Figure 2.5. The details of the abbreviations are as follows:

TN - Task Number

D - Duration

EST - Earliest Start Time

LST - Latest Start Time

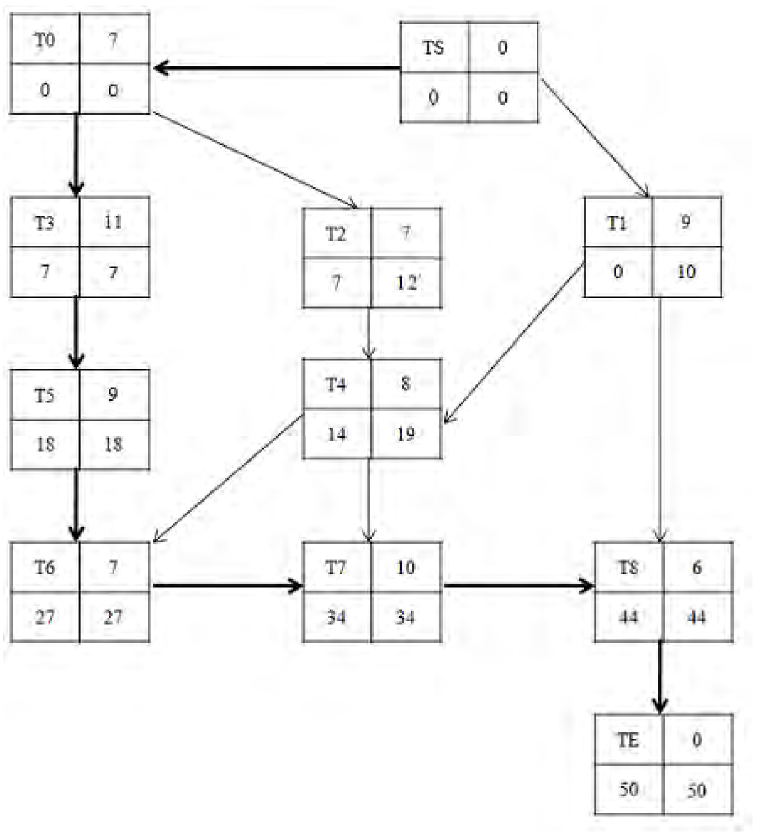

Figure 2.6: Task Graph with EST and LST

The TG of Figure 2.4 is represented in Figure 2.6 with new node structure. Two dummy nodes with zero computation time namely TS and TE have been added. There are edges from TS to the nodes which have no parent node and edges to TE from the nodes which have no child node.

From the TG as represented in the 2.6, we see that there are some nodes whose EST and LST are same. So, the slack time of those tasks are zero. Other tasks whose LST is greater than EST, have nonzero slack time. The paths from TS to TE on which all tasks have zero slack time, are the critical paths. There must be at least one critical path in a TG. In the TG shown in Figure 2.6, there is only one critical path which is marked with relatively thick edges. The critical path is $TS \rightarrow T0 \rightarrow T3 \rightarrow T5 \rightarrow T6 \rightarrow T7 \rightarrow T8 \rightarrow TE$.

2.6 Related Works on Resource Reservation in Grid

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

Chen [10] 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. Their target applications are those involving high-volume data streams and requiring distributed processing of data arising from a distributed set of sources. They focused on pipelined processing and real-time constraint required by distributed streaming applications. 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 mainly works on multimedia grid based streaming applications and fails to provide any real and generalized architecture for conventional grid computing.

Kun [34] 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 inside the Grid networking environment. Specifically, the efficiency of this QoS mechanism is maximized by policy-based management taking care of the flexible control of QoS parameters/components and active networks technology looking after the fast delivery of various QoS configurations. It provides solution for on demand resource request but there is no provision for resource reservation for future use.

Xing [33] introduces a flexible advance reservation for grid applications. Since requests of advance reservation with fixed parameters, i.e. start time, end time and resource capability, may be rejected due to instantaneous peaks of resource utilization, the call acceptance rate of reservation would decrease dramatically, and the performance of resource may be reduced. In fact, many resource reservations for grid applications do not need fixed parameters. 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. They have shown that their approach 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. Though call acceptance rate and resource utilization is increased, the QoS parameters like delay have not been considered here.

Sulistio [31] 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 presented modified versions of Linked List and Segment Tree data structures to support add, delete, and search, as well as the interval search operation to deal with advance reservations in computational grids. For this, they had to developed an algorithm for finding closest interval to a requested reservation for Segment Tree they introduced and adapted Calendar Queue data structure for managing reservations as well. They also propose a new data structure that is tailored to handle the operations for future reservations efficiently. The new data structure is called Grid advanced reservation

Queue (GarQ), which is a combination of Calendar Queue and Segment Tree, for administering reservations efficiently for the above required operations. They demonstrate this by doing a comprehensive performance evaluation using several real-world workload traces. Their results show that GarQ has a better performance time on average when dealing with reservation operations compared to other data structures. The data structure they proposes improves some weaknesses of the aforementioned data structures but highly depends on parameters such as size of interval. Moreover they consider only single task applications thus making the system very simple one with no provision for resource utilization.

Yin [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. Another drawback is that their algorithm does not work for multiple resources.

Cirne [11] discussed about how to run Bag-of-Tasks applications on computational grids. Bag-of-Tasks applications are both relevant and amendable for execution on grids. They investigated the reason for why few users execute their Bag-of-Tasks applications on grids and introduced MyGrid, a system designed to overcome the identified difficulties. MyGrid provides a simple, complete and secure way for a user to run Bag-of-Tasks applications on all resources. MyGrid embeds two important research contributions to grid computing. First, they introduced some simple working environment abstractions that hide machine configuration heterogeneity from the user. Second, they introduced a scheduling heuristics that attains good performance without relying on information about the grid or the application, although consuming a few more cycles. Non dependancy on information makes the scheme much easier to deploy in practice. However, their proposed system does not work for workflow-based applications which are more complex to schedule in a grid environment.

Blythe [7] has presented a resource reservation approach for workflow-based applications in grid. The algorithm is an iterative one where in each iteration, an initial allocation is constructed in a greedy approach. In principle a number of local modifications might be considered by swapping pairs of tasks, but this is not implemented in the system. The initial allocation algorithm computes the tasks whose parents have already been scheduled on each pass, and considers every possible resource for each such task. For each (task, resource) pair, the algorithm computes the increase to the current makespan of the workflow if the task is allocated to this resource. If I-min be the lowest increase found and I-max be the largest, the algorithm picks one pair at random from those whose increase I is less than I-$min + \alpha(I$-max, I-$min)$ for some width parameter α, $0 \leq \alpha \leq 1$, and continues until all tasks are allocated. The width parameter α determines how much variation is allowed each time a candidate workflow allocation is constructed. When α = 0, each iteration of the algorithm behaves like task-based min-min solution [9]. When α = 1, each iteration is random. In some domains, a non-zero α is essential to find globally optimal allocations, while in others the variation due to several component allocations having equally good heuristic scores is enough to find the optimal. They didn't consider some important properties of a workflow-based application like slack time of the tasks, critical paths of the application etc.

Viera [32] presents an authentication architecture to access grid resources from mobile devices, utilizing a lightweight user-centric authentication approach. As is the case for most access to grid resources, the use of mobile devices requires user authentication. Given the limited amount of power available for mobile devices, the applications on these devices, including the authentication services used to access grid resources, need to be designed to conserve power. They proppposed an approach to overcome the limitations of mobile devices, such as limited battery power. The approach applies to user authentication with any grid resource or service such that mobile users can utilize grid environments in a transparent and secure way. This system mainly works for access of mobile devices to grid, but fails to provide any generalized approach for conventional grid computing.

Chapter 3

New Resource Reservation Scheme in Grid

We proposed a new resource reservation scheme for workflow-based applications in grid computing environment. In this chapter a detailed description of the system architecture of our resource reservation scheme is presented with an illustrative example. The proposed data structure is also described with necessary examples.

3.1 Problem Statement

The problem of scheduling the tasks of a task graph on distributed services belongs to a class of problems known as NP-complete problems. For such problems, no known algorithm is able to generate the optimal solution within polynomial time. Even though the scheduling problem can be solved by exhaustive search, the complexity of the methods for solving it is very high. In grid environments, scheduling decisions must be made with in a time limit, because there are many users competing for resources. Thus, in practice, heuristics are most often used to schedule workflow applications in distributed system environments like grid.

An application may not start execution at the time when it is expected to be due to unavailablity of resources at that time. The application starts execution when resources are available. So, the scheduled start time may be delayed from the expected start time. Let, a workflow-based application(WBA) is represented by ($ExST, G$) where $ExST$ is the expected start time of execution of the WBA, G is the directed acyclic graph (DAG) that represents the WBA and $G = (T$, E). Here, $T = \{ T_1, T_2, T_3, \cdots, T_n \}$ is the set of nodes of DAG and represents the tasks of the WBA . $E = \{ e_1, e_2, \cdots, e_n \}$ is the set of edges and e_i = (T_a, T_b) represents that task T_a must be completed before task T_b starts. Now consider there are n Computing Nodes (CN) available in the grid and they are $CN_1, CN_2, \cdots, CN_n$. $t(T_i)$ denotes the time task T_i takes to execute on a CN.

Scheduled Duration (SD) is the duration from expected start time of execution of the WBA to the time when execution of all tasks of the WBA is completed. Let, $t_e(WBA)$ denotes the time when all tasks in the workflow is completed. So, scheduled duration is defined as

$$SD = t_e(WBA) - ExST \tag{3.1}$$

The Minimum Duration (MD) is the minimum time required to complete the application. So, minimum duartion of a WBA is the length of the critical path. Considering a WBA without edge weight, the length of a critical path is the summation of duration of all individual tasks in the path. Let, CP is the set of all tasks of a critical path of the WBA. So, minimum duration is defined as

$$MD = \sum_{\forall T \in CP} t(T) \tag{3.2}$$

Now, the delay that a WBA is to experience to complete its execution is the difference between SD and MD. Let, $DELAY_{WBA}$ denotes the delay.

$$DELAY_{WBA} = SD - MD \tag{3.3}$$

The objective is to schedule the tasks of the WBA to the CNs with an aim to minimize the $DELAY_{WBA}$.

3.2 Proposed Resource Reservation Scheme

The overall system architecture of our proposed resource reservation scheme is shown in Figure 3.1. The components of the system, messages and their sequences in the system have been described by the caption of the blocks in the figure. Scheduling an application in grid means to map the tasks of the application to the CNs of the grid. So, the terms $Schedule$ and Map have been used interchangeably in this literature.

3.2.1 Components of the system

There will be four major components of the system namely $User$, $Resource\ Provider$ (RP), $Computing\ Node$(CN) and $Broker$.

User

A $User$ is the entity that needs its applications to be executed. A $User$ must register with the $Broker$ before using the grid. $User$ has to agree with the policies and conditions as provided by the $Broker$ before registration is completed and a registered $User$ can quit from the grid following the policies and conditions. Each $User$ has a unique Identification Number(ID) in the system.

Computing Node

A Computing Node (CN) is the unit on which a task will be executed. Resources in grid may be of heterogeneous capacity. Scheduling interdependent tasks in heterogeneous grid is relatively complex. Another problem is that such systems are less fault tolerant since failure of any resource requires another free resource with equal or more capacity to accomodate the tasks that the failed resource was to execute. Specially if a powerful resource is failed at any time, the possibility to get an equal or more powerful free resource at that time is very low. If such free resource is not available, then many tasks may need to be rescheduled. So, the performance of the system and benefits of advance reservation degrade. In our system the resources may be of heterogeneous capacity, but homogenous capacity from each resource is used. This homogeneous capacity is called a CN. Since homogeneous capacity is used, in case of failure of any computing node the possibility to get another free one is very high. This makes our system more fault tolerant. In grid, resources with about similar capacity may be grouped together to schedule an application. Since now a days almost all computers support multitasking, a resource with more capacity can be considered to be of multiple CNs. The CNs in grid are not dedicated for use in the grid only. Generally the idle time or unused capacity of a CN is used to process tasks of the grid. So, the regular and intended uses of the CNs of a grid are not disturbed. Each CN has a unique ID in the system.

Resource Provider

A Resource Provider (RP) is a collection of CNs. An RP works as an interface to the grid and is responsible for coordination and managing its CNs for use in the grid. So, a $Broker$ deals with the RP instead of dealing with the individual CNs. Generally an RP consists of the CNs of an organization, laboratory, home, office and so on. Even a single CN itself can work as an RP. A CN can be associated with only one RP. An RP must register with the $Broker$ before being used in the grid. The RP has to agree with the policies and conditions as provided by the $Broker$ before registration is completed and a registered RP can quit from the grid following the policies and conditions. All the CNs of an RP are connected logically. Each RP has a unique ID and an RP provides and maintains the ID of its CNs.

Broker

The $Broker$ coordinates and works as an interface between the $Users$ and the RPs. Both $Users$ and RPs register with the $Broker$ in order to participate in the Grid. The resource reservation will be implemented by the $Broker$. All the $Users$ and the RPs are connected to the $Broker$ and deal with it for their activities in grid. During registration $Broker$ provides each of the $Users$ and RPs their IDs which will latter be used for further activities in the grid.

3.2.2 Phases of resource reservation

The phases in our proposed system are similar to that of [28] to some extent. Details of the phases of our proposed system are described bellow:

Request Phase

Whenever a $User$ has an application to reserve resources for it, the application along with QoS requirements are sent to the $Broker$ mentioning the expected start time of the application and other necessary information. If the $Broker$ is busy, the application is queued otherwise it is handled immediately. Upon taking an application in hand, the $Broker$ sends request to all the listed RPs asking for availability of their CNs mentioning the expected start time and other necessary information required.

Response Phase

Upon receiving the request from the $Broker$, an RP checks the availability its CNs from expected start time. The CNs that are available to provide resources for the application are listed along with their availability status and other necessary information. Then the list is sent to the $Broker$.

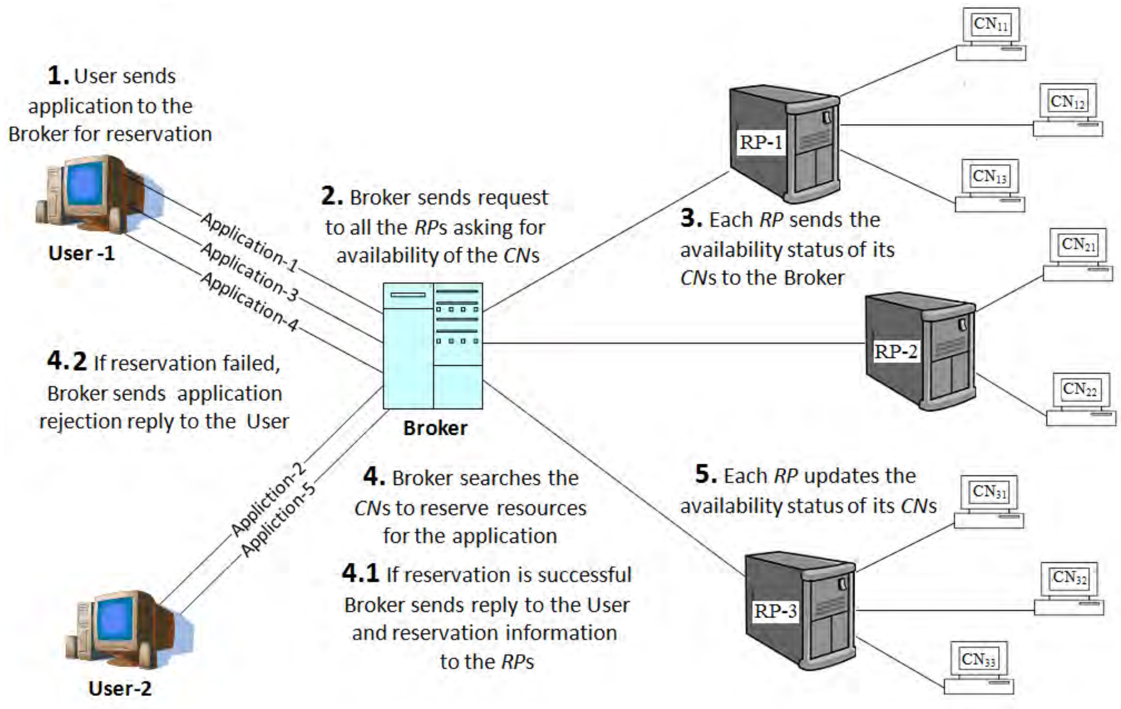
1. User sends application to the Broker for reservation
User -1
Application-1
Application-3
Application-4
2. Broker sends request to all the RPs asking for availability of the CNs
RP-1
CN11
CN12
CN13
3. Each RP sends the availability status of its CNs to the Broker
CN21
RP-2
CN22
4.2 If reservation failed, Broker sends application rejection reply to the User
Broker
Application-2
Application-5
4. Broker searches the CNs to reserve resources for the application
4.1 If reservation is successful Broker sends reply to the User and reservation information to the RPs
5. Each RP updates the availability status of its CNs
CN31
RP-3
CN32
CN33
User-2

Figure 3.1: Architecture of our proposed system

Search Phase

When the $Broker$ receives the status of the CNs from the RPs, it searches for the available time slots to allocate the tasks of the applications. Scheduling the tasks of the application to the CNs is the main challenge. Because any algorithm that produces optimal solution is NP-complete. So, a heuristic is used here. The heuristic is used to compute a scheduling that will result optimized (but not optimal) duration.

Reply Phase

After the search is completed, if the requirements of the user is satisfied, the $Broker$ sends a reply message to the $User$ indicating the total time required to complete the application and the scheduling information to the corresponding RPs so that the RPs can reserve their CNs for the tasks. Otherwise, a message is sent to both the $User$ and the corresponding RPs indicating the failure.

Reservation Phase

When an RP receives the reservation information, it reserves its CNs for the time period as specified to schedule tasks. It also saves the schedule so that the result produces after completion of a

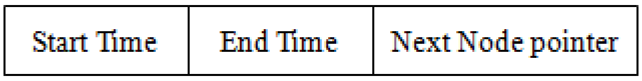

Figure 3.2: A node of the linked list to contain reservation information

task can be sent to the CNs as specified by the dependency of the tasks.

3.3 Major Data Structures

The major data structures used in the system are for the application, the resource reservation information of the CNs and information for task to CN mapping. The data structures for the application have already been described in the previous chapter. Others have been described in the following subsections.

3.3.1 Data structures for Resource Availability Status

For each of the CN, there must be an appropriate data structure to hold the information of availability of its resources. A link list is used here and each node of the list represents a time slot for which the resources of the CN is reserved. Each node contains information about the start time and end time of the slot. So, resources are available for the application when they are not reserved. The nodes are sorted in order of start time. The structure of a node is shown in Figure 3.2. RPs use this data sructure to maintain the status of resource reservation of their CNs.

A special node used as head which contains the ID of the CN, the resource type and the reference time from which other time will be counted. A dummy node whose start and end time both are zero marks the end of the linked list. Figure 3.3 shows the structure of the linked list used for reservation status of a single resource or a collection of resources that are required by the application. ST_i and ET_i represent the start time and end time of the ith slot. Type in the head means the resource type for which reservation information is kept in the list.The resources of the CN is free for the time between ET_i and ST_{i+1}.

Suppose the time unit is minute and the processor of a CN is reserved as per Table 3.1. If the reference or base time is considered to be 00:00 at 20-04-2011, then the relative start and end time would be as per Table 3.2. Figure 3.4 shows the linked list containing the reservation information of Table 3.2. Here, CN_1 represents the ID of the CN and $Proc$ means that the resource type is processor.

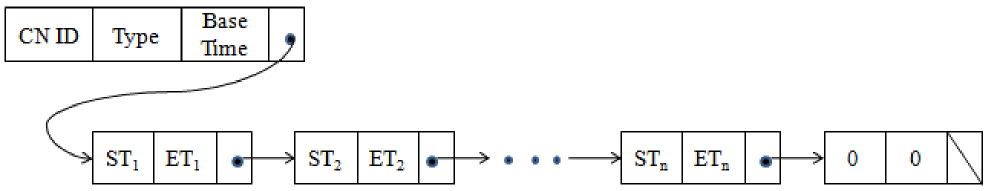

Figure 3.3: Linked list to keep reservation information of a resource of a CN

Table 3.1: Example of reserved time space of a CN

Date	Start Time	End Time
20-04-2011	8:00	8:15
20-04-2011	14:30	16:12
21-04-2011	5:25	17:46

3.3.2 Data structure for task to CN mapping

When the tasks of an application are scheduled or mapped to the CNs, a data structure is needed to keep this mapping information. For each application User ID(UID) , Application ID($AppID$) and Scheduled Start Time (SST) of the application is stored. A table is used to store the allocation information for each of the tasks. The entries of the table are $Task\ No.$, $Duration$, $Dependency$, $RP\ ID$, $CN\ ID$, $Start\ Time$. For example, let there are nine tasks in the application named $T0$, $T1$, $T2$, $T4$, $T5$, $T6$, $T7$ and $T8$ and two RPs in the system, $RP1$ and $RP2$. $RP1$ has two CNs and $RP2$ has one CN. The CNs of $RP1$ are $CN11$ and $CN12$ and of RP is $CN21$. The scheduled start time of the application is at 5:00 on 20-04-2011. Table 3.3 shows an example of storing mapping information of the tasks of an application to the CN.

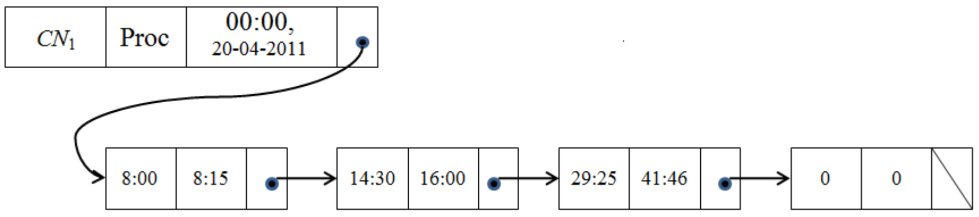

Figure 3.4: Linked list to keep resource reservation information as shown in Table 3.2

Table 3.2: Start and end time relative to reference time

Node No.	**Start Time**	**End Time**
1	8:00	8:15
2	14:30	16:12
3	29:25	41:46

Table 3.3: An Example of storing mapping information for an application

Task	**Duration**	**Dependencies**	**RP ID**	**CN ID**	**Scheduled Start Time**
$T0$	7	-	$RP1$	$CN11$	5:00
$T1$	9	-	$RP1$	$CN12$	5:05
$T2$	7	$T0$	$RP2$	$CN21$	5:15
$T3$	11	$T0$	$RP1$	$CN12$	5:18
$T4$	8	$T1, T2$	$RP2$	$CN21$	5:25
$T5$	9	$T3$	$RP2$	$CN11$	5:33
$T6$	7	$T4, T5$	$RP1$	$CN11$	5:50
$T7$	10	$T4, T6$	RP_1	$CN12$	5:58
$T8$	6	$T1, T7$	RP_2	$CN11$	6:10

3.4 Proposed Approach

The algorithm presented here for scheduling the tasks of an application to the CNs is an iterative one. It runs multiple iterations where each iteration produces a mapping of the tasks of the application to the CNs. The less the duration of a mapping, the better it is. If an iteration produces better result than the best result of the previous iterations, the latter result is stored as the best replacing the previous one. If a mapping with zero delay is produced, further iterations are not executed, because no better result is possible. So, the greater the number of iterations, the better the result is expected. The maximum number of iterations is not fixed rather it is set by the system based on factors like processing speed of the system i.e. how fast it can run the algorithm, number of waiting applications in the queue, the delay allowed etc. System may set the number of iterations based on statistics of previous mapping. Thus, this approach will provide more flexibility to deal with different scenarios and QoS requirements.

Due to dependency of the tasks of a WBA, at any time, only the tasks whose all parents have been scheduled are considered for scheduling at that time. These are the $ready\ tasks$ for scheduling. The main challenge is to map the tasks to CNs that will result less scheduled duration of the whole application. We have used the slack time of the tasks to select a task from the list of ready tasks. Since delay of any task with zero slack time i.e. any task on the critical paths will result a delay of the whole application, these tasks will be given the highest priority. In other word, the more the slack time of a task, the less the priority would be given to the task and vice versa. But, if tasks with less slack time is always selected for scheduling, some better result may always be left. So, we have used a probability to select a candidate task at any time. From all the task, we select a task randomly where task with higher slack time has less possibility to be selected and vice versa.

To calculate probability for the tasks, each of the ready task is assigned a value. This value is calculated by dividing the sum of the slack time of all tasks by the slack time of the task. Since there are some tasks with zero slack time, one is added with the slack time of a task for this division operation. This results the greater value for task with less slack time and vice versa. The probability of any task to be selected is its value divided by the sum of value of all the ready tasks. Since task with less slack time has greater value, the task has higher probability to be selected.

Let, at any time n ready task to be scheduled are $T_1, T_2, \cdots T_n$. S_i and $Value_i$ denote the slack time and value of task T_i respectively.

The sum of slack time of all tasks is,

$$S = \sum_{i=1}^{n}(S_i + 1) \tag{3.4}$$

Value of task T_i is,

$$Value_i = \frac{S}{S_i + 1} \tag{3.5}$$

So, the probality of T_i to be selected is,

$$Pr(T_i) = \frac{Value_i}{\sum_{j=1}^{n} Value_j} \tag{3.6}$$

During implemention of the algorithm, each task T_i is assigned a range defined by two numbers, $rangeMin_i$ and $rangeMax_i$ that indicate the marking for selection of the task and their difference is equal to $Value_i$.

$$Value_i = rangeMax_i - rangeMin_i \tag{3.7}$$

The ranges of the tasks are contiguous and $rangeMin_1 = 0$. For $i \geq 2$, $rangeMin_i$ is defined as,

$$rangeMin_i = rangeMin_{i-1} + Value_{i-1} \tag{3.8}$$

A random number is generated with uniform distribution in the range [0, $rangeMax_n$). Consider the generated random number is $randomNum$. Now the scheduled task will be $T_{selected}$ if the following condition holds

$$rangeMin_{selected} \leq randomNum < rangeMax_{selected} \tag{3.9}$$

From the set of ready tasks, $T_{selected}$ is selected for scheduling.

When a task is selected from the candidate lists, all the CNs are searched for available resources to reserve for the task. The CN which results minimum delay is selected. If more than one CN is available with minimum delay, one of them is selected randomly. The task is mapped to the selected CN. If the minimum delay is non zero, earliest start time of the task is increased by the minimum delay. If the adjusted earliest start time plus the duration of the selected task is greater than the earliest start time of its child tasks, the earliest start time and slack time of those child tasks are adjusted. The same procedure is applied for the child tasks and all its descendents.

During adjusting the child tasks, slack time of some tasks may be negative. For example, consider an application as shown in Figure 3.5. Let $T0$ has been scheduled at its start time at zero. Due to unavailability of resources, $T2$ can not be scheduled at its start time at 4. Resources are available at time 15 to schedule $T2$. So, the adjusted start time of $T2$ is set to 15 and the delay is $15 - 4 = 11$ time unit. Now, $T4$ has slack time of 8 time unit and it has been delayed by 11 time unit. So, the adjusted start time of $T4$ is $7 + 11 = 18$ and the slack time is $8 - 11 = -3$. Now, $T4$ will be finished by $18 + 2 = 20$. So, the adjusted start time of $T5$ is set to 20 and it is delayed by $20 - 17 = 3$ time units. $T5$ has zero slack time and so its slack time is set to $0 - 3 = -3$ time unit. Now, $T1$ and $T2$ has slack time of zero and that of $T4$ and $T5$ is -3. So, relatively $T1$ and $T2$ has 3 unit more slack time than the others. Here the generated minimum slack time is negative. To adjust the minimum slack time to zero, we reduce the slack time of all unmapped task by the generated minimum slack time. As a result, the slack time of $T4$ and $T5$ is further adjusted to zero and that of $T1$ and $T2$ to 3.

After scheduling a task to a CN and adjusting other unmapped tasks, same procedure is applied to schedule next task. All tasks of the application are mapped in this way and a mapping is produced. Multiple iterations are executed and a new mapping is generated in each of the iterations. Among the mappings generated, the best mapping i.e. mapping with the minimum duration is selected to schedule the application. This strategy to select the best one from multiple mapping has been adapted from Blythe [7], with which we have compared our algorithm. The number of iterations to execute would be determined based on various factors like time available to run the scheduling program, QoS requirements of the application, user priority etc.

3.5 Major Algorithms and Their Descriptions

3.5.1 Algorithms Related to Data Structure

The list of major algorithms related to data structures used in our system are given below with the operations they perform. For details please refer to appendix A.

1. *CreateNode*(*int startTime, int duration*) - Creates a single node of reservation status list for the $duration$ from the $startTime$.

2. *InsertNode*(*Node n, ComputingNode cn*) - Inserts a single node n in the reservation status list of the cn

3. *SearchTime*(*int startTime, int duration, ComputingNode cn*) - Finds the earliest available time slot for the $duration$ from the $startTime$ in the reservation status list of the cn.

3.5.2 Algorithms Related to Resource Reservation Scheme

The details of the attributes used for different objects in the following algorithms have been given below.

Table 3.4: Details of the attributes used

Object	Attribute	Details
Mapping	duration	The total duration that is required to complete the execution of an application if the tasks of the application are scheduled according to Mapping
Task	value	The value of Task that is used to find out the probability of selection of the task
Task	rangeMin	Minimum of the range of value of Task
Task	rangeMax	Maximum of the range of value of Task
Task	startTime	The time when Task will start its execution
Task	slackTime	The slack time of Task
Task	duration	The time Task requires to complete execution

Algorithm 1: SceduleApplication(Application app, int maxIterations, List<CN> cnList)

```
Output: Map app to the CNs of cnList and return best mapping information
1  $minDelay \leftarrow \infty$
2  $minDuration \leftarrow length\ of\ a\ critical\ path\ of\ app$
3  $mapping \leftarrow null$
4  $iterationNo \leftarrow 1$
5  $originalApp \leftarrow app$                    // Used to restore original app
6  $originalCNList \leftarrow cnList$            // Used to restore original cnList
7  while $iterationNo \leq maxIterations\ AND\ minDelay \neq 0$ do
8      $mapping \leftarrow CreateMapping(app, cnList)$     // Maps tasks of app
                                                        // to the CNs of cnList
9      $delay \leftarrow mapping.duration - minDuration$
10     if $delay < minDelay$ then
11         $bestMapping \leftarrow mapping$
12         $minDelay \leftarrow delay$
13     endif
14     $iterationNo \leftarrow iterationNo + 1$
15     $app \leftarrow originalApp$                      // Restore original app
16     $cnList \leftarrow originalCNList$              // Restore original cnList
17 endw
18 return $bestMapping$
```

Algorithm 2: CreateMapping (Application app, List <CN>cnList)

```
Output: Map the tasks of app to the CNs of cnList and return mapping information
1 $mapping \leftarrow null$
2 while $all\ tasks\ in\ app\ are\ not\ mapped$ do
3     $availTasks \leftarrow all\ unmapped\ tasks\ with\ every\ parent\ mapped$
4     $selectedTask \leftarrow SelectTask\ (availTasks\ )$ // Select a task for mapping
5     $selectedCN \leftarrow SelectCN(selectedTask, cnList, app)$        // Select a CN
                                                                       // for selectedTask
6     add $(selectedTask, selectedCN)$ to $mapping$
7 endw
8 return $mapping$
```

Algorithm 3: SelectTask(List<Task> availTasks)

```
Output: Select a task from availTasks
1 $lastRangeMax \leftarrow 0$                    // Used as minRange for next task
2 $total \leftarrow \sum_{\forall_t \in availTasks}(slackTime\ of\ t + 1)$
3 forall the $task\ t\ in\ availTasks$ do
4     $t.rangeMin \leftarrow lastRangeMax$
5     $t.value \leftarrow \frac{total}{t.slackTisme\ +\ 1}$
6     $t.rangeMax \leftarrow t.rangeMin + t.value$
7     $lastRangeMax \leftarrow t.rangeMax$
8 endfall
9 $randomNum \leftarrow A\ random\ number\ uniformly\ distributed\ in\ [0, lastRangeMax)$
10 $selectedTask \leftarrow task\ t\ such\ that\ randomNum \in [t.rangeMin, t.rangeMax)$
11 return $selectedTask$
```

Algorithm 4: SelectCN(Task task, List<CN> cNList, Application app)

```
Output: Select a CN from cnList to map task with minimum delay
1  $startTime \leftarrow \infty$
2  $selectedCN \leftarrow null$
3  $tempCNList \leftarrow cnList$
4  while $tempCNList$ is not empty AND $startTime > task.startTime$ do
5      $cN \leftarrow randomly\ remove\ a\ CN\ from\ tempCNList$
6      $stTime \leftarrow searchTime(task.startTime, task.duration, cN)$      // Searches
                                                  // availability of cN for task
7      if $stTime \leq startTime$ then
8          $startTime \leftarrow stTime$
9          $selectedCN \leftarrow cN$
10     endif
11 endw
12 $delay \leftarrow startTime - task.startTime$
13 if $delay > 0$ then
14     $AdjustTime(task, app)$                   // Adjust the start time and
                                     // slack time of task and its descendents
15 endif
16 $node \leftarrow CreateNode(task.startTime, task.duration)$        // Creates a node
                                                       // of reservation list
17 $InsertNode(node, selectedCN)$                      // Insert node into the
                                            // reservation list of selectedCN
18 return $selectedCN$
```

Algorithm 5: AdjustTime(Task task, Application app)

```
Output: adjust the slack time and earliest start time of the unmapped tasks
1 $task.startTime \leftarrow task.startTime + delay$
2 $AdjustChildTime(task)$                    // Adjust the start time and
                                 // slack time of the descendents of task
3 $minSlackTime \leftarrow minimum\ slackTime\ among\ all\ unmapped\ tasks\ of\ app$
       // Minimum slack time of all tasks are adjusted to zero.
4 if $minSlackTime < 0$ then
5     $unmappedTasks \leftarrow all\ unmapped\ tasks\ of\ app$
6     forall the $task\ t\ in\ unmappedTasks$ do
7         $t.slackTime = t.slackTime - minSlackTime$
8     endfall
9 endif
```

Algorithm 6: AdjustChildTime(Task task)

```
Output: adjust the slack time and earliest start time of the children of task
1 forall the $child\ t\ of\ task$ do
2     $delay = task.startTime + task.duration - t.startTime$
3     if $delay > 0$ then
4         $t.startTime = t.startTime + delay$     // Increase start time of t
5         $t.slackTime = t.slackTime - delay$     // Decrease slack time of t
6         $AdjustChildTime(t)$
7     endif
8 endfall
```

3.6 An Illustrative Example

In this section we illustrate the approach and its important features through an example. Consider a $User$ has sent an application to the $Broker$. The application has six tasks namely $T0$, $T1$, $T2$, $T3$, $T4$ and $T5$. The application is as shown in Figure 3.5(a) and with earliest start time and latest start time for each of the tasks in Figure 3.5(b).

Now consider a grid environment where two RPs named $RP1$ and $RP2$ are available. Each of

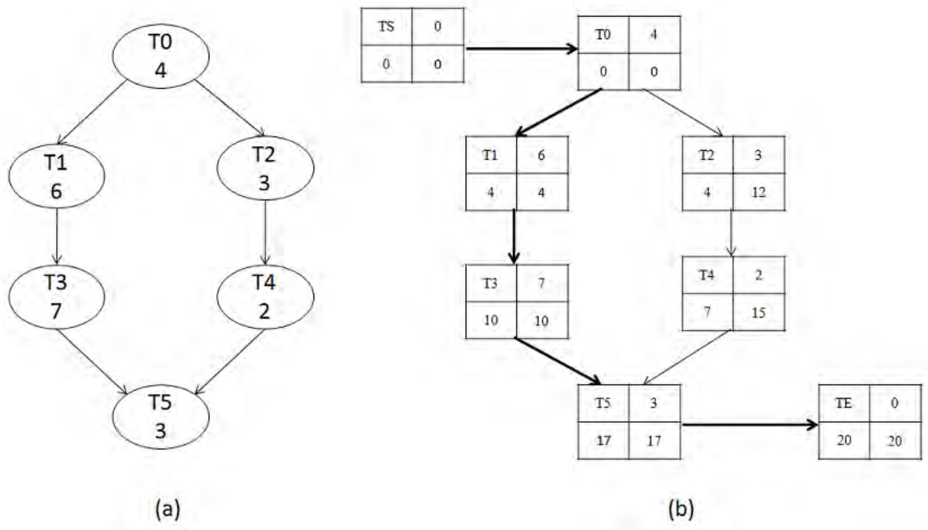

Figure 3.5: (a) Application used as example in this section (b) Representation of the application with earliest and latest start time

the RPs provides one CN each, named $CN1$ and $CN2$ respectively. To make the understanding easy, we use an array which is equivalent to the linked list for status of a CN. An example of a linked list and its equivalent array structure is shown in Figures 3.6(a) and 3.6(b) respectively. The symbols used and their meanings has been shown in Figure 3.6(c). Each slot of the array represents a time unit and the time sequence has been numbered. In this example, we consider the initial reservation status of the CNs as shown in Figure 3.7

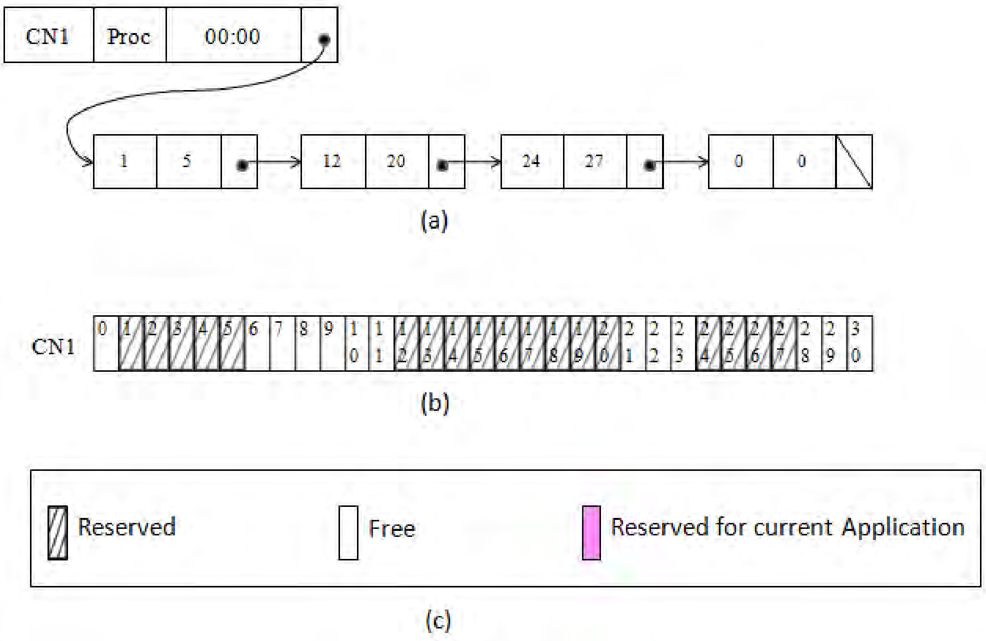

Figure 3.6: (a) A linked list structure for reservation status (b) equivalent array structure of the list (c) Symbols used in the array and their meanings

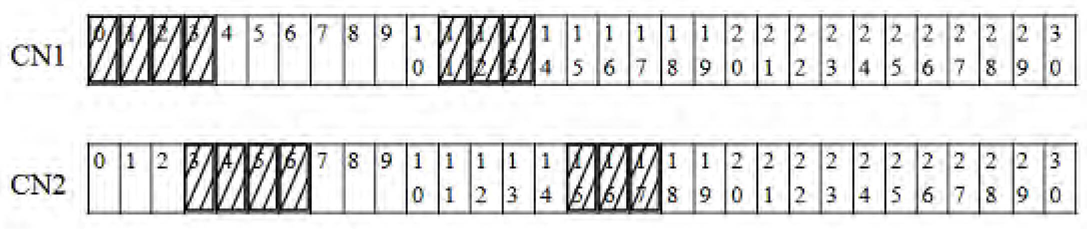

Figure 3.7: Initial reservation status of the CNs considered for the example

Let us consider the execution of the application is expected to start at 5. The tasks of the application, their expected start time, scheduled start time, slack time, dependencies and status are shown in Table 3.5. Since the expected start time of the application is 5, the expected start time of all the tasks have been increased by 5. When the start time of a task is delayed, it is adjusted and represented by Adjusted Start Time. Initially the expected start time and adjusted start time of a task are same. Scheduled start time of a task is determined when the task is mapped. The tasks with zero slack time composes the critical paths. The details of the status used here are as follows:

Dummy - The task is a dummy node and is not mapped. They are helpful for different calculations.

Mapped - The task has been mapped.

Ready - All the parent tasks of this task has been mapped and the task is ready for mapping.

Fresh - The task is not still ready for mapping.

In this example the maximum number of iterations has been set to 2. Each of the following subsection illustrates an iteration of the proposed algorithm. Each of the iterations starts with the initial status of the tasks, application and CNs. In case where more than one CN is available to map a task at the same earliest start time, one of the CNs is selected randomly to map that task.

Table 3.5: The initial status of the tasks

Task	Expected Start Time	Adjusted Start Time	Scheduled Start Time	Dura-tion	Slack Time	Depend-encies	Status
TS	5	5	-	0	0	-	Dummy
T0	5	5	-	4	0	TS	Ready
T1	9	9	-	6	0	T0	Fresh
T2	9	9	-	3	8	T0	Fresh
T3	15	15	-	7	0	T1	Fresh
T4	12	12	-	2	8	T2	Fresh
T5	22	22	-	3	0	T3, T4	Fresh
TE	25	25	-	0	0	T5	Dummy

3.6.1 Details of First Iteration

As per dependancy graph and random selection method, let us assume the sequence of tasks to be mapped is $T0$, $T2$, $T1$, $T3$, $T4$, $T5$.

Mapping $T0$ and $T2$

Table 3.6: Status of the tasks after $T0$ and $T2$ are mapped in the first iteration

Task	Expected Start Time	Adjusted Start Time	Scheduled Start Time	Dura- tion	Slack Time	Depend- encies	Status
TS	5	5	5	0	0	-	Dummy
T0	5	5	5	4	0	TS	Mapped to CN1
T1	9	9	-	6	0	T0	Ready
T2	9	9	9	3	8	T0	Mapped to CN2
T3	15	15	-	7	0	T1	Fresh
T4	12	12	-	2	8	T2	Ready
T5	22	22	-	3	0	T3, T4	Fresh
TE	25	25	-	0	0	T5	Dummy

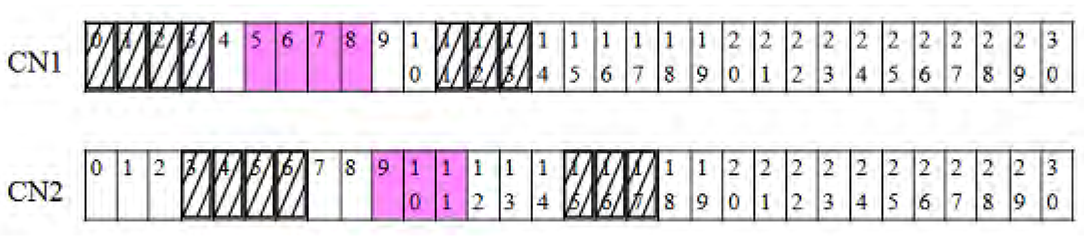

Figure 3.8: Status of the CNs after $T0$ and $T2$ are mapped in the First Iteration

Mapping $T1$

None of the CNs has available time to schedule $T1$ at its earliest start time. The next earliest available time for $T1$ is 14 at $CN1$ and so $T1$ is mapped with $CN1$. Here, the difference between the scheduled start time and earliest start time of $T1$ is 5. So, the start times of $T3$ and $T5$ are adjusted and increased by 5. The Slack Time $T4$ have also been increased by 5.

Table 3.7: Status of the tasks after $T1$ is mapped in the First Iteration

Task	Expected Start Time	Adjusted Start Time	Scheduled Start Time	Dura-tion	Slack Time	Depend-encies	Status
TS	5	5	5	0	0	-	Dummy
T0	5	5	5	4	0	TS	Mapped to CN1
T1	9	14	14	6	0	T0	Mapped to CN1
T2	9	9	9	3	8	T0	Mapped to CN2
T3	15	20	-	7	0	T1	Ready
T4	12	12	-	2	13	T2	Ready
T5	22	27	-	3	0	T3, T4	Fresh
TE	25	30	-	0	0	T5	Dummy

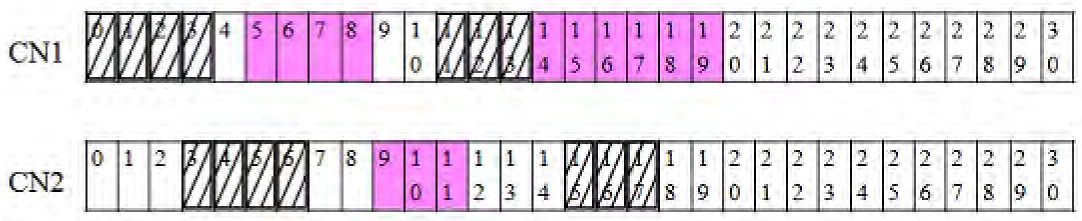

Figure 3.9: Status of the CNs after $T1$ is mapped in the First Iteration

Mapping $T3$, $T4$ and $T5$

Table 3.8: Status of the tasks after $T3$, $T4$ and $T5$ are mapped in the First Iteration

Task	Expected Start Time	Adjusted Start Time	Scheduled Start Time	Duration	Slack Time	Dependencies	Status
TS	5	5	5	0	0	-	Dummy
T0	5	5	5	4	0	TS	Mapped to CN1
T1	9	14	14	6	0	T0	Mapped to CN1
T2	9	9	9	3	8	T0	Mapped to CN2
T3	15	20	20	7	0	T1	Mapped to CN1
T4	12	12	12	2	13	T2	Mapped to CN2
T5	22	27	27	3	0	T3, T4	Mapped to CN2
TE	15	30	30	0	0	T5	Dummy

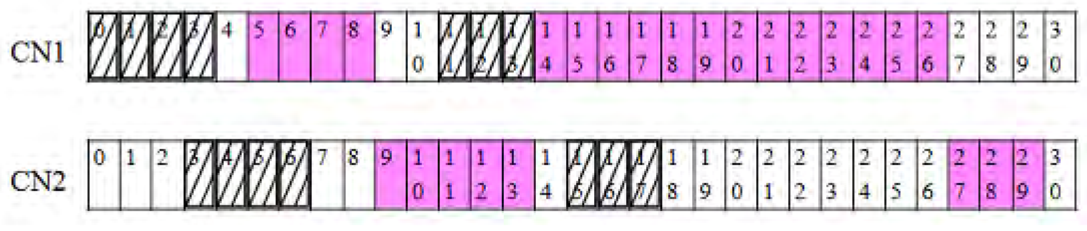

Figure 3.10: Status of the CNs after $T3$, $T4$ and $T5$ are mapped in the First Iteration

Result of First Iteration

From the mapping information as shown in Table 3.8, we see that expected start time of TE is 25 but its scheduled start time is 30. TE is a dummy task and its duration is zero. The difference between its scheduled start time and expected start time is the delay of the application. So, after the First Iteration, a mapping has been generated that results a total delay of 5 to complete the execution of the application in the Grid. Here,

Expected start time of TE = 25

Scheduled start time of TE =30

Total delay$= 30 - 25 = 5$

3.6.2 Details of Second Iteration

In this iteration let us assume the sequence of tasks to be mapped is $T0$, $T1$, $T3$, $T2$, $T4$, $T5$.

Mapping $T0$, $T1$ and $T3$

Table 3.9: Status of the tasks after $T0$, $T1$ and $T3$ are mapped in the Second Iteration

Task	Expected Start Time	Adjusted Start Time	Scheduled Start Time	Duration	Slack Time	Dependencies	Status
TS	5	5	5	0	0	-	Dummy
T0	5	5	5	4	0	TS	Mapped to CN1
T1	9	9	9	6	0	T0	Mapped to CN2
T2	9	9	-	3	8	T0	Ready
T3	15	15	15	7	0	T1	Mapped to CN1
T4	12	12	-	2	8	T2	Fresh
T5	22	22	-	3	0	T3, T4	Fresh
TE	25	25	-	0	0	T5	Dummy

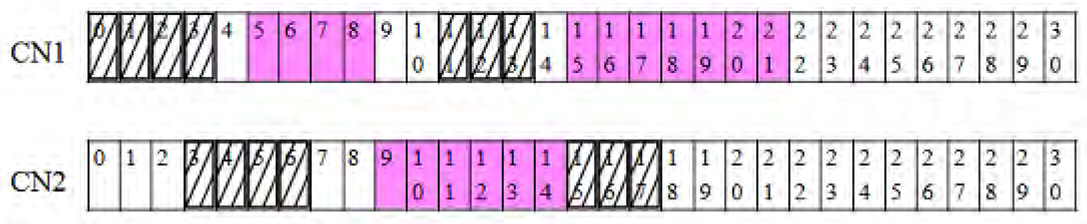

Figure 3.11: Status of the CNs after $T0$, $T1$ and $T3$ are mapped in the Second Iteration

Mapping $T2$

None of the CNs has available time to schedule $T2$ at its earliest start time. So, the next earliest available time for $T2$ is 18 at $CN2$. So, $T2$ is mapped with $CN2$. Here, the difference between the scheduled start time and earliest start time of $T2$ is 9. So, the start times of $T4$ and $T5$ are adjusted and increased by 9 and 1 time unit respectively. The slack times of the these tasks have been set to zero.

Table 3.10: Status of the tasks after $T2$ is mapped in the Second Iteration

Task	**Expected Start Time**	**Adjusted Start Time**	**Scheduled Start Time**	**Dura-tion**	**Slack Time**	**Depend-encies**	**Status**
TS	5	5	5	0	0	-	Dummy
T0	5	5	5	4	0	TS	Mapped to CN1
T1	9	9	9	6	0	T0	Mapped to CN2
T2	9	18	18	3	0	T0	Mapped to CN2
T3	15	15	15	7	0	T1	Mapped to CN1
T4	12	21	-	2	0	T2	Ready
T5	22	23	-	3	0	T3, T4	Fresh
TE	25	26	-	0	0	T5	Dummy

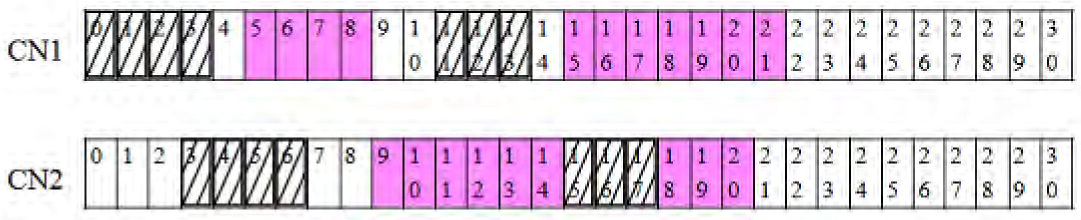

Figure 3.12: Status of the CNs after $T2$ is mapped in the Second Iteration

Mapping $T4$ and $T5$

Table 3.11: Status of the tasks after $T4$ and $T5$ are mapped in the Second Iteration

Task	**Expected Start Time**	**Adjusted Start Time**	**Scheduled Start Time**	**Dura-tion**	**Slack Time**	**Depend-encies**	**Status**
TS	5	5	5	0	0	-	Dummy
T0	5	5	5	4	0	TS	Mapped to CN1
T1	9	9	9	6	0	T0	Mapped to CN2
T2	9	9	18	3	0	T0	Mapped to CN2
T3	15	15	15	7	0	T1	Mapped to CN1
T4	12	21	21	2	0	T2	Mapped to CN2
T5	22	23	23	3	0	T3, T4	Mapped to CN1
TE	25	26	26	0	0	T5	Dummy

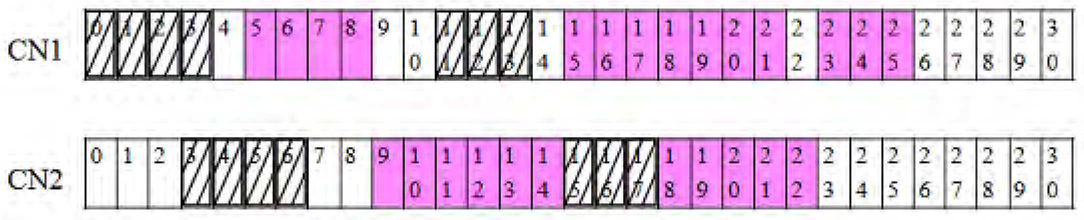

Figure 3.13: Status of the CNs after $T4$ and $T5$ are mapped in the Second Iteration

Result of Second Iteration

From the mapping information as shown in Table 3.11, we see that expected start time of TE is 25 but scheduled start time is 26. So, after the Second Iteration, a mapping has been generated that results a total delay of 1 to complete the application execution in the Grid.

Here,

Expected start time of TE = 25

Scheduled start time of TE = 26

Total delay $= 26 - 25 = 1$

Since the mapping generated in the Second Initeration is better than the previous best, so it will be selected as the new best mapping. The maximum iterations for this example has been set to 2 and so resources will be reserved according to this mapping which results a schedule with 1 time unit delay. If furhter iterations are allowed then the algoritm will run until zero delay mapping is generated or the number of iterations reached to the maximum allowed.

3.7 Complexity Analysis

Let, M, N, R and L denotes maximum iterations, number of tasks in an application, number of CNs available and average lenght of a resource reservation list respectively. In our algorithm, the $CreateMapping$ procedure runs for each iteration. In each iteration all the tasks are mapped. So, each of $SelectTask$ and $SelectCN$ procedures run a total of N times in $CreateMapping$. The maximum number of unmapped task at any time is N. So, in $CreateMapping$ the total time required for $SelectTask$ is $O(N^2)$.

Now, in $SelectCN$ each of the CNs are searched for finding time slot for a task. $SearchTime$ requires $O(L)$ time. So, the total time to find a CN for the task is $O(RL)$. Since the maximum number of child nodes that can be adjusted are N-1, run time of $AdjustChildTime$ is $O(N)$. As stated previously, the maximum number of unmapped task is N and so run time of $AdjustChild$ is $O(N) + O(N) = O(N)$. So, each run of $SelectCN$ requires time of $O(RL) + O(N) = O(N + RL)$. Since, $SelectCN$ is called N times from $CreateMapping$, the total time required for this procedure during each iteration of the algorithm is $O(N^2 + NRL)$.

The add operation in $CreateMapping$ procedure takes O(1) time and it also runs total N times, resulting $O(N)$ time for each of the iterations of the algorithm. So, the time complexity of $CreateMapping$ function is $O(N^2) + O(N^2 + NRL) + O(N) = O(N^2 + NRL)$.

Statements other than $CreateMapping$ procedure take $O(1)$ time in each iteraion of the algorithm. So, time required for each iteration is $O(N^2 + NRL)$. Hence time complexity of our algorithm is $O(MN^2 + MNRL)$. On the other hand, the time complexity of Blythe's algorithm is $O(MN^2RL)$. This shows superiority of our algorithm in terms of time complexity.

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. We have used a simulator for Grid environment named *GridSim* to perform simualtion. The simulation is run using a computer having Intel Core 2 Duo processor, 1.80GHz, 2 GB of memory and Windows Vista operating system.

4.1 Grid Simulation Tools

Simulation is extensively used for modeling and evaluation of real world systems. Consequently, many standard and application-specific tools and technologies have been built for simulation and modeling. They include simulation languages (e.g. Simscript [2]), simulation libraries (e.g. SimJava2 [1][19]), and application specific simulators (e.g. NS-2 network simulator [24]). While there exists a large number of tools, very few well-maintained tools are available for application scheduling simulation in grid computing environments.

4.1.1 Simulation Tools for Grid

OptorSim [6] is developed as part of the EU DataGrid project. It aims to mimic the structure of an EU DataGrid Project and study the effectiveness of several grid replication strategies. It is quite a complete package as it incorporates few auction protocols and economic models for replica optimization. However, it mainly focuses more on the issue of data replication and optimization.

The SimGrid toolkit[8], developed at the University of California at San Diego (UCSD), is a C language based toolkit for the simulation of application scheduling. It supports modeling of resources that are time-shared and the load can be injected as constants or from real traces. It is a powerful system that allows creation of tasks in terms of their execution time and resources, with respect to a standard machine capability.

The MicroGrid emulator[30] , undertaken at the UCSD, is modeled after Globus[16], a software toolkit used for building Grid systems. It allows execution of applications constructed using the Globus toolkit in a controlled virtual Grid resource environment. MicroGrid is actually an emulator meaning that actual application code is executed on the virtual Grid. Thus, the results produced by MicroGrid are much closer to the real world as it is a real implementation. However, using MicroGrid requires knowledge of Globus and implementation of a real system/application to study.

GangSim [13], developed at the University of Chicago, is targeted towards a study of usage and scheduling policies in a multi-site and multi-VO (Virtual Organization) environment. It is able to combine discrete simulation techniques and modeling of real Grid components in order to achieve scalability to Grids of substantial size.

Finally, GridSim [6], development led by the University of Melbourne, supports simulation of various types of Grids and application models scheduling. The following sections explain some major features and capabilities of GridSim.

4.1.2 GridSim Toolkit

GridSim is an open-source software platform, written in Java, that allows modeling and simulation of entities in parallel and distributed computing. for design and evaluation of scheduling algorithms. It provides a comprehensive facility for creating different classes of heterogeneous resources that can be aggregated using resource brokers for solving compute and data intensive applications. GridSim is based on SimJava2, a general purpose discrete-event simulation package implemented in Java. All components in GridSim communicate with each other through message passing operations defined by SimJava2.

Some of the features of GridSim are outlined below:

- It incorporates failures of Grid resources during runtime

- New allocation policy can be made and integrated into the GridSim Toolkit

- It has framework to support advance reservation in a grid system
- An auction model is incorporated into GridSim
- Datagrid extension is included into GridSim
- Network extension is incorporated in GridSim so that resources and other entities can be linked in a network topology
- Background network traffic functionality based on a probabilistic distribution is available. This is useful for simulating over a public network where the network is congested
- It incorporates multiple regional GridInformationService (GIS) entities connected in a network topology. Hence, it is possible to simulate an experiment with multiple Virtual Organizations (VOs)

4.2 Experimental Setup

In SimJava2, each simulated component that interacts with others, is referred to as an entity. GridSim provides both built-in and facilities for user defined entities, algorithms and data structures. We have designed the experiment in GridSim to simulate the system as shown in Figure 3.1. We defined several entities in the experiment. Important entities are described below.

- *User*- This is the entity that sends an application to the Broker for reserving resources for the application. The expected start time of the application is also specified with the application.
- *Broker* - Broker is the most important entity that searches for optimal scheduling of an application to the computing nodes.
- *Application* - An application is generally a task graph which consists of several tasks.
- *Task* - Task is the smallest entity that should be executed in a computer without any break.
- *ApplicationGenerator* - An application generator generates applications from the Standard Task Graph Archives.
- *ResourceProvider* - A resource provider consists of several ComputingNode and it is responsible for the management of its resources and communication with the Broker.

- *ComputingNode* - A Computing Node is the entity that executes the tasks. So, resources of computing nodes are reserved for the tasks for future execution

- *RPGenerator* - RPGenerator generates a set of Resource Providers which are used in the grid.

4.3 Simulation data

To evaluate the performance of the proposed scheme we run the simulation on data from standard Task Graph Archive which is available at [22]. Both task graphs with communication cost and without communication cost are available in the archive. For our simulation we selected task graphs without communication cost. The simulation has been run for three sets of applications. Applications of the first, second and third set consist of 50 tasks, 100 tasks and 300 tasks respectively. The simulation has been performed on different loads of the computing nodes. The system was loaded with some of the applications of the task graph archive before the test application is run. The different loads for which the test application has been run are 0%(Completely free), 25%(Light load), 50% (Average load) and 75% (High load). Different numbers of computing nodes were also used in the simulation. The numbers of nodes for different runs were 10, 15, 20, 25, 30, 35, 40, 45, 50. So, there are 3 different task sets, 4 different loads and 9 different size of computing node sets, which result a total of $3 \times 4 \times 9 = 108$ runs. For each run, we have taken 100 applications randomly from a set and scheduled each of them separately to the computing nodes. The expected start time of the applications were generated randomly.

We have run the simulation both with our proposed algorithm and with the algorithm presented by Blythe et al. [7]. For each run, we have set the number of maximum iterations to 50.

4.4 Measurement Metrics

The metrics considered for evaluation are ZeroDelayedApp, AverageDelay and RunTime.

- *ZeroDelayedApp*: The percentage of the applications which can be scheduled without delay

- *AverageDelay*: The average scheduling delay of an application

- *RunTime*: The average time required to schedule an application

4.5 Outcome of the simulation

We compare our system with an existing system for resource management in grid computing. The work done by Blythe et al. [7] is the most appropriate to compare. This work provides a iterative approach for scheduling applications using Estimated Completion Time (ECT). In Blythe's system there is no consideration of the slack time of the tasks and the critical paths of the applications. We have implemented both our system and Blythe's System in the GridSim simulator. Major outcomes of our experiment have been given in tabular form in Appendix B.

4.6 Comparison of the result

From the detailed outputs as shown in Appendix B, we see that our algorithm performs better than Blythe's algorithm. We have presented here some graphical analysis of the result for the different set of applications at average load (50%) of the computing nodes. Figures 4.1, 4.2 and 4.3 shows the percentage of applications that are scheduled without any delay for different application sets. Here the result has been shown agianst the number of CNs. As the number of CNs increases, the percentage of such applications increases both in our system and Blythe's system, as expected. But the percentage in our system is higher in most of the cases than that of the Blythe's system. So, our system schedules more applications without delay.

Figures 4.4, 4.5 and 4.6 show the average delay that is required to schedule the tasks of an application for different application sets. Similar to the previous result, it has been shown against number of CNs. Here average delay decreases as the number of CNs increases, both in our system and Blythe's system. Figures show that, the average delay in our system is less than that of the Blythe's system in most of the cases.

The more the number of iterations in each run, the better the obtained result is. Figures 4.7, 4.8 and 4.9 show the percentage of applications for which zero delay scheduling is generated as the number of iterations increases at average (50%) load. Here the result has been shown for 20 CNs as sample. From the figures we see that in most of the cases our system produces better result for the same number of iterations used in the algorithms.

Figures 4.10, 4.11 and 4.12 show the average delay of an application as the number of iterations increases at average load. The figures show that our system produces better result for the same number of iterations used in the algorithms. In other words, our system produces same result with

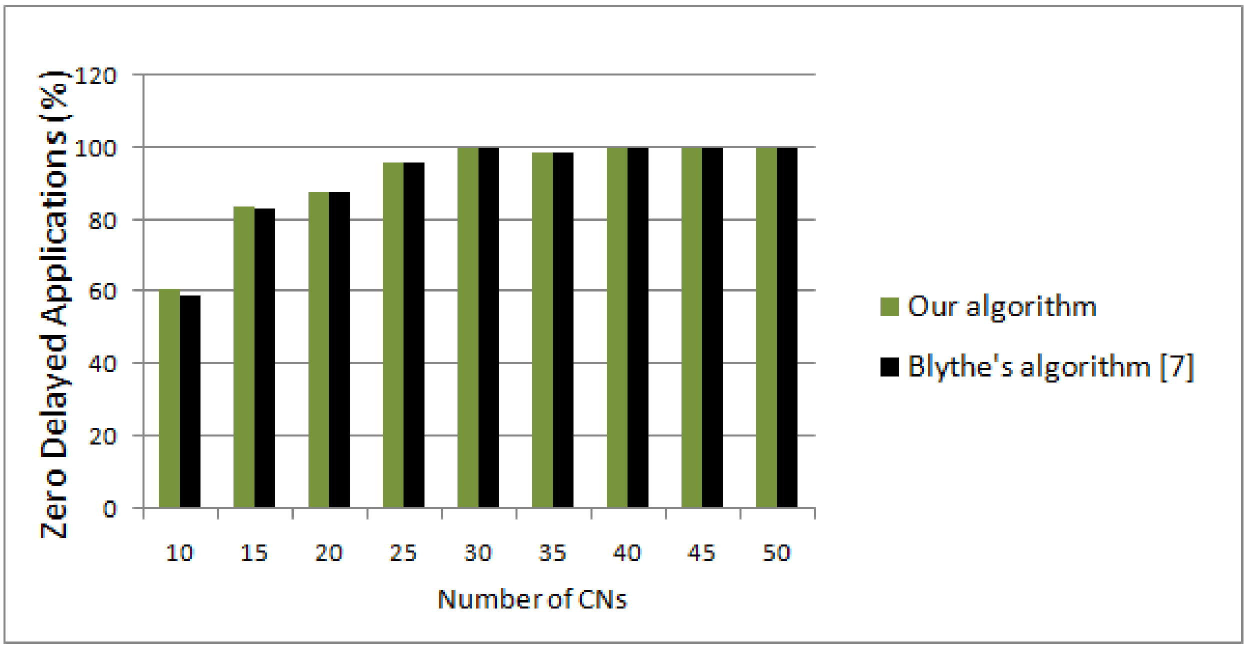

Figure 4.1: Percentage of applications with 50 tasks with zero delay scheduling for different number of CNs at average load

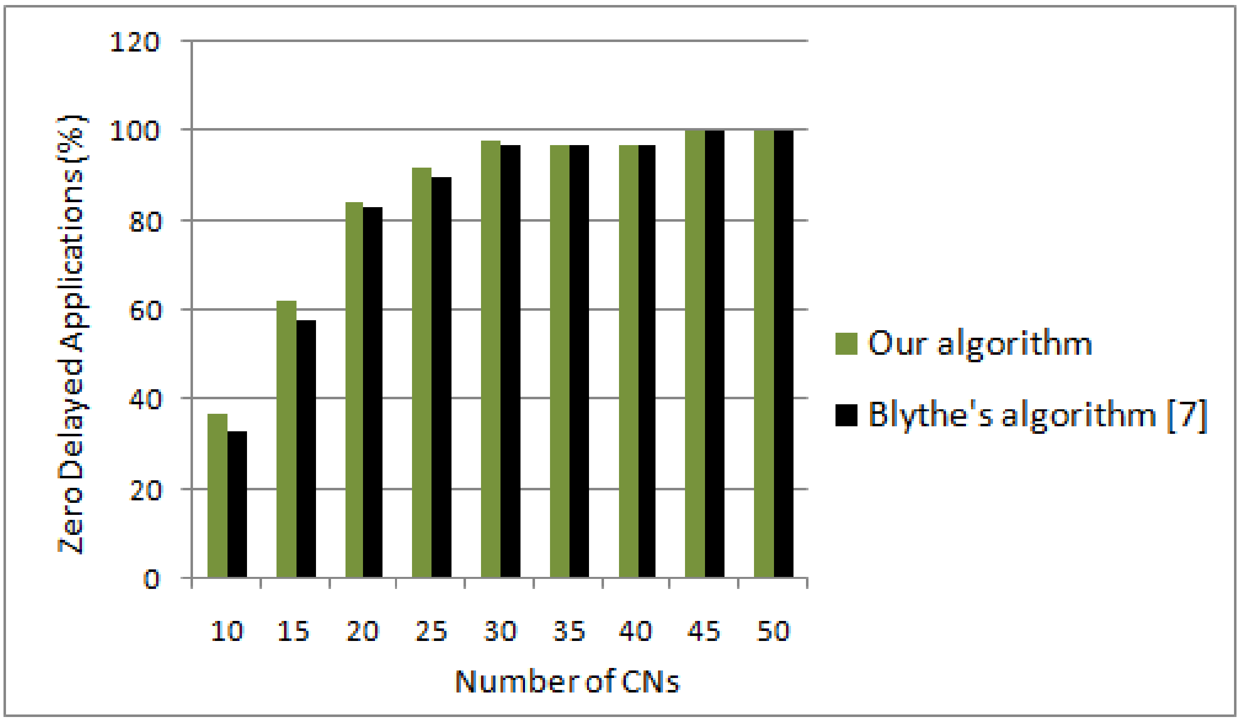

Figure 4.2: Percentage of applications with 100 tasks with zero delay scheduling for different number of CNs at average load

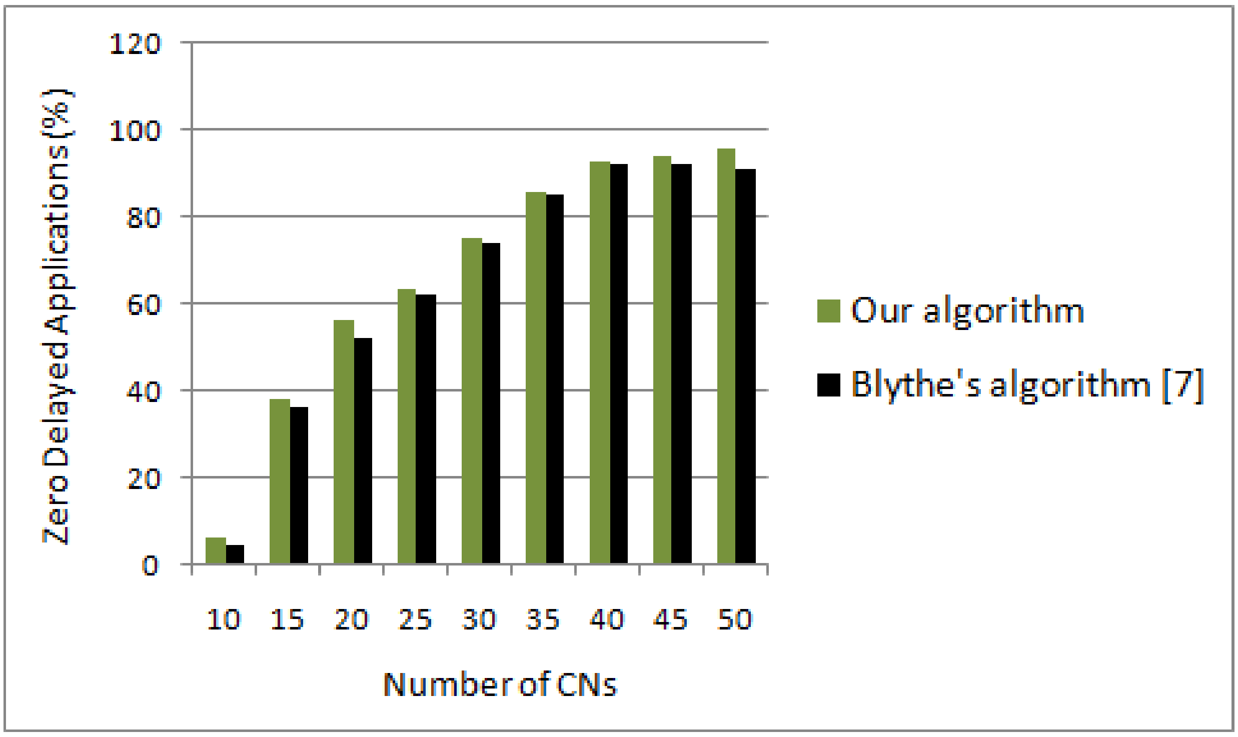

Figure 4.3: Percentage of applications with 300 tasks with zero delay scheduling for different number of CNs at average load

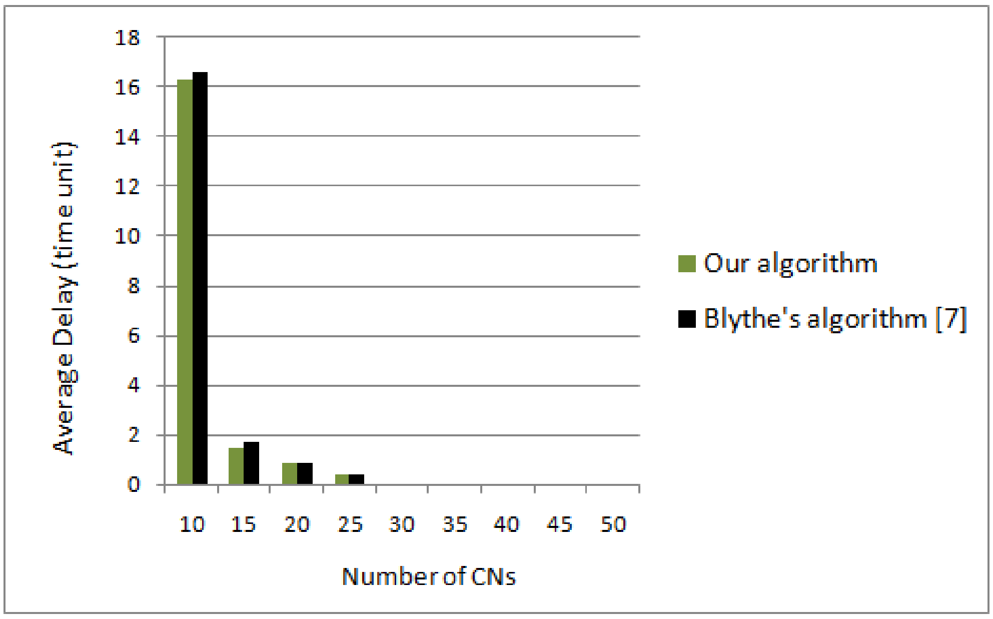

Figure 4.4: Average delay for an application with 50 tasks for different number of CNs at average load

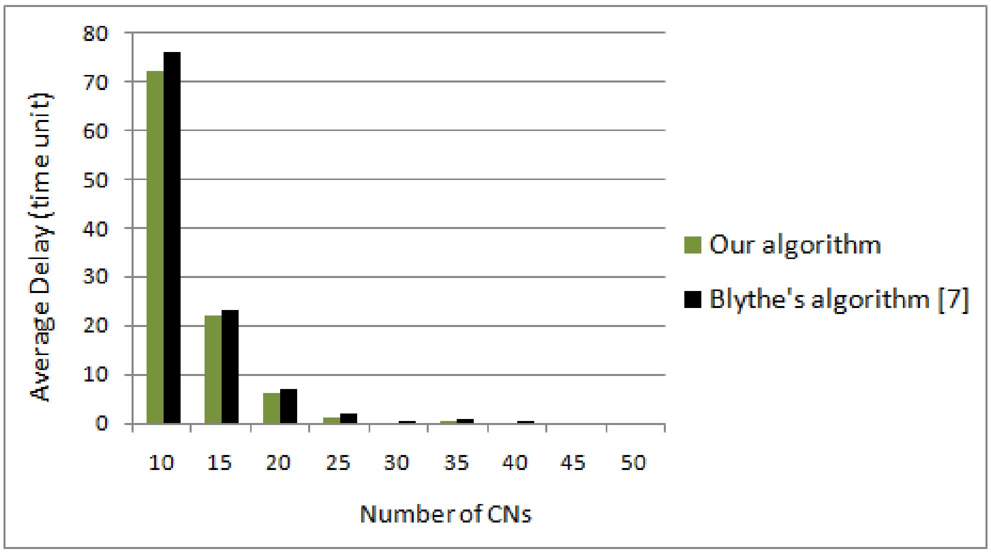

Figure 4.5: Average delay for an application with 100 tasks for different number of CNs at average load

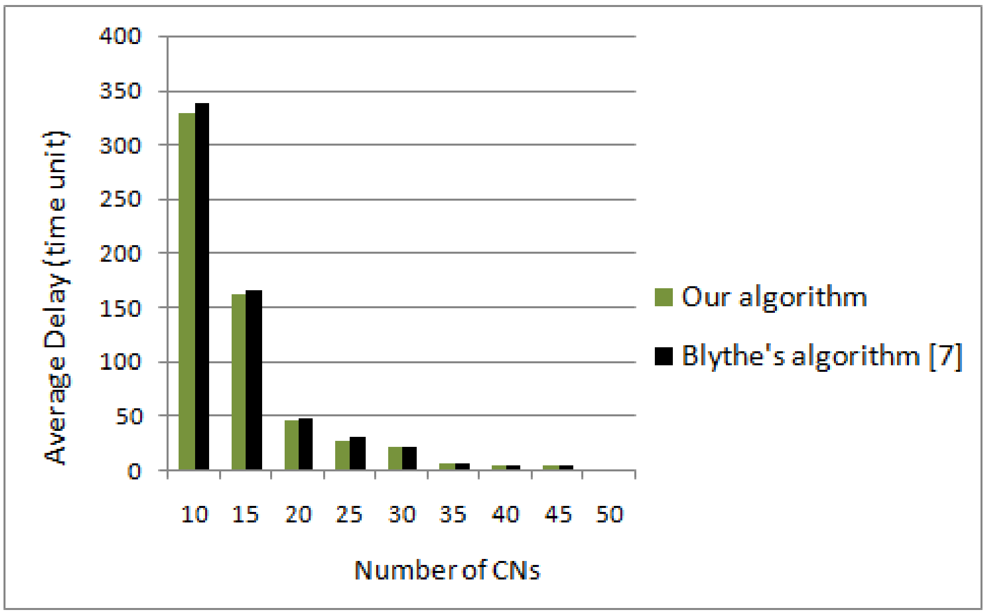

Figure 4.6: Average delay for an application with 300 tasks for different number of CNs at average load

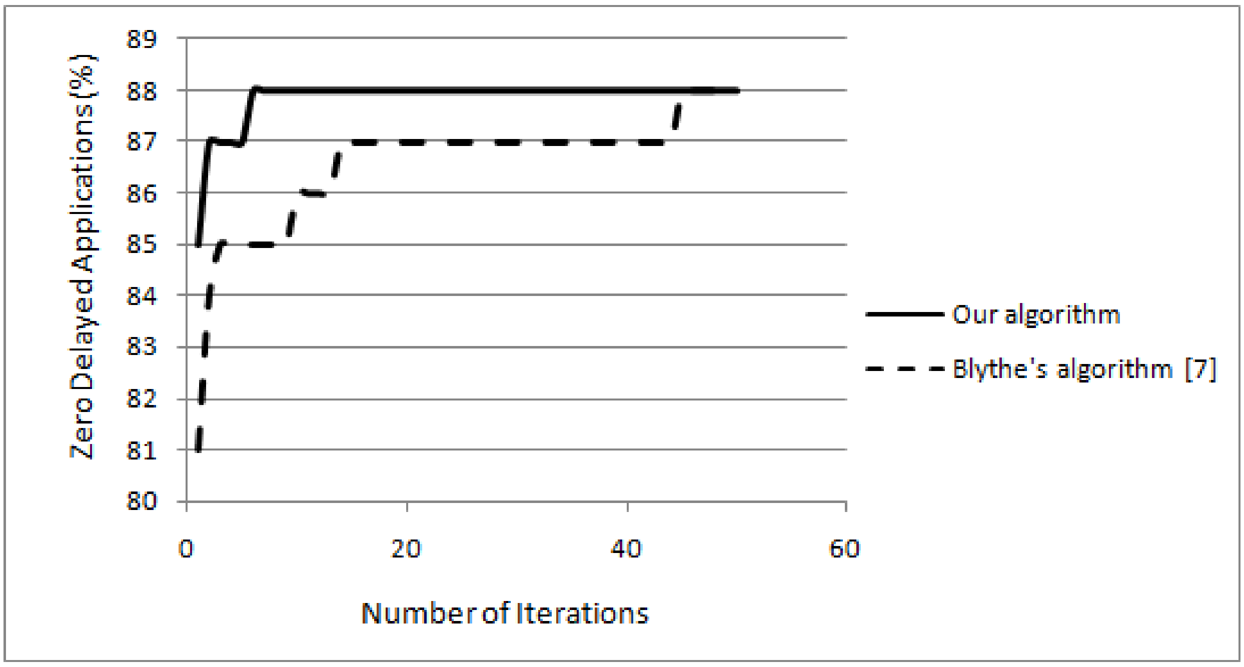

Figure 4.7: Percentage of zero delayed applications for applications with 50 tasks at average load for 20 CNs

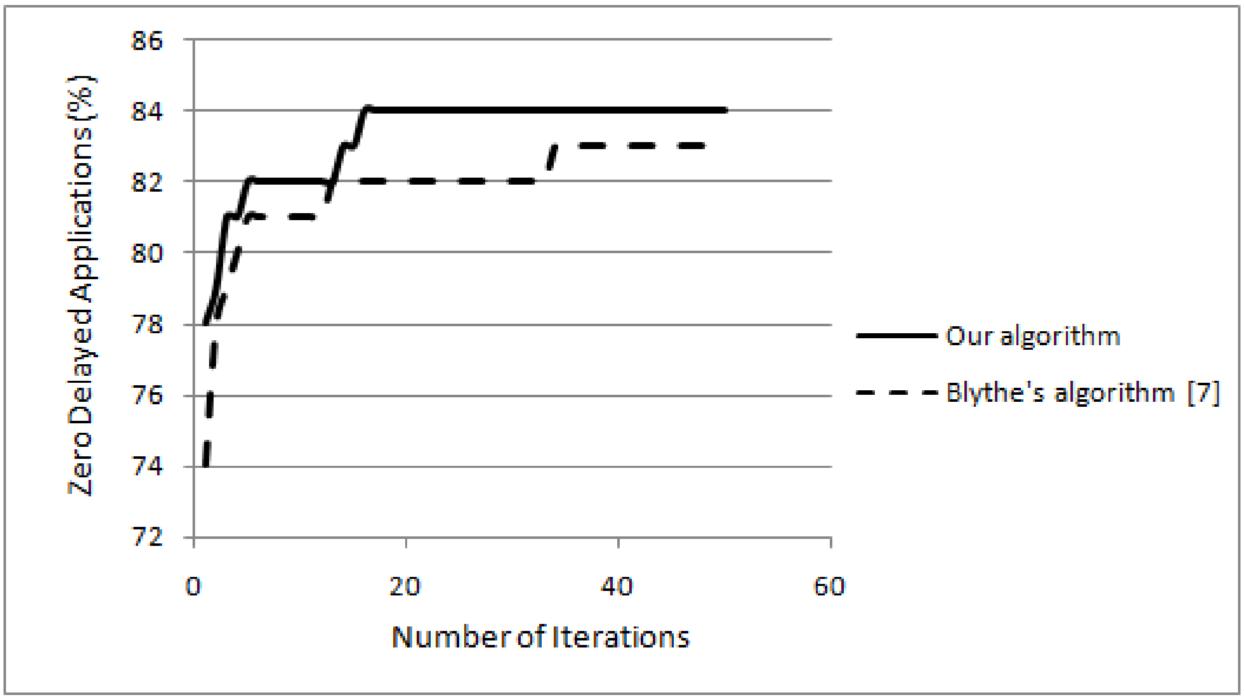

Figure 4.8: Percentage of zero delayed applications for applications with 100 tasks at average load for 20 CNs

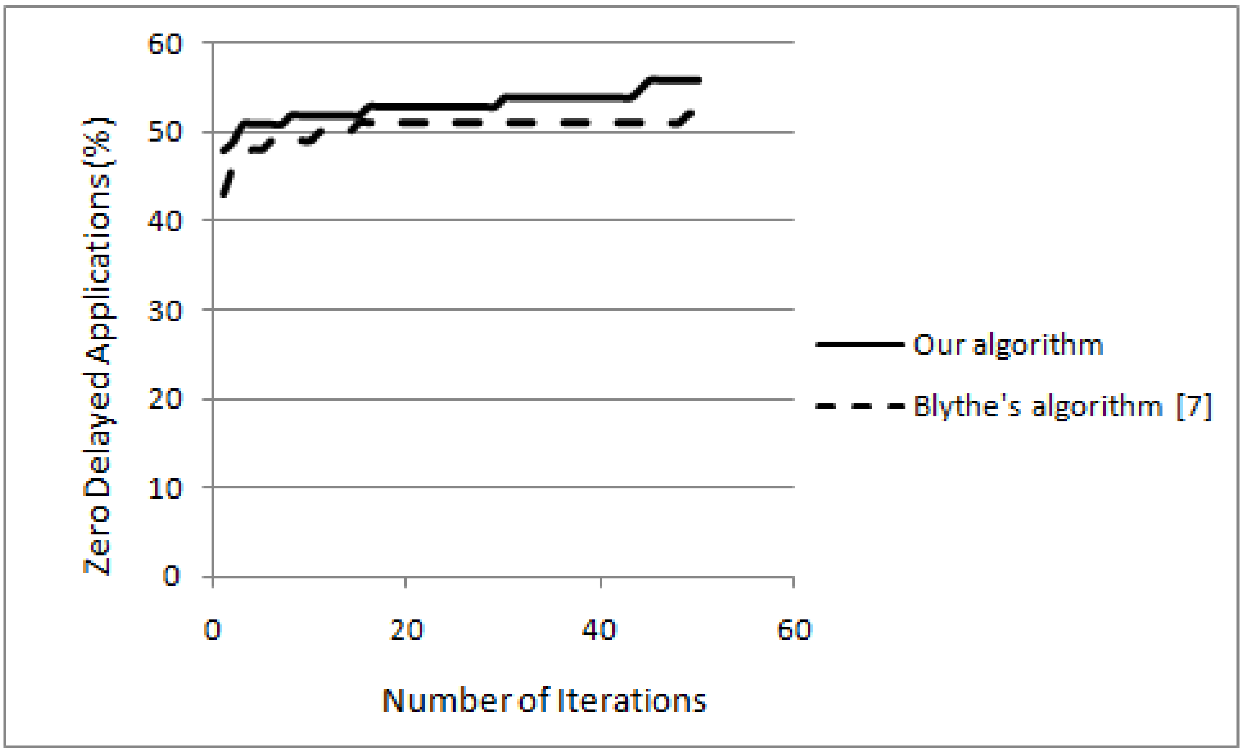

Figure 4.9: Percentage of zero delayed applications for applications with 300 tasks at average load for 20 CNs

less number of iterations than Blythe's algorithm does. As stated earlier, delay of any task in a critical path results delay of the whole application. So giving more priority to tasks with less slack time for scheduling is expected to produce better result. Our algorithm takes this approach and simulation results show betterness of our algorithm.

We have set the number of iterations to 50 which is justified by the results shown in Figures 4.7- 4.12. The more the number of iterations the better the result is, but more time is required. So, the number of iterations should be fixed based upon the time available for computing the scheduling algorithm, the number of applications in the queue waiting for reservation, expected delay to complete execution of the application etc.

The average time required to run the scheduling algorithms for an application for different number of CNs at average load has been shown in Figures 4.13, 4.14 and 4.15 for different application sets respectively. Here the figures show that our algorihm takes less time to schedule an application than that of Blythe's algorithm. In Blythe's algorithm, all the CNs are searched against all the ready tasks to find out Estimated Completion Time (ECT) of each of the tasks. Then a small pool of tasks with smaller ECT is created and a task is randomly selected from the pool. But in our algorithm, CNs are not considered during selection of a task. A task is selected

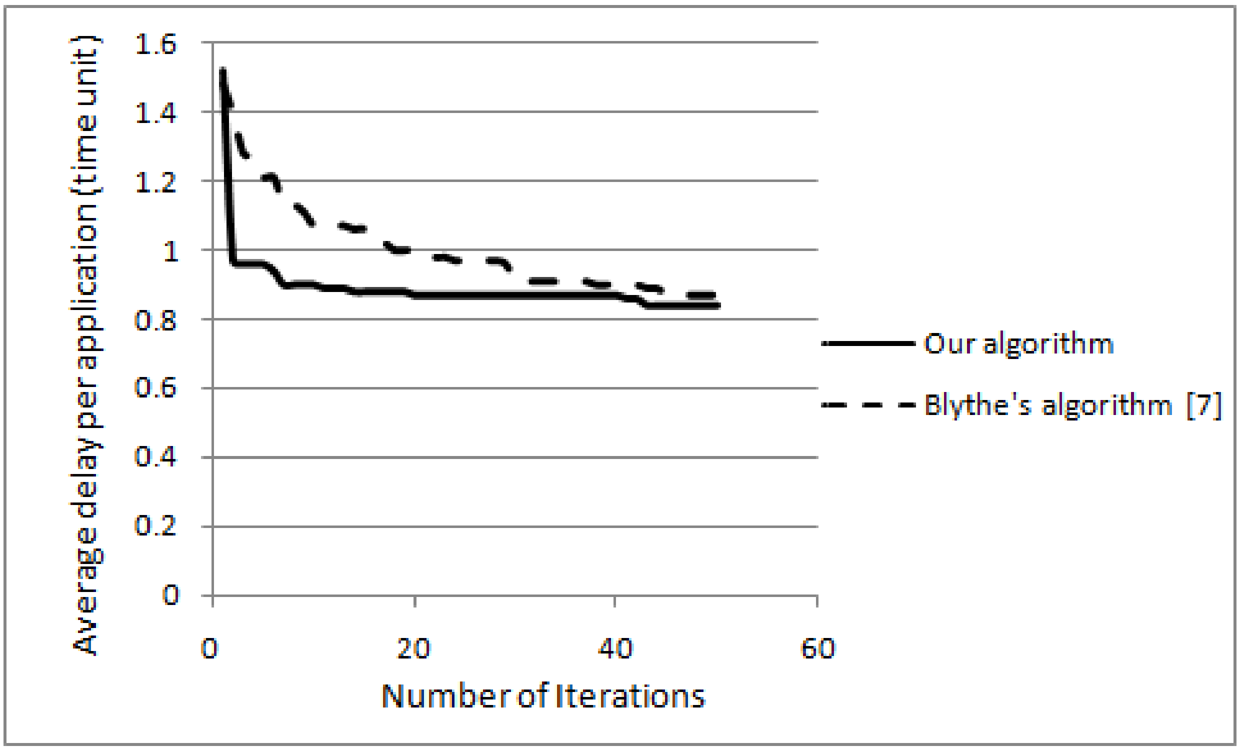

Figure 4.10: Average delay for applications with 50 tasks at average load for 20 CNs

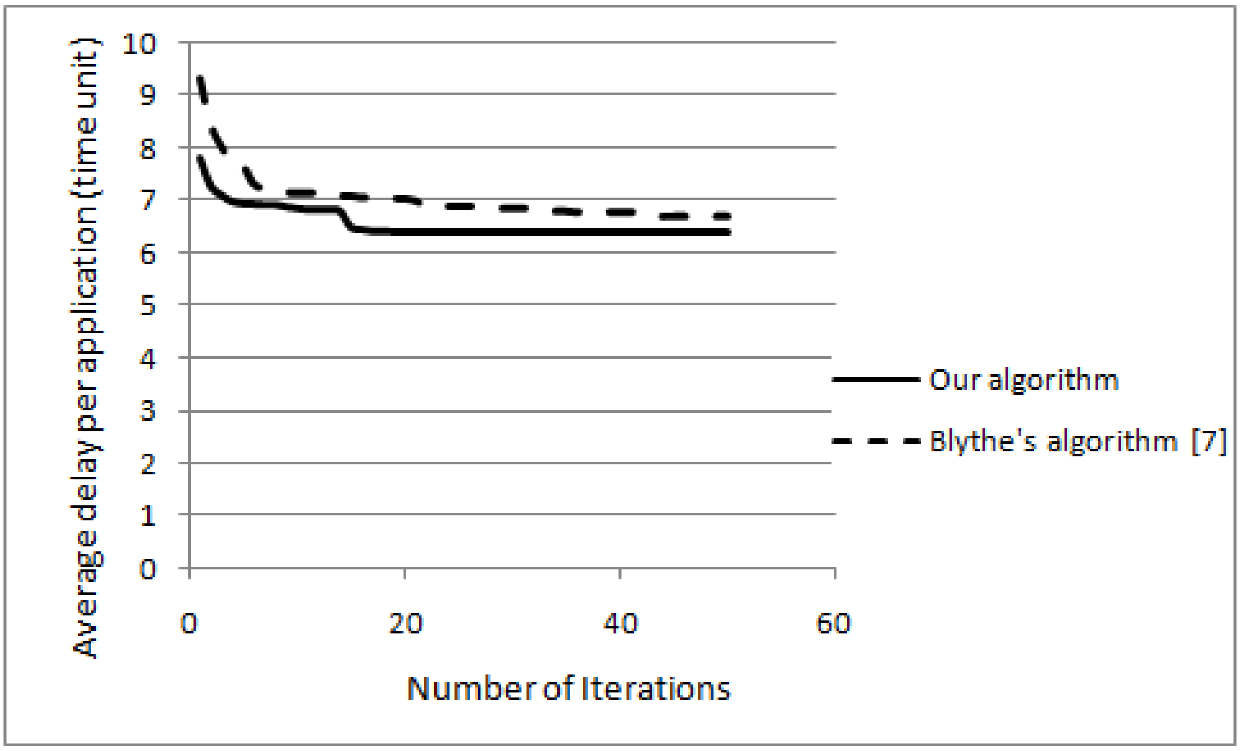

Figure 4.11: Average delay for applications with 100 tasks at average load for 20 CNs

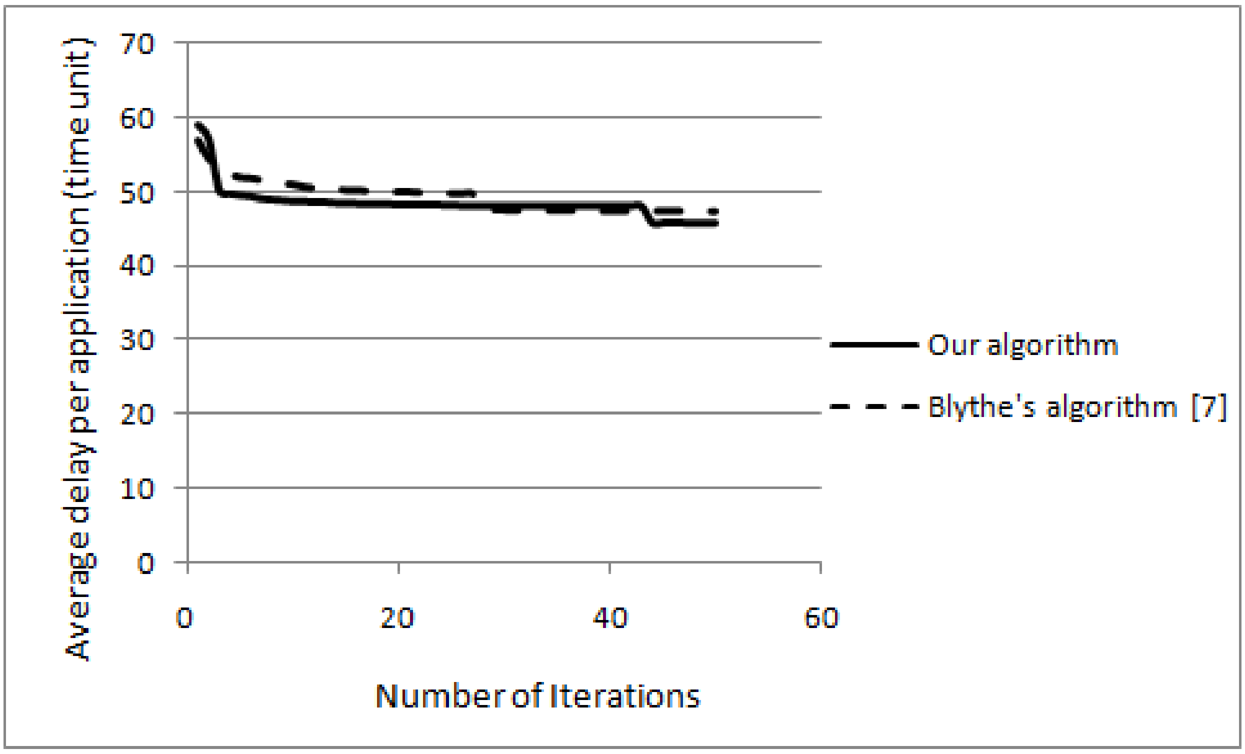

Figure 4.12: Average delay for applications with 300 tasks at average load for 20 CNs

based on the slack time of the ready tasks only. After selecting the task, our algorithm searches for the CNs to schelue the task at its earliest possible. Also, another reason is that our algorithm takes less iterations to produce a zero delayed application schedule. Zero delayed application scheduing is best and so no more iterations is required when a zero delayed scheduling is obtained. As a result our algorithm takes less time on average to scheule an application than that of Blythe's algorithm. Also from the complexity analysis, as shown in previous chapter, it is obvious that our algorithm is faster than Blythe's algorithm.

The results also show that our algorithm takes less time as the number of CNs increases. This is due to the fact that in our algorithm selection of a task does not depend on the CNs, but more CNs result more zero delayed applications in less number of iterations. So, average time required to schedule an application is reduced. Though more CNs result more zero delayed applications in less number of iterations for Blythe's algorithm also, the time is not reduced as like in our algorithm. The reason is that in Blythe's algorithm selection of each task for scheduling depends on the CNs. So if the number of CNs increases, the time required by Blythe's algorithm increases. But the time required for 15 CNs is less than 10 CNs. This is due to the fact that the number of zero delayed applications for 10 CNs is much less than 15 CNs. As stated earlier, less number of zero delayed applications indicates more applications require all the iterations. So, the average

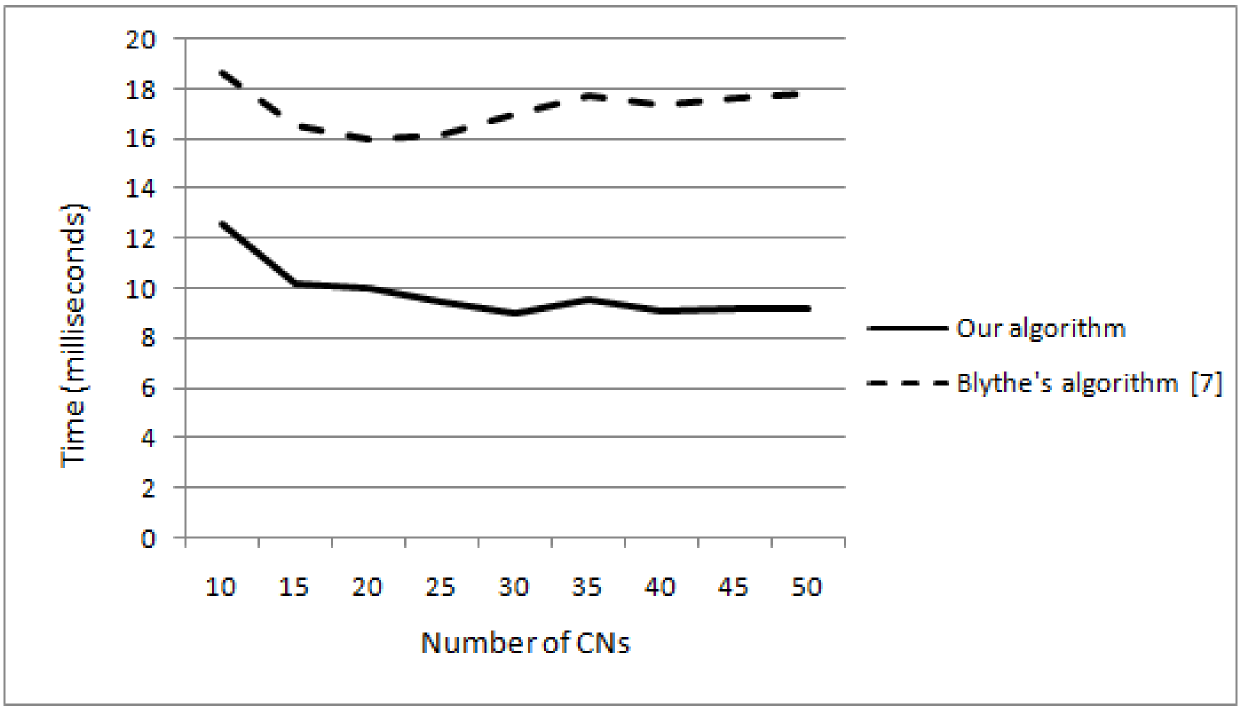

Figure 4.13: The average time required to schedule an application with 50 tasks at average load

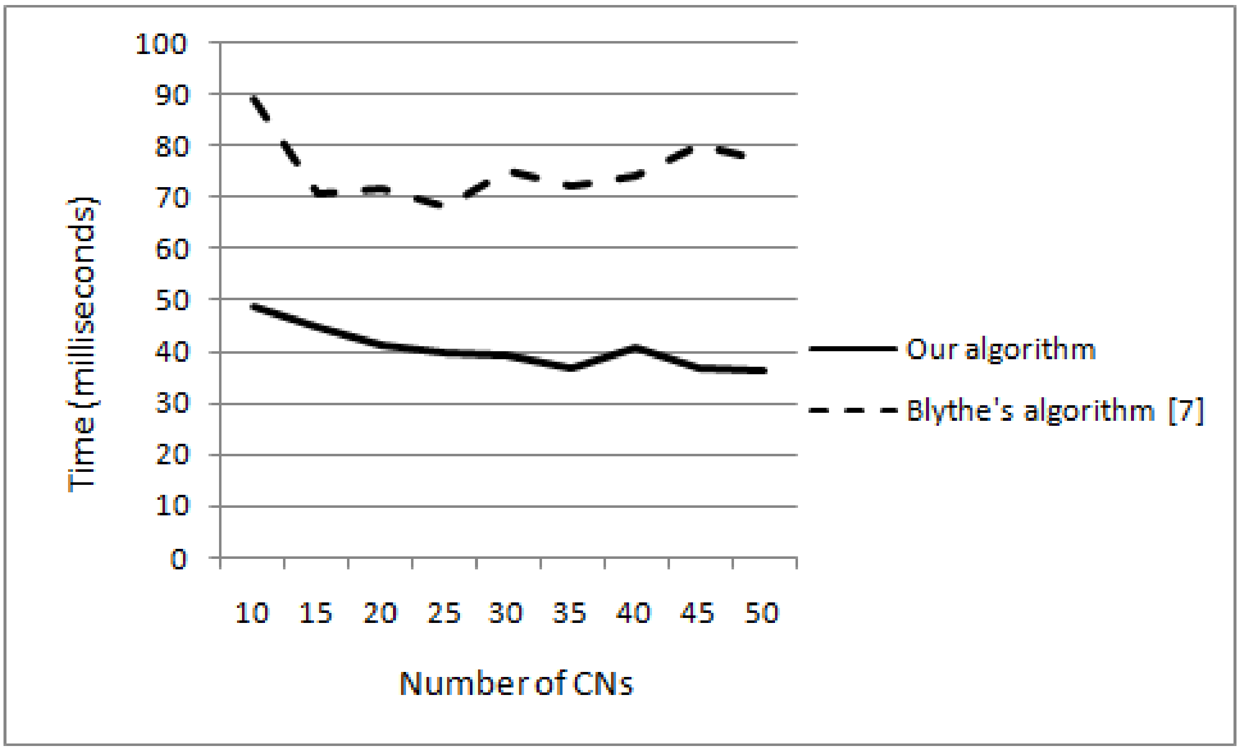

Figure 4.14: The average time required to schedule an application with 100 tasks at average load

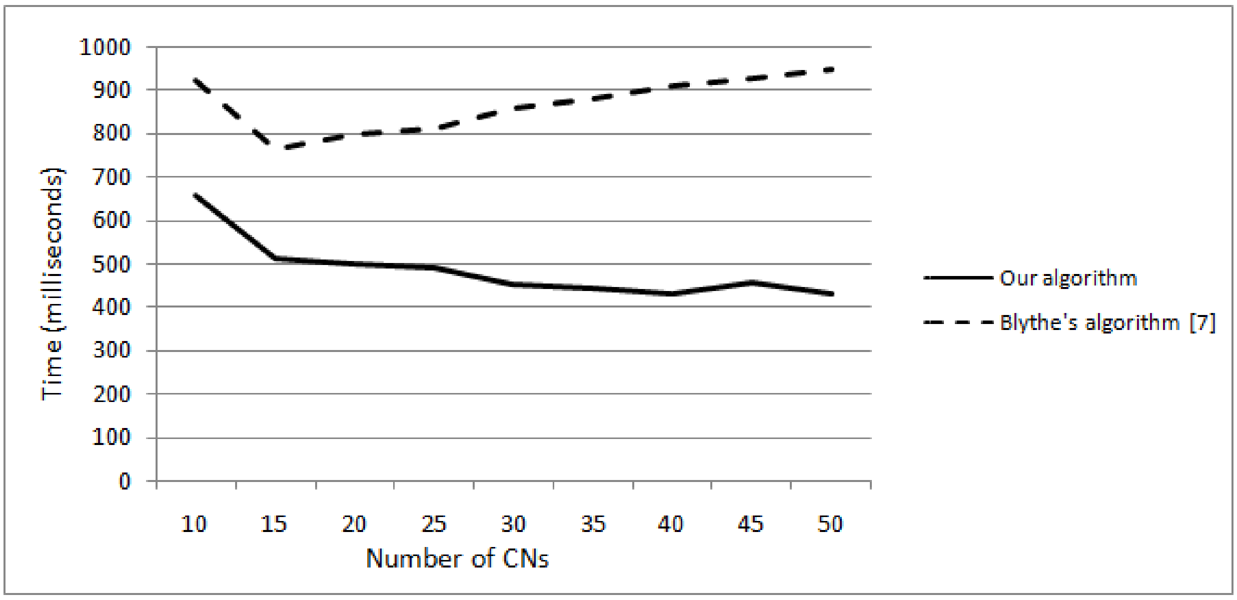

Figure 4.15: The average time required to schedule an application fwith 300 tasks at average load

time required for 10 CNs is more than 15 CNs. Also we see that the average run time for our algorithm does not decreases steadily. This is same for Blythe's algorithm. For Blythe's algorithm increment of time is not steady too. We can see from Figures 4.1, 4.2 and 4.3 that the number of zero delayed applications does not change significantly as the number of CNs increases. For the cases, where applications take more iterations to produce zero delayed schedule, more average time is required.

We performed a paired t-test on the average delay generated by our algorithm and Blythe et el. As a sample, the test was performed on average delay of applications with 100 tasks for average load and 20 CNs. The two-tailed P value for the average delays after iteration 1, 25 and 50 are 0.0187, 0.0105 and 0.0483 respectively. The result shows that the differences are statistically significant. The mean of our algorithm minus Blythe's equals 1.54 after first iteration. 95% confidence interval of this difference is: From 0.26 to 2.82. After iteration 25, the difference is 0.49. 95% confidence interval of this difference is: From 0.12 to 0.86. The difference is 0.3 after iteration 50. 95% confidence interval of this difference is: From 0.01 to 0.61. From this test we can confidently claim that our algorithm is consistent relative to Blythe's algorithm.

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 major 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 reservation system for grid computing that reserves resources for applications with less average delay and less compution time compared to others. Also the number of applications that are scheduled with zero delay is more in our system than the system we compared with.

- The system 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 application.

- The number of iterations is flexible. System will set the number of iterations required based on time available to compute the schedule, the performance required, the number of applications in the queue for reservation etc. So,the system will produce scheduling as required based on the factors mentioned earlier.

- We performed the detailed performance evaluation of our resource reservation system and compared with an existing system using applications provided by Standard Task Graph

Archive. The performance of our system in terms of total number of zero delayed applications and average delay 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 number of zero delayed applications, average delay and iterations required.

5.2 Directions of Further Research

Any research on any topic always makes a way to further research. Ours is not an exception. Resource reservation 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 reservation in grid are given below

- Reserving multiple resources like processor time, memory, banwidth for applications. The resources may be homogenious or heterogenious.
- Reserving resources for data intensive applications and thus design the system considering the data transfer delay for a task from a resource provider to other resource for its child tasks.
- Rearrange the reservation for tasks to accomodate a new application with reduced delay. In such case the arrangment of the tasks will maintain the dependency and also should not increase the delay of any application for which resources have been reserved.

Bibliography

[1] SimJava: a discrete event and process oriented simulation package. `http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/`.

[2] Simscript: a simulation language for building large-scale complex simulation models. `http://www.simscript.org`.

[3] AccessGrid. `http://www.accessgrid.org/`.

[4] D. Anderson, J. Cobb, and E. Korpela. Seti@home: An experiment in public-resource computing. *Communication of the ACM*, 45(11):56–61, November 2002.

[5] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster. Applying chimera virtual data concepts to cluster finding in the sloan sky survey. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Baltimore, MD, U.S.A., November 2002. IEEE Computer Society Press.

[6] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Milar, K. Stokinger, and F. Zini. Simulation of dynamic grid replication strategies in optorsim. In *Proceedings of the 3rd International Workshop on Grid Computing (Grid'02)*, Baltimore, USA, November 2002.

[7] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, volume 2, 2005.

[8] H. Casanova, A. Legrand, and M. Quinson. Simgrid: a generic framework for large-scale distributed experimentations. In *Proceedings of the 10th International Conference on Computer Modeling and Simulation (UKSim'08)*, Cambridge, UK, April 2008.

[9] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *9th Heterogeneous Computing Workshop*, Cancun, Mexico, May 2001.

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

[11] W. Cirne, D. Paranhos, L. Costa, E. S. Neto, F. Braseleiro, and J. Sauve. Running bag-of-tasks applications on computational grids: The mygrid approach. In *International Conference on Parallel Processing (ICPP'03)*, Kaohsiung, Taiwan, October 2003.

[12] E. G. Coffman and J. L. Bruno. *Computer and job-shop scheduling theory*. John Wiley and Sons Inc, USA, 1976.

[13] C. L. Dumitrescu and I. Foster. Gangsim: A simulator for grid scheduling studies. In *Proceedings of the 5th International Symposium on Cluster Computing and the Grid (CCGrid'05)*, Cardiff, UK, May 2005.

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

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

[16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Supercomputer Applications*, 11(2), 1997.

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

[18] U. Hoenig and W. Schiffmann. A comprehensive test bench for the evaluation of scheduling heuristics. In *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS'04)*, Cambridge, USA, November 2004.

[19] F. Howell and R. McNab. simjava: a discrete event simulation package for java with applications in computer systems modelling. In *Proceedings of the First International Conference on Web-based Modelling and Simulation*, San Diego, CA, USA, January 1998. Society for Computer Simulation.

[20] B. Jacob, M. Brown, K. Fukui, and N. Trivedi. *Introduction to Grid Computing*. IBM Corporation, North Castle Drive Armonk, NY, U.S.A., 2005.

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

[22] Kasahara Laboratory. `http://www.kasahara.elec.waseda.ac.jp/schedule/`.

[23] J. MacLaren. Advance reservations: State of the art (draft). GWD-I, Global Grid Forum (GGF), June 2003.

[24] Ns-2 network simulator. `http://www.isi.edu/nsnam/ns`.

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

[26] A. Schill, F. Breiter, and S. Kuhn. Design and evaluation of an advance reservation protocol on top of rsvp. In *Proceedings of the IFIP TC6/WG6.2 4th International Conference on Broadband Communications (BC'98)*, Stuttgart, Germany, April 1998.

[27] SETI@home. `http://setiathome.berkeley.edu/sah_about.php`.

[28] R. Shahriyar. A distributed optimized resource reservation scheme for grid computing. In *M.Sc.Engg. Thesis*. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, December 2009.

[29] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'00)*, Cancun,Mexico, May 2000.

[30] H. J. Song, X. Liu, D. Jackobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The microgrid: A scienti

c tool for modeling computational grids. *Scientific Programming*, 8(3), 2000.

[31] A. Sulistio, U. Cibej, S. K. Prasad, and R. 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.

[32] M. Viera, C. Rocha, M. Capretz, M. Bauer, and M. Dantas. A user-centric authentication for advanced resource reservation in mobile grid environments. In *International Conference on Grid Computing and Applications (GCA'10)*, Las Vegas, USA, July 2010.

[33] J. Xing, C. Wu, M. Tao, L. Wu, and H. Zhang. Flexible advance reservation for grid computing. In *Grid and Cooperative Computing(GCC)*, pages 241–248, 2004.

[34] K. Yang, X. Guo, A. Galis, B. Yang, and D. Liu. Towards efficient resource on-demand in grid computing. *ACM SIGOPS Operating Systems Review*, 37(2):37–43, 2003.

Appendix A

Major Algorithms

A.1 Algorithms realted to data structures

Algorithm 7: CreateNode(int start, int duration)

Output: Creates and returns a single node for reservation status list

```
1 $n \leftarrow new\ node\ for\ reservation\ status\ list$
2 $n.startTime \leftarrow start$
3 $n.duration \leftarrow duration$
4 $n.next \leftarrow null$
5 **return** $n$
```

Algorithm 8: InsertNode(Node n, ComputingNode CN)

```
Output: Insert a single node n in the reservation status list of CN
if $CN.reservationList\ is\ null$ then
    $add\ n\ to\ CN.reservationList$
    $return$
endif
$nd \leftarrow first\ node\ of\ CN.reservationList$;
if $n.startTime < nd.startTime$ then
    $n.next \leftarrow nd$
    $add\ n\ to\ CN.reservationList$
    $return$
endif
while $nd.next \neq null$ AND $nd.next.startTime < n.startTime$ do
    $nd \leftarrow nd.next$
endw
$n.next \leftarrow nd.next$
$nd.next \leftarrow n$
```

Algorithm 9: SearchTime(Node n, ComputingNode CN)

Output: Returns the earliest possible time in the reservation status list of CN for node n

```
1  $nd \leftarrow first\ node\ of\ CN.reservationList$
2  **if** $CN.reservationList\ is\ null$ *OR* $n.startTime + n.duration \leq nd.startTime$ **then**
3      $return\ n.startTime$
4  **endif**
5  $lastNd \leftarrow last\ node\ of\ CN.reservationList$
6  **if** $n.startTime \geq lastNd.startTime + lastNd.duration$ **then**
7      $return\ n.startTime$
8  **endif**
9  **while** $nd.next \neq null$ **do**
10     **if** $nd.next.startTime$ - $(nd.startTime + nd.duration) \geq n.duration$ *AND*
       $nd.next.startTime - n.startTime \geq n.duration$ **then**
11         **if** $n.startTime \geq nd.startTime + nd.duration$ **then**
12             $return\ n.startTime$
13         **endif**
14         **else**
15             $return\ nd.startTime + nd.duration$
16         **endif**
17     **endif**
18     $nd \leftarrow nd.next$
19 **endw**
20 **return** $nd.startTime + nd.duration$
```

Appendix B

Simulation Outputs

The simulation has been performed on different loads of the computing nodes. The different loads for which the test applications have been run are 0%(Completely free), 25%(lightly loaded), 50% (Average Loaded) and 75% (Higly loaded). Different numbers of computing nodes were also used in the simulation. The numbers of nodes for different runs were 10, 15, 20, 25, 30, 35, 40, 45, 50. So, there are 3 different task sets, 4 different load and 9 different size of computing node set which results a total of $3 \times 4 \times 9 = 108$ runs. We have run the simulation both with our proposed algorithm and with the algorithm presented by Blythe et al. [7]. The details output of simulation is presented in Table B.1, B.2 and B.3 for applications with 50 tasks, 100 tasks and 300 tasks respectively.

Table B.1: Output for the applications with 50 tasks

Number of Compiuting Nodes	Load	Application Delayed(%)		Average Delay	
		Ours	Blythe's	Ours	Blythe's
10	0	15	18	0.56	1.15
10	25	34	39	6.21	6.94
10	50	39	41	16.22	16.53
10	75	59	65	34.75	36.18
15	0	0	1	0	0.03
15	25	4	6	0.34	0.41
15	50	16	17	1.51	1.73
15	75	23	23	5.58	5.83
20	0	0	0	0	0
20	25	0	0	0	0
20	50	12	12	0.84	0.87
20	75	12	12	4.28	4.29
25	0	0	0	0	0
25	25	0	0	0	0
25	50	4	4	0.42	0.42
25	75	10	11	1.86	2.12
30	0	0	0	0	0
30	25	0	0	0	0
30	50	0	0	0	0
30	75	2	2	0.07	0.07
35	0	0	0	0	0
35	25	0	0	0	0
35	50	1	1	0.02	0.02
35	75	2	2	0.53	0.53
40	0	0	0	0	0
40	25	0	0	0	0
40	50	0	0	0	0
40	75	0	0	0	0
45	0	0	0	0	0
45	25	1	1	0.03	0.03
45	50	0	0	0	0
45	75	1	1	0.06	0.06
50	0	0	0	0	0
50	25	0	0	0	0
50	50	0	0	0	0
50	75	0	0	0	0

Table B.2: Output for the applications with 100 tasks

Number of Compiuting Nodes	Load	Application Delayed(%)		Average Delay	
		Ours	Blythe's	Ours	Blythe's
10	0	20	32	2.14	3.18
10	25	40	45	19.08	20.31
10	50	63	67	72.47	76.05
10	75	77	78	141.57	143.23
15	0	5	7	0.63	1.01
15	25	26	27	8.94	9.38
15	50	38	42	22.31	23.28
15	75	56	56	64.22	65.96
20	0	2	3	0.06	0.36
20	25	3	3	0.77	0.77
20	50	16	17	6.4	6.7
20	75	27	29	10.67	12.87
25	0	0	1	0	0.07
25	25	4	7	0.31	0.54
25	50	8	10	1.4	1.73
25	75	7	12	2.43	2.52
30	0	0	0	0	0
30	25	0	1	0	0.02
30	50	2	3	0.2	0.28
30	75	15	16	3.9	4.1
35	0	0	0	0	0
35	25	0	0	0	0
35	50	3	3	0.71	0.67
35	75	6	8	1.77	1.82
40	0	0	0	0	0
40	25	0	0	0	0
40	50	3	3	0.35	0.37
40	75	0	0	0	0
45	0	0	0	0	0
45	25	0	0	0	0
45	50	0	0	0	0
45	75	1	1	0.08	0.08
50	0	0	0	0	0
50	25	0	0	0	0
50	50	0	0	0	0
50	75	1	1	0.14	0.14

Table B.3: Output for the applications with 300 tasks

Number of Compiuting Nodes	Load	Application Delayed(%)		Average Delay	
		Ours	Blythe's	Ours	Blythe's
10	0	53	61	32.81	37.26
10	25	78	84	96.9	103.14
10	50	94	96	327.85	338.66
10	75	94	95	684.31	697.75
15	0	12	16	3.37	4.39
15	25	38	43	64.68	66.44
15	50	62	64	162.04	165.98
15	75	78	81	305.08	308.5
20	0	6	7	2.48	2.99
20	25	29	29	25.57	26.85
20	50	44	48	45.65	47.49
20	75	65	67	88.37	93.24
25	0	11	15	3.3	4.67
25	25	15	20	6.3	7.77
25	50	37	38	27.68	30.32
25	75	42	45	35.15	36.83
30	0	8	8	1.48	2.2
30	25	20	20	7.19	8.05
30	50	25	26	21.04	21.74
30	75	31	33	35.57	36.32
35	0	4	6	0.34	0.79
35	25	9	11	2.15	2.56
35	50	14	15	5.39	6.12
35	75	22	23	12.03	14.25
40	0	4	5	0.67	1.24
40	25	6	9	1.66	2.07
40	50	7	8	4.52	4.86
40	75	14	14	5.91	6.45
45	0	1	3	0.06	0.22
45	25	5	6	1.5	1.8
45	50	6	8	3.47	3.87
45	75	10	12	5.93	6.15
50	0	1	3	0.05	0.36
50	25	2	3	0.2	0.49
50	50	4	9	0.51	0.99
50	75	6	7	1.34	1.6