Makespan Optimization Algorithm for Scheduling in Grids

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

Master of Engineering
in
Software Engineering

Thapar University, Patiala

By:
Gurpreet Kaur
(80631005)

Under the supervision of:
Dr. Seema Bawa
Professor & Head
Computer Science & Engineering Department
Thapar University, Patiala, INDIA

MAY 2008

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004
INDIA

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Makespan Optimization Algorithm for Scheduling in Grids"**, in partial fulfilment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Seema Bawa and refers other researcher's works which are duly listed in the reference section.

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

(***Gurpreet Kaur)***

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

(**Dr. Seema Bawa)**
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(SEEMA BAWA)
Professor & Head
Computer Science & Engineering. Department
Thapar University
Patiala

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

Acknowledgement

First of all I would like to thank the Almighty, who has always guided me to work on the right path of the life. My greatest thanks are to my parents who bestowed ability and strength in me to complete this work.

This work would not have been possible without the encouragement and able guidance of my supervisor **Dr. Seema Bawa.** Her enthusiasm and optimism made this experience both rewarding and enjoyable. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. I am equally grateful to **Dr. Maninder Singh**, Assistant Professor, Computer Science & Engineering Department, a nice person, an excellent teacher and a well-credited researcher, who always encouraged me to keep going with work and always advised me with his invaluable suggestions.

I would like to express my sincere gratitude towards Ms. Anju Sharma, Ms. Shashi, full time research scholars and all the members of the **Centre of Excellence in Grid Computing** at Computer Science and Engineering Department, Thapar University for their unending support.

I am also very thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct–indirect help, cooperation, love and affection, which made my stay at Thapar University memorable.

(Gurpreet Kaur)

(80631005)

Abstract

Grid Computing is a service aggregation of both the information and the computational resources in heterogeneous environment. It allows the sharing of geographically distributed resources in an efficient way, extending the boundaries of what we perceive as distributed computing. Various sciences such as life sciences, e-sciences etc can benefit from the use of grids to solve computation intensive and data intensive problems thereby, creating potential benefits for the entire society. With further development of grid technology, it is very likely that enterprises, academic institutions and naïve users will exploit grids to enhance their computing infrastructure.

To manage the Grid resources efficiently, a Grid Resource Management System is required. It deals with the process of identifying requirements, matching resources to applications, allocating, monitoring and scheduling these resources. Due to dynamicity and heterogeneity of Grid resources as well as grid applications, scheduling becomes a challenging task. The presence of large number of resources and diverse set of jobs to be scheduled further makes scheduling, a complex optimization problem.

In this thesis an efficient scheduling algorithm is proposed, which optimizes the makespan of the job to be scheduled on Grid. The proposed algorithm efficiently schedules jobs having arbitrary inter-dependency constraints and arbitrary processing durations. A user-friendly interface has also been designed for the scheduler. The proposed algorithm is implemented using DAG (Directed Acyclic Graph) approach.

The Makespan Optimization Algorithm is simulated using Open Source GridSim Toolkit 4.1. The outcomes of the Algorithm simulation on GridSim Toolkit, under different test scenarios shows that, it optimizes makespan.

Organization of Thesis

The First Chapter briefly introduces Grid Computing concepts. It explains concept of Grid Computing with relevant examples, Building blocks of Grid, Benefits, Applications of Grid Computing and Grid Topologies.

The Second Chapter presents Review of Literature in Grid Scheduling. It covers Grid Application Considerations, Challenges in Grid Scheduling, Generic Grid Scheduler Architecture and Steps involved in Grid Scheduling. Further parameters to be considered while scheduling are also covered.

The Third Chapter covers Problem Formulation for this thesis work and explains the Proposed Solution Design. The Proposed Makespan Optimization Algorithm for Scheduling in Grids is presented with complete terminology and complexity analysis.

The Fourth Chapter explains the Experimental Setup and Implementation Details. It covers setting up the environment for simulation of proposed solution on GridSim Toolkit 4.1 with relevant screenshots at each point.

The Fifth Chapter presents the Experimental results obtained and inferences drawn henceforth.

Finally Thesis work has been concluded in Sixth Chapter along with mention of scope for future work.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Overview of Grid Computing

Grid computing is gaining a lot of attention within the IT industry. Grid computing is a distributed computing taken to the next evolutionary level. It provide the electronic foundation for a global society in business, government, research, science and entertainment .The goal is to create the illusion of a large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The Grid Computing integrates networking, communication, computation and information to provide a virtual platform for computation and data management in the same way that the Internet integrates resources to form a virtual platform for information.

Grid computing is analogous to a power grid. When you plug an appliance or other object requiring electrical power into a receptacle, the power is available. The vision of grid computing is similar. Individual users can access computers and data, or even contribute its resources to the existing grid, without having to consider location, operating system, account administration, and other details. The resources are not visible to the user, just as the consumer of electric power is unaware of how their electricity is being generated.

Grid infrastructure will provide us with the ability to dynamically link together resource as an ensemble to support the execution of large-scale, resource-intensive, and distributed applications [1].

Grid applications often involve large amounts of data and computing and often require secure resource sharing across organizational boundaries. Sharing resources over the Internet raises many technical problems, such as large pool of resources, the heterogeneity of platforms, the diversity in user behaviors and the lack of reliability.

In Grid systems, an important functionality is resource scheduling. Given a description of grid resources and jobs that are to be submitted to grid environment, the scheduler takes the decision about how jobs are to be assigned to resources.

In the next few years, the Grid holds the potential for fundamental infrastructure not only for e-Science but also for e-Business, e-Government, e-Science and e-Life. This emerging infrastructure will exploit the revolutions driven by Moore's law [2] for CPU's, disks and instruments as well as Gilder's law [3] for (optical) networks. Grid computing tries to bring, under one definitional umbrella all the work being done in the high performance, cluster, peer-to-peer, and Internet computing arenas.

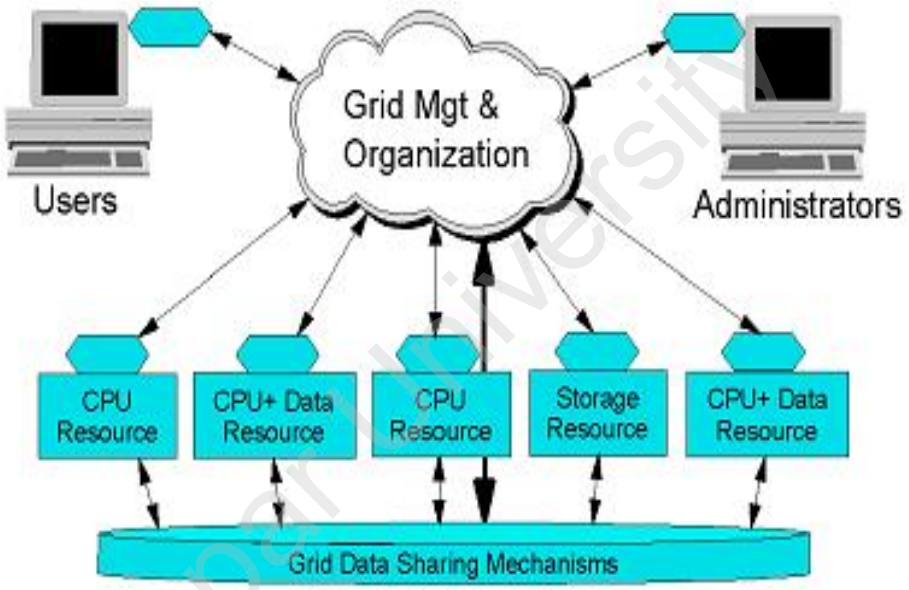


Figure 1.1: A Simple Grid [11].

The Need for Grid Technologies

- Increasingly complex problems
- Need for high computation and storage

By providing scalable, secure, high-performance mechanisms for discovering and negotiating access to remote resources, the Grid promises to make it possible for scientific collaborations to share resources on an unprecedented scale, and for geographically distributed groups to work together in ways that were previously impossible.

1.2 Building Blocks of The Grid

1.2.1 Networks

Networks form the heart of any Grid as it links together geographically distributed resources and allows them to be used collectively to support execution of a single application. High bandwidth networks enable applications to use distributed resources in a more integrated and data-intensive fashion.

1.2.2 Computational 'Nodes'

Networks connect heterogeneous resources on the Grid, computers being the most important among them with their associated data storage and computational power. Grid computing involves high-performance parallel machines or clusters that provide major resources for simulation, analysis, data mining and other compute-intensive activities.

1.2.3 Common Infrastructure: Standards

To allow incorporation of new technologies and software interoperability into Grid computing, standards are required. The Global Grid Forum (GGF) [4] is building key Grid-specific standards such as OGSA, the emerging de facto standard for Grid infrastructure. Also the Web and the W3C consortium [5] have defined key standards such as TCP/IP, HTTP, SOAP, XML and WSDL (Web Services Definition Language) that underlines OGSA.

1.3 Benefits of Grid Computing

1.3.1 Exploiting Underutilized Resources

In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5% of the time. Often, machines have enormous unused disk drive capacity. Grid computing provides a framework for exploiting these underutilized resources. Data grid can be used to aggregate the unused storage into a much larger virtual data store. Also a grid can provide a consistent way to balance the loads on a wider federation of resources.

1.3.2 Access to Additional Resources

Grid users can access resources that they might not normally have access to. For e.g., an organization might have a Grid machine connected to an electron microscope or a telescope. Some machines may have expensive licensed software installed that the user requires. Grid users who do not normally have access to such equipment can be given access to it, whether they belong to the organization or not [11].

1.3.3 Parallel Computing

Many industries and scientific communities require the use of parallel computing in order to run applications or solve certain problems. Grid Computing provides a framework that allows jobs to be split up into multiple sub jobs, and each sub-job can be made to execute in parallel on different machines in the Grid.

1.3.4 Virtual Resources and Virtual Organizations for Collaboration

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. The grid can help in enforcing security rules and implement policies, which can resolve priorities for both resources and users [11].

1.3.5 Resource Balancing

The Grid framework can be used to improve resource utilization. For e.g., jobs can be scheduled to run on idle machines or machines with low activity levels. Grids also offer load balancing. If jobs running on Grid require a high level of communication between each another, they could be scheduled in a manner that minimizes the cost of communication or the amount of traffic on their communication lines [11].

1.3.6 Reliability

The systems in a grid can be relatively inexpensive and geographically dispersed. Thus, if there is a power failure or other kind of failure at one location, the other parts of the grid are not likely to be affected. Grid management software can automatically resubmit jobs to other machines on the grid when a failure is detected. Jobs can even

be submitted multiple times in order to ensure that at least one copy of the job executes successfully.

1.4 Grid vs. other Related Technologies

1.4.1 Grid vs. Distributed Computing

Distributed computing is a subset of grid computing. The difference between Grid Computing and traditional distributed computing lies in computation size, scope, heterogeneity and communication. Grid Computing is distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

1.4.2 Grid vs. Cluster Computing

As clusters and Grids both operate on the same underlying principle that a group of computers acts as one but there are certain differences.

- In clusters, a centralized resource manager performs the resource allocation and all nodes work together cooperatively as a single unified resource. Within a Grid, each node has its own resource manager and provision of a single system view is not a goal.
- While Grids consist of heterogeneous resources, cluster computing is primarily concerned with homogeneous computational resources.
- Grid Computing integrates storage, networking, and computation resources. Clusters usually contain a single type of processor and operating system.
- Clusters contain a static number of processors and resources. Resources are provided and removed from the Grid on an ongoing basis.
- Clusters are homogeneous and in close physical proximity to one another, while Grids can be heterogeneous and geographically distributed [12].

1.5 Grid Architecture

The grid architecture defines the purpose and functions of its components, while indicating how these components interact with one another [13]. The main focus of the architecture is on interoperability among resource providers and users in order to establish the sharing relationships. This interoperability, in turn, necessitates common

protocols at each layer of the architectural model, which leads to the definition of a grid protocol architecture.

This protocol architecture defines common mechanisms, interfaces, schema, and protocols at each layer, by which users and resources can negotiate, establish, manage, and share resources. Figure 1.5 shows the component layers of the grid architecture and the capabilities of each layer. Each layer shares the behaviour of the underlying component layers.

1.5.1 Fabric Layer

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

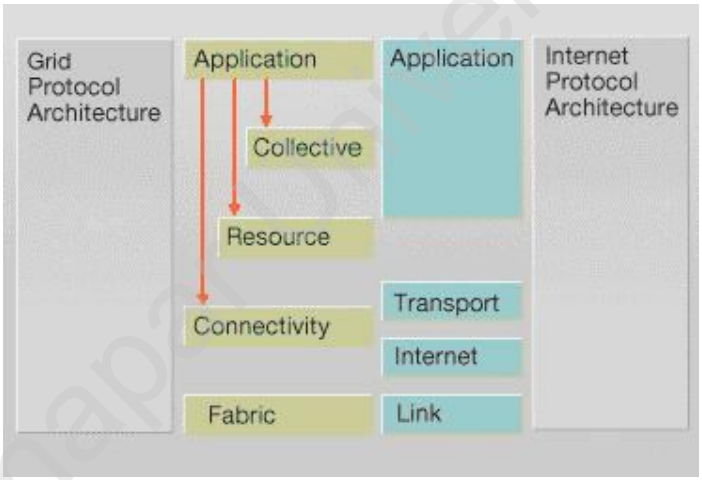


Figure 1.2: The Layered Grid Architecture [13]

1.5.2 Connectivity Layer

The connectivity layer defines the basic communication and authentication protocols required for grid-specific networking-service transactions.

1.5.3 Resource Layer

This layer uses the communication and security protocols (defined by the connectivity layer) to control secure negotiation, initiation, monitoring, accounting, and payment for the sharing of functions of individual resources. The resource layer calls the fabric

layer functions to access and control local resources. This layer only handles individual resources, ignoring global states and atomic actions across the resource collection pool, which are the responsibility of the collective layer.

1.5.4 Collective Layer

While the resource layer manages an individual resource, the collective layer is responsible for all global resource management and interaction with collections of resources. This protocol layer implements a wide variety of sharing behaviours using a small number of resource-layer and connectivity-layer protocols.

1.5.5 Application Layer

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

1.6 Grid Applications

The Grid serves as an enabling technology for a broad set of applications in science, business, entertainment, health and other areas. As we continue to develop the software infrastructure that better realizes the potential of the Grid, the application and user community for the Grid will continue to expand.

Life Science Applications

One of the fastest-growing application areas in Grid Computing is the Life Sciences. Computational biology, bioinformatics, genomics, computational neuroscience and other areas are implementing Grid technology to access, collect and mine data. For e.g. the Protein Data Bank [6], the myGrid Project [7], the Biomedical Information Research Network (BIRN) [8], accomplish large-scale simulation and analysis.

Engineering Oriented Applications

The Grid has provided an important platform for making resource-intensive engineering applications more cost-effective. The NASA IPG [9] in the United States is a large-scale engineering-oriented Grid application that provides a blueprint for

revolutionizing the way in which NASA executes large-scale science and engineering problems. Figure: 1.2 shows a set of Web (OGSA) services for satellite control, data acquisition, analysis, visualization and linkage with simulations.

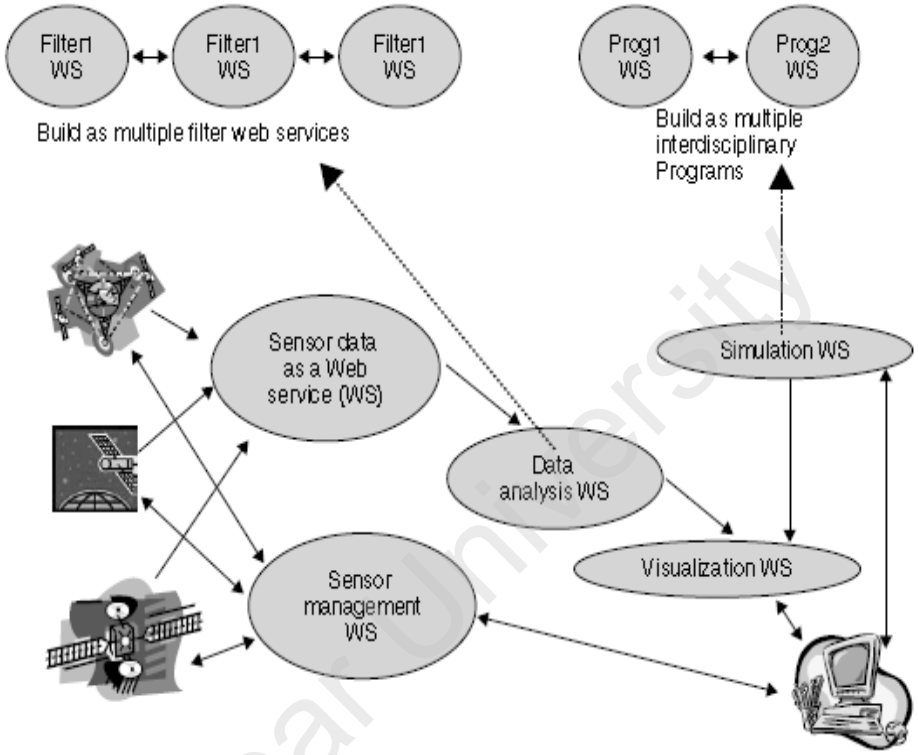


Figure 1.3: A Grid for Satellite Operation Showing Spacecraft Operation [9]

Data Oriented Applications

The Grid can be used to collect, store and analyze data and information, as well as to synthesize knowledge from data. An example of a data-oriented application is Distributed Aircraft Maintenance Environment (DAME) [10]. DAME is an industrial application being developed in the United Kingdom in which Grid technology is used to handle the gigabytes of in-flight data gathered by operational aircraft engines and to integrate maintenance, manufacturer and analysis centers. The project aims to build a Grid-based distributed diagnostics system for aircraft engines.

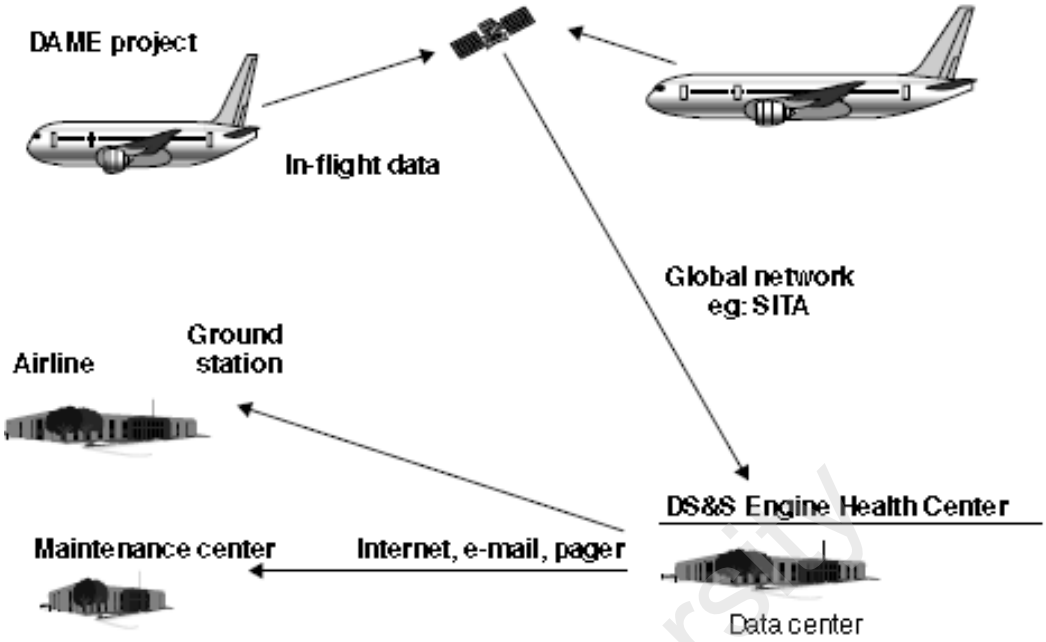


Figure 1.4: DAME Grid to Manage Data from Aircraft Engine Sensors [10].

Commercial Applications

In the commercial world, Grid, Web and distributed computing, and information concepts are being used in an innovative way in a wide variety of areas including inventory control, enterprise computing, games and so on.

Enterprise computing areas where the Grid approach can be applied include:

- End-to-end automation
- End-to-end security
- Virtual server hosting
- Disaster recovery
- Heterogeneous workload management
- End-to-end systems management
- Scalable clustering
- Accessing the infrastructure
- 'Utility' computing
- Accessing new capability more quickly
- Better performance
- Reducing up-front investment

• Gaining expertise not available internally

• Web-based access (portal) for control (programming) of enterprise function.

Trends in research: e-Science in a collaboratory

E-Science captures the new approach to science involving distributed global collaborations enabled by the Internet and using very large data collections and high-performance visualizations. E-Science is about global collaboration in key areas of science, and the next generation of infrastructure, namely the Grid, that will enable it. Figure 1.4 summarizes the e-Scientific method [17].

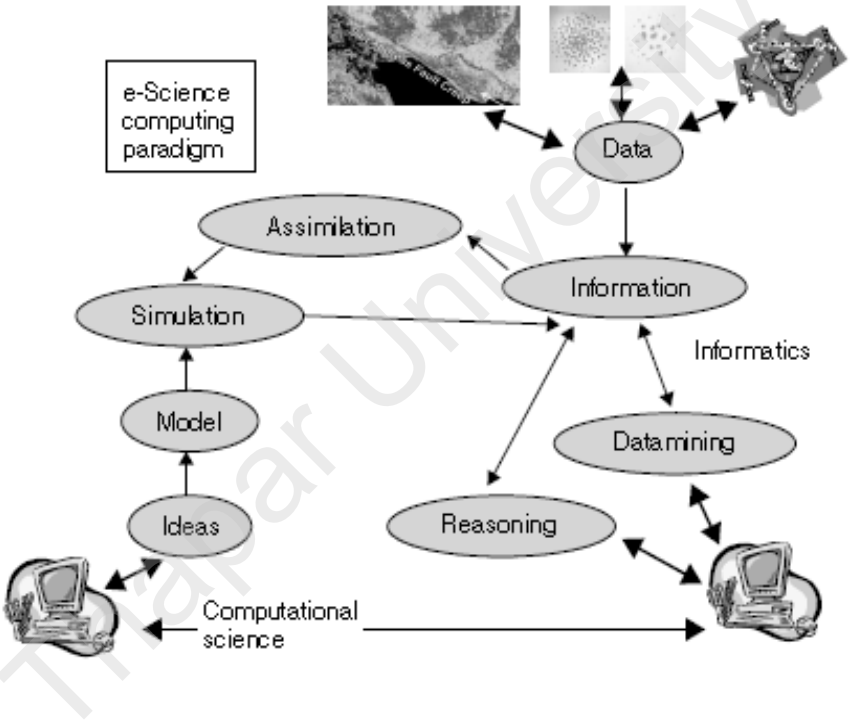


Figure 1.5: Computational Science and Information Technology merge in e-Science [17].

1.6.1 Grid Application Considerations

Grid applications can be categorized in one of the following three categories:

- Applications that are not enabled for using multiple processors but can be executed on different machines.
- Applications that are already designed to use the multiple processors of a grid.
- Applications that need to be modified or rewritten to better exploit a grid.

1.6.1.1 CPU Considerations for Grid Application

High Performance Computing clusters are sometimes used to handle the execution of applications consume large amounts of CPU time and that can utilize parallel processing, grids provide the ability to run these applications across a set of heterogeneous, geographically disperse set of clusters. If the algorithm is such that each computation depends on the prior calculation, then a new algorithm would need to be found. Not all problems can be converted into parallel calculations [15].

For e.g., take the process of adding up a large list of numbers. The simple serial program may be written to add numbers. Here each calculation depends on prior one. We could break the list up into seven pieces, for e.g., with seven separate programs adding up the numbers in each list, and then a final eighth program adding the seven sums to form the final answer.

Some computations cannot be rewritten to execute in parallel. For example, in physics, there are no simple formulas that show where three or more moving bodies in space will be after a specified time when they gravitationally affect each other.

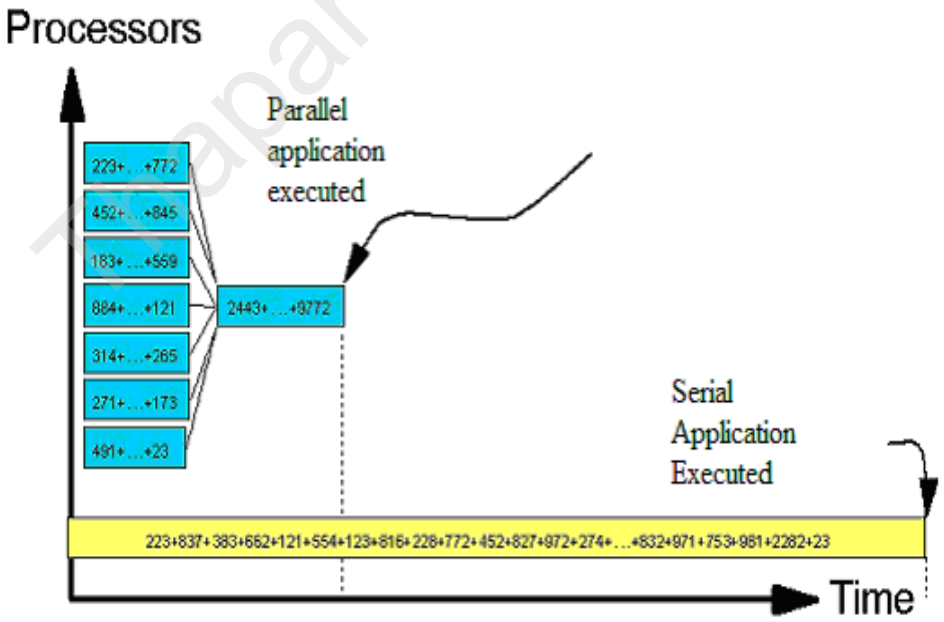


Figure 1.6: Rearranging computations to execute in parallel [15]

These kinds of computations are done by applying Newton's laws to small time increments, and computing how the forces and bodies affect each other, given the new position of the objects after each tiny time increment. Because the time increments are not infinitely small, after many increments, the small errors start adding up.
An application may be a mix of independent and dependent computations. One needs to analyze the application to see if there is a way to split some subset of the work.

Another approach to reducing data dependency on prior computations is to look for ways to use **redundant computations**. If the dependency is on a subset of the prior computations, we may recompute those results instead of waiting for them to arrive from another job. If the dependency is on a computation that has a yes/no answer, it is better to compute the next calculations for both of the "yes" and "no" cases and throw away the wrong choice when the dependency is finally known, as illustrated in figure. This technique can be taken to extremes in various ways.

Here **heuristics** (rules of thumb) could be developed to make the best possible guesses.

By altering some potentially unimportant rules in the computations involved in a calculation, we may be able to break the ordering requirement and thus make it possible to execute more of the application in parallel.

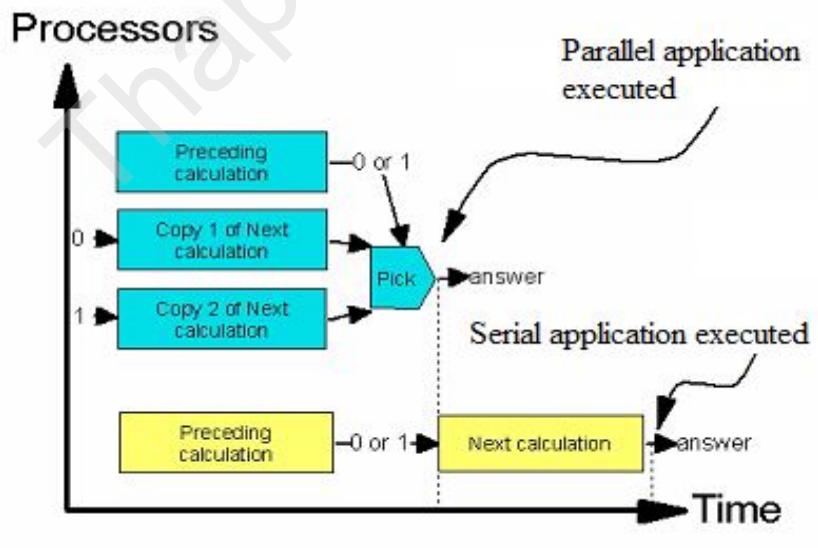


Figure 1.7: Redundant Speculative Computation to Reduce Latency [15]

For example, in a bank account, deposits and withdrawals are serially calculated and if the account ever goes negative, then the transaction may be rejected, a fine may be imposed, or the account may be frozen. If, however, the bank changes its rules and says that the account must simply be positive at the end of the day. Then withdrawals processed before the deposits would not cause a problem and all of these calculations could be broken up into separate parallel running jobs.

Another approach for grid enabling an application is to revisit the choices made when the application was originally written. Saving the results of computation performed more than once using the same data will reduce significant portion of the overall work.

In a distributed application, partial results or data dependencies may be met by **communicating among sub jobs**. One job may compute some intermediate result and then transmit it to another job in the grid.

1.6.1.2 Data Considerations for Grid Applications

When splitting applications for use on a grid, it is important to consider the amounts of data that are needed to be sent to the node performing a calculation and the time required to send it. If the application can be split into small work units requiring little input data and producing small amounts of output data, that would be most ideal. The data in this kind of case is said to be **"staged"** to the node doing the work. Sending this data along with the executable file to the grid node doing the work is part of the function of most grid systems.

The goal is to locate the shared data closer to the jobs that need the data. If the data is going to be used more than once, it could be replicated to the degree that space permits. One should exercise caution not to create situations, which might cause a synchronization deadlock with two sub jobs waiting for each other to unlock a resource, the other needs.

There are three ways that are usually used to prevent this problem.

- Waits for resources to include time-outs. If the time-out is reached, then the operation must be undone and started over in an attempt to have better luck at completing the transaction.

- Lock all of the resources in a predefined order ahead of the operation. If all of the locks cannot be obtained, then any locks acquired should be released and then, after an optional time period, another attempt should be made.

Use deadlock detection software. A transitive closure of all of the waiters is computed before placing the requesting task into a wait for the resource. If it would cause a deadlock, the task is not put into a wait. The task should release its locks and try again later. If it would not cause deadlock, the task is set to automatically wait for the desired resource [15].

1.7 Grid Types

Grids can be classified on the basis of two parameters

- Grid Functionality
- Grid Topology

1.7.1 Grids as Classified by Functionality:

According to the distinctly targeted application realms, Grid systems can be classified into two categories [14]. But there are actually no hard boundaries between these Grid categories. Real Grids may be a combination of two of these types. The two categories of Grid systems are described below:

Computational Grid

Computational Grids can be recognized by these primary characteristics:

- Made up of clusters of clusters.
- Enables CPU scavenging to better utilize resources.
- Provides the computational power to process large-scale jobs to satisfy the business requirement for instant access to resources on demand

Data Grid

Data Grids focus on providing secure access to distributed and heterogeneous pools of data. For applications that require access to large amounts of data often terabytes of data, data Grids are used.

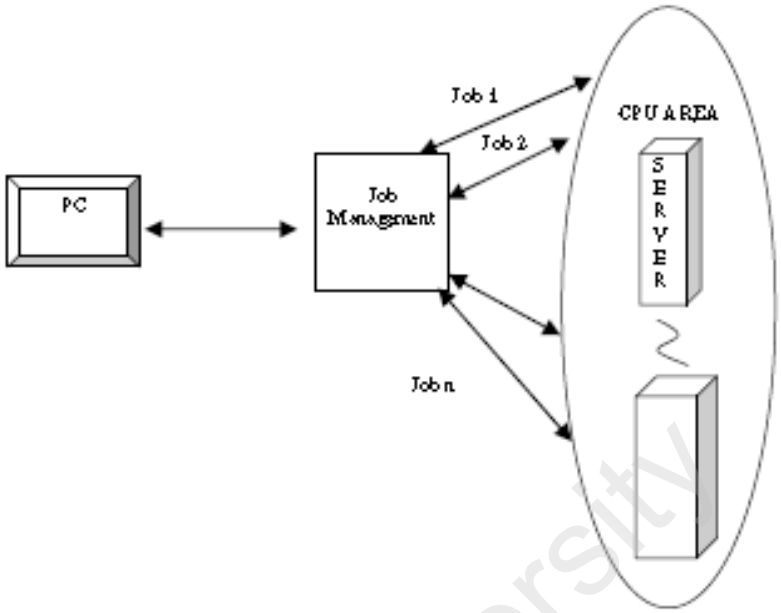


Figure 1.8: Visualization of a Computational Grid [27]

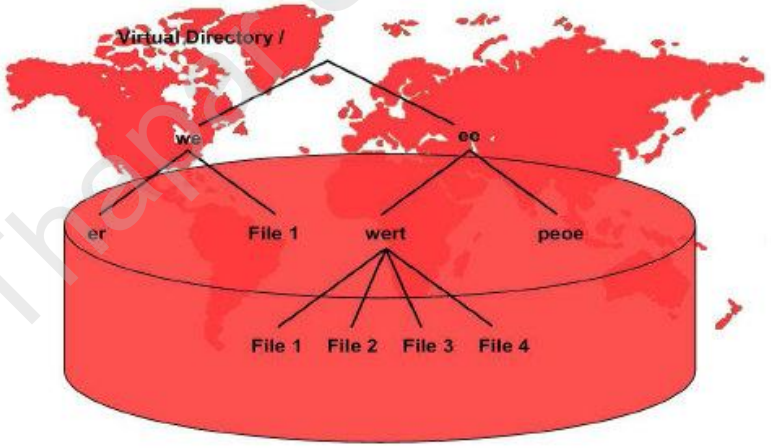


Figure 1.9: Visualization of a Data Grid [27]

1.7.2 Grids as Classified by Topology:

Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as hierarchy spanning the world. Grids can be classified into three categories according to the topology of Grid [14].

IntraGrid	ExtraGrid	InterGrid
Single organizations	Multiple organizations	Many organizations
No partner integration	Partner integration	Multiple partners
A single cluster	Multiple clusters	Many multiple clusters

Table 1.1: Grids as Classified by Topology

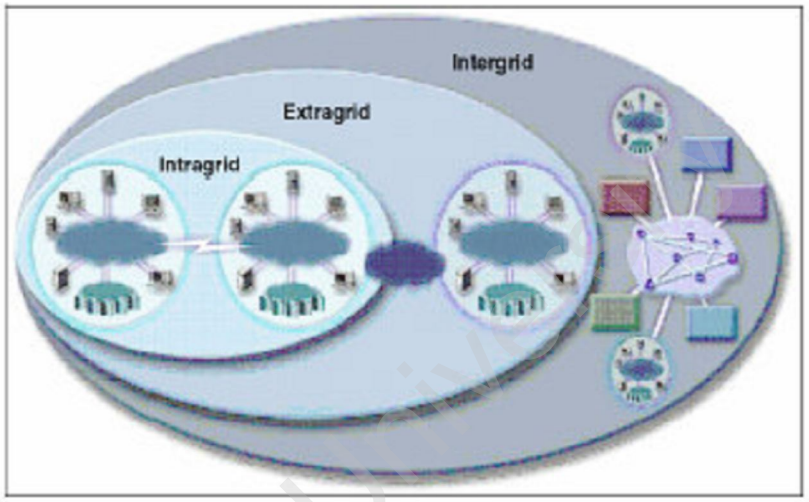


Figure 1.10: Grid Types [11]

1.8 Grid Components

In this section, we describe at a high level the primary components of a grid environment.

Portal / User Interface

A grid portal provides the interface for a user to launch applications that will use the resources and services provided by the grid. From this perspective, the user sees the grid as a virtual computing resource just as the consumer of power sees the receptacle as an interface to a virtual generator.

Security

The security component of Grid Computing is of crucial concern. A major requirement for grid computing is security. At the base of any grid environment, there must be mechanisms to provide security, that perform following functions:

- Secure communication between elements of a computational Grid.
- Security across organizational boundaries, thus prohibiting a centrally-managed security system.
- Facilities like "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites.

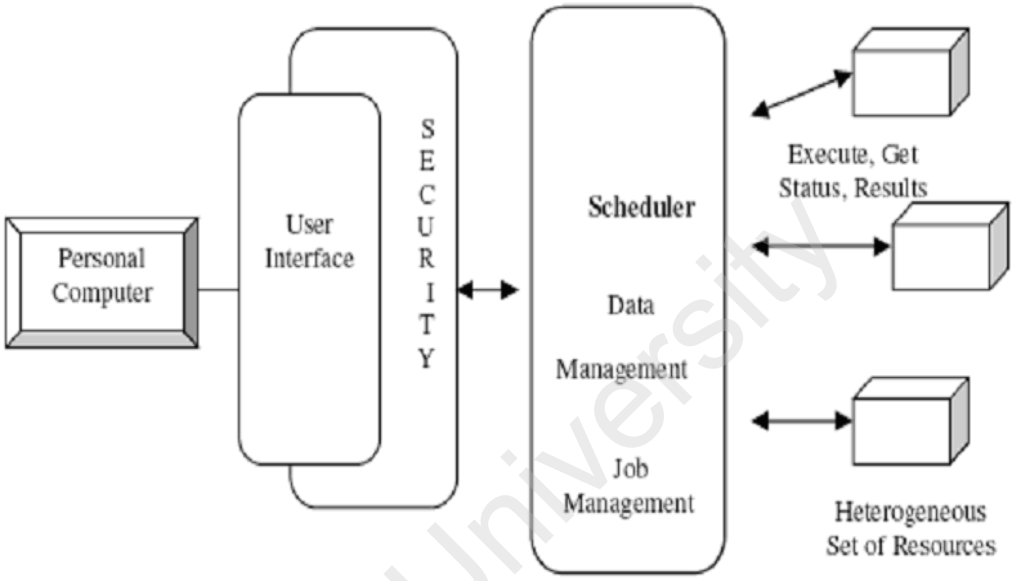


Figure 1.11: Grid Components

Workload Management

Applications that a user wants to run on a Grid must be aware of resources that are available. Application can communicate with the workload manager to discover the available resources and their status.

Data Management

Data management is required for ensuring data is available when the job is scheduled to run. A secure and reliable data management facility takes care of moving required data to the right place across various machines, encountering various protocols.

Scheduler

A Grid scheduler must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed, and information

about the systems is often limited or dated. This can be as simple as taking the next available resource, but often this task involves prioritizing job queues, managing the load, finding idle machines.

1.8.1 Grid Resource Management System

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [15]. Grid Resources can be computers, storage space, instruments, software applications, and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring, resource management, and so forth

To manage grid resources a **Grid Resource Management System** is required which deals with the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible.

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

1.8.2 Grid Scheduling

Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites.

Job: anything that needs a resource – from a bandwidth request, to an application, to a set of applications (for example, a parameter sweep).

Resource*:* anything that can be scheduled: a machine, disk space, a QoS network etc.

Grid Scheduling is a software framework with which the scheduler collects resource state information, selects appropriate resources, predicts the potential performance for

each candidate schedule, and determines the best schedule for the applications to be executed on a Grid System subject to QoS goals.

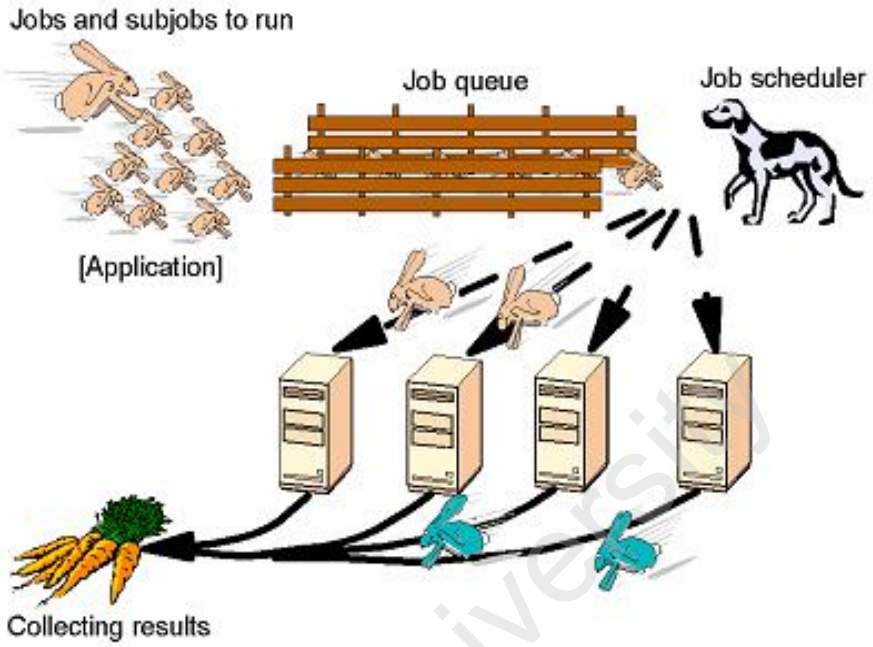


Figure 1.12:Application is one or more jobs that are scheduled to run on grid [11]

Grid Resource Management involves several different layers of schedulers. At the highest level are Grid-level schedulers that may have a more general view of the resources but are very "far away" from the resources where the application will eventually run. At the lowest level is a local resource management system that manages a specific resource or set of resources.

Differences between a Grid scheduler and a local resource scheduler:

Grid scheduler does not "own" the resources at a site (unlike the local resource scheduler) and therefore does not have control over them. The Grid scheduler must make best-effort decisions and then submit the job to the resources selected, generally as the user. Furthermore, often the Grid scheduler does not have control over the full set of jobs submitted to it, or even know about the jobs being sent to the resources it is considering use of, so decisions that tradeoff one job's access for another's may not be able to be made in the global sense. This lack of ownership and control is the source of many of the problems to be solved in this area.

Chapter 2

Review of Literature: Grid Scheduling

2.1 Grid Scheduler Architecture

The generic Grid Scheduler architecture provides a high-level view of Grid schedulers. Every component seems necessary for any comprehensive Grid Scheduling System [16]. Generic scheduler does the following:

- Interact with local resource managers
- Interact with external services that are not defined in the Grid Scheduling Architecture, like information, forecasting, submission, security or execution services
- Receive a scheduling problem
- Calculate a schedule, and return a scheduling decision

Real resources lay on the bottom of the architecture in the Figure each being Managed by a **Local Resource Manager (LRM)**. An **LRM** act as the interface between a Grid Scheduler and a local autonomous site. A Local Resource Manager (LRM) is mainly responsible for two jobs: local scheduling inside a resource domain, where not only jobs from exterior Grid users, but also jobs from the domain's local users are executed, and reporting resource information to GIS. Examples of such local schedulers include OpenPBS and Condor .An LRM also collects local resource information by tools such as Network Weather Service [28].

Information Service

Information about the status of available resources is very important for a Grid scheduler to make a proper schedule, especially when the heterogeneous and dynamic nature of the Grid is taken into account. The role of the Grid information service (GIS) is to provide such information to Grid schedulers. GIS is responsible for collecting and predicting the resource state information, such as CPU capacities, memory size, network bandwidth, software availabilities and load of a site in a particular period. GIS can answer queries for resource information or push information to subscribers. The Globus Monitoring and Discovery System (MDS) [33] is an example of GIS [28].

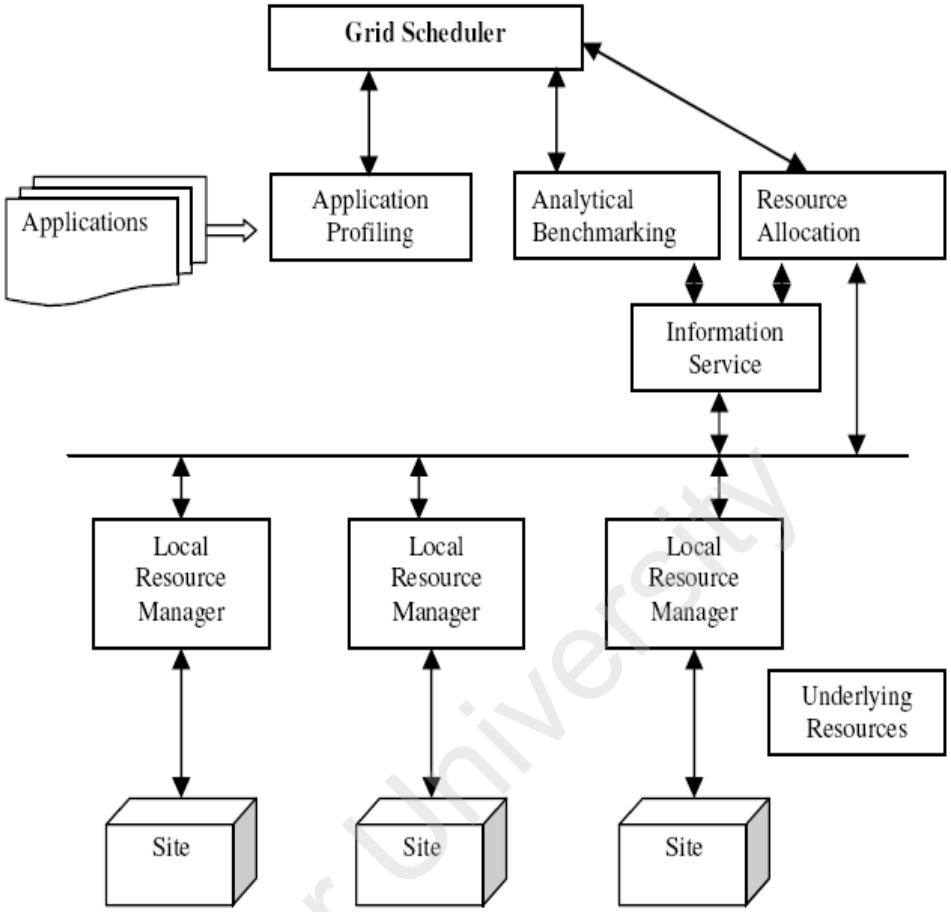


Figure 2.1: General Grid Scheduler Architecture [16]

Application Profiling and Analytical Benchmarking

Application profiling (AP) is used to extract properties of applications, while analogical benchmarking (AB) provides a measure of how well a resource can perform a given type of job. On the basis of knowledge from AP and AB, and following a certain *performance model*, cost estimation computes the cost of candidate schedules, from which the scheduler chooses those that can optimize the objective functions [28].

Grid Scheduler

This is the core component in the architecture. The GS needs to do two jobs: one is *resource selection* and the other one is *mapping*. Resource selection is the process of selecting feasible and available resources for a given application to be scheduled.

Mapping is the process of placing the jobs and communications of the application onto the resources and networks. Each mapping of jobs to feasible resources produces a candidate schedule. Each candidate schedule is then estimated for its performance potential based on the performance model.

Resource Allocation

This component implements a finally determined schedule through allocating the resources to the corresponding jobs. Resource allocation may involve data staging and binary code transferring before the job starts to execute on the computational resource.

2.2 Grid Scheduling Approaches

2.2.1 Application Centric

Scheduling systems in the application centric category try to favor performance of individual applications. Typical performance goals sought by application-centric Scheduling systems include minimizing execution time, maximizing the speed up etc. e.g. an application-centric Scheduling system will exploit a mapping algorithm, which allocates the application to resources that are likely to produce the best performance without considering the rest of pending applications.

2.2.2 System-Centric

A system centric Scheduling system concerns the overall performance of the whole set of applications and the whole Grid system. The performance goals desired by a system centric policy typically include resource utilization, system throughput and average application response time. Ordering on the pending applications is usually performed in order to achieve a higher system centric performance.

2.2.3 Economy Based

An economy based Scheduling system introduces the idea of market economy. Under this scheme, Scheduling decisions are made based on the economy model. The economy model defines each application having the desired QoS, such as execution time and deadline and the cost that the application will pay for the desired QoS; each

resource is specified by its cost and the capacity. For each application, it wants to get as higher QoS as possible within the budget constraint. For each resource, it wants to get profit as much as possible by keeping itself busy. Nimrod-G is a Scheduling system, which exploits idea of economy mechanism.

2.3 Grid Scheduling Procedure

Grid Scheduling involves 3 phases:

Phase 1: Resource Discovery

Resource discovery is the process that takes as input a user request and returns a list of resources or services that can possibly fulfil the given request.
Resource discovery is the first phase of resource scheduling. It involves the user selecting a set of resources. At the beginning of this phase, the potential set of resources is the empty set, and at the end of this phase, the potential set of resources is some set that has passed a minimal feasibility requirement [15].

Steps in Resource Discovery

1. Authorization filtering,
2. Job requirement knowledge, and
3. Filtering to meet the minimal job requirements.

Step 1: Authorization Filtering

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

Step 2: Application requirement definition

In order to proceed in Resource Discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources the set of possible job requirements can be very broad, and vary significantly between jobs. This may include any information about the job that should be specified to make sure that the job could be matched to a set of resources.

Step 3: Minimal requirement filtering

Given a set of resources to which a user has access and the minimal set of requirements the job has, the third step in the Resource Discovery phase is to filter out the resources that do not meet the minimal job requirements. The user eliminates the ones that do not meet the job requirements. It could also be combined with the gathering of more detailed information about each resource. However, when being done by hand, if a user can eliminate an inappropriate resource it is done at this stage to simplify the information gathering in the next phase.

Phase 2: System Selection

Step 4: Dynamic Information Gathering

In order to make the best possible job/resource match, detailed dynamic information about the resources is needed. The dynamic information-gathering step has two components: what information is available and how the user can get access to it. The information available will vary from site to site, and users currently have two man sources–a GIS and the local resource scheduler.

Step 5: System Selection

With the detailed information gathered in Step 4, the next step is to decide which resource (or set of resources) to use. For example Condor matchmaking, multi-criteria and meta-heuristics.

Phase 3: Job Execution

Step 6: Advance Reservation (Optional)

In order to make the best use of a given system, part or all of the resources may have to be reserved in advance. Depending on the resource, an advance reservation can be easy or hard to do and may be done with mechanical means or human means. Moreover, the reservations may or may not expire with or without cost. One issue in having advance reservations become more common is the need for the lower-level resource to support the fundamental services on the native resources.

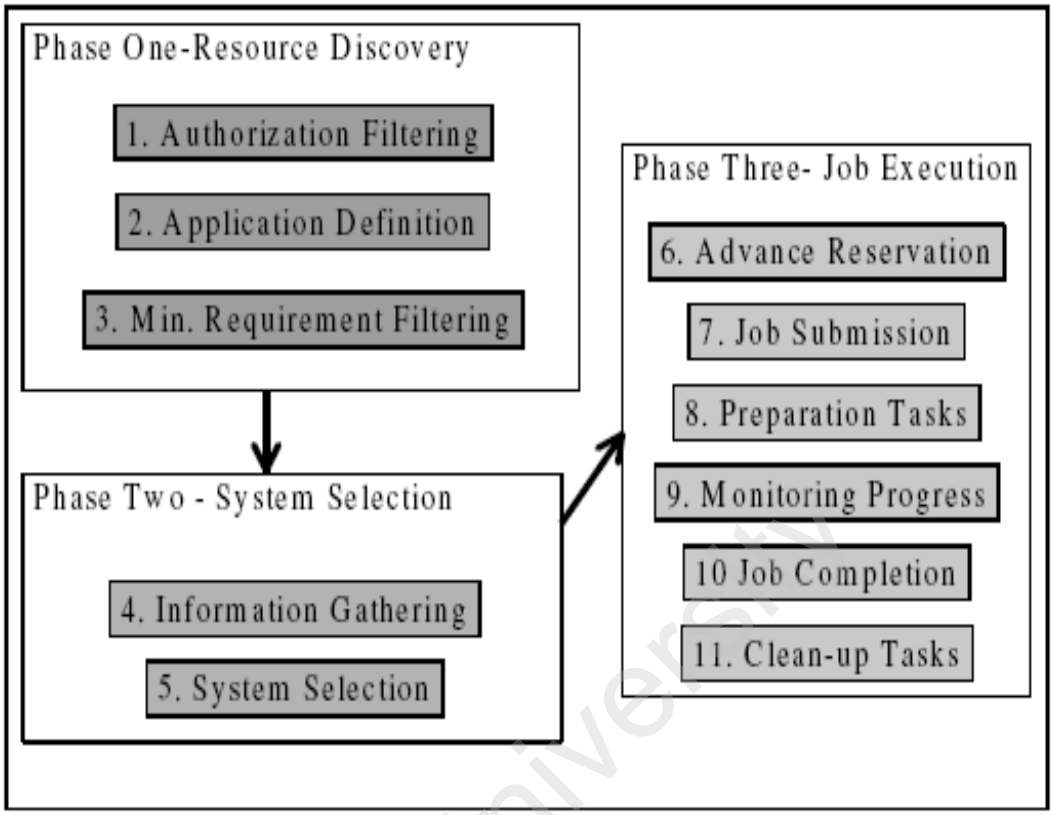


Figure 2.2: Resource Scheduling Phases [15]

Step 7: Job Submission

Once resources are chosen, the application can be submitted to the resources. Job submission may be as easy as running a single command or as complicated as running a series of scripts and may or may not include setup or staging (see Step 8). In a Grid system, the simple act of submitting a job can be made very complicated by the lack of any standards for job submission.

Step 8: Preparation Tasks

The preparation stage may involve setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application. One of the first attempts at writing a scheduler to run over multiple machines at NASA was considered unsuccessful because it did not address the need to stage files automatically. Most often, a user will run ftp or a large file transfer protocol such as GridFTP to ensure that the data files needed are in place. In a Grid setting, authorization issues, such as

having different user names at different sites or storage locations, as well as scalability issues, can complicate this process.

Step 9: Monitoring Progress

Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing. Such monitoring is done by repetitively querying the resource for status information. If a job is not making sufficient progress, it may be rescheduled. Such rescheduling is significantly harder on a Grid system than on a single parallel machine because of the lack of control. It may be possible to develop additional primitives for interactions between local systems and Grid schedulers to make this behavior more straightforward.

Step 10: Job Completion

When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines will include an e-mail notification parameter. For fault-tolerant reasons, however, such notification can prove surprisingly difficult. Moreover, with so many interacting systems one can easily envision situations in which a completion state cannot be reached. And of course, end-to-end performance monitoring to ensure job completion is a very open research question.

Step 11: Cleanup Tasks

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, remove temporary settings, and so forth. Any of the current systems that do staging (Step 8) also handle cleanup. Users generally do this by hand after a job is run, or by including clean-up information in their job submission scripts.

2.4 Challenges in Grid Scheduling

2.4.1 Resource Heterogeneity

Networks used to interconnect the resources may differ significantly in terms of their bandwidth and communicational protocols. Heterogeneity in Computational resources

is due to different hardware, such as instruction set, computer architecture, number of processors, physical memory size, CPU Speed and so on and also different software such as different operating systems, file systems, cluster management software and so on. The heterogeneity results in differing capability of processing jobs. Resources with different capacity cannot be considered uniformly [16].

2.4.2 Site Autonomy

In a Grid resources are usually autonomous and the Grid scheduler does not have full control of the resources. It cannot violate local policies of resources, which makes it hard for the Grid scheduler to estimate the exact cost of executing a task on different sites. The autonomy also results in the diversity in local resource management and access control policies, such as, the priority settings for different applications and the resource reservation methods [28].

A single overall performance goal is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performance goal.

2.4.3 Local Priority

Each site within the Grid has its own Scheduling policy. For example, it can be expected that local jobs will be assigned higher priorities such that local jobs will be better served on the local resources.

2.4.4 Resource Non-Dedication

Due to the non-dedication of resources, a resource may join multiple Grids simultaneously. Also the workloads from both local users and other Grids share the resource concurrently.

2.4.5 Application Diversity

This problem arises because the Grid Applications are from a wide range of users, each having its own special requirements. For example, some applications may require sequential execution, some applications may consist of a set of independent jobs, and others may consist of a set of dependent jobs. In this context, building a general-purpose Scheduling system seems extremely difficult [16].

2.4.6 Dynamic Behavior

Dynamics exists in both the networks and computational resources. Grid schedulers work in a dynamic environment where the performance of available resources is constantly changing. The change comes from site autonomy and the competition by applications for resources. For example many parties may share a network and hence it cannot provide guaranteed bandwidth. Both the availability and capability of computational resources will exhibit dynamic behavior. Resources may join or leave Grid dynamically or there may be network failures. An adequate scheduler should adapt to such dynamic behavior. After a new resource joins the Grid, the scheduler should be able to detect it automatically and use these resources in the later Scheduling decision-making process.

2.5 Parameters to be Considered While Scheduling

To arrive at a scheduling decision, the scheduling system needs to take various parameters into consideration including the following: [18]

- Resource Architecture and Configuration
- Resource Capability (clock speed, memory size)
- Resource State (such as CPU load, memory available, disk storage free)
- Resource Requirements of an Application
- Access Speed (such as disk access speed)
- Free or Available Nodes
- Priority (that the user has)
- Queue Type and Length
- Network Bandwidth, Load, and Latency (if jobs need to communicate)
- Reliability of Resource and Connection
- User Preference
- User Capacity/Willingness to Pay for Resource Usage
- Resource Cost Variation in terms of Time-scale
- Historical Information, including Job Consumption Rate
- Deadline (period by which an application execution needs to be completed)

2.6 Grid Scheduling Algorithms

The Grid Scheduling Algorithms can be classified as:

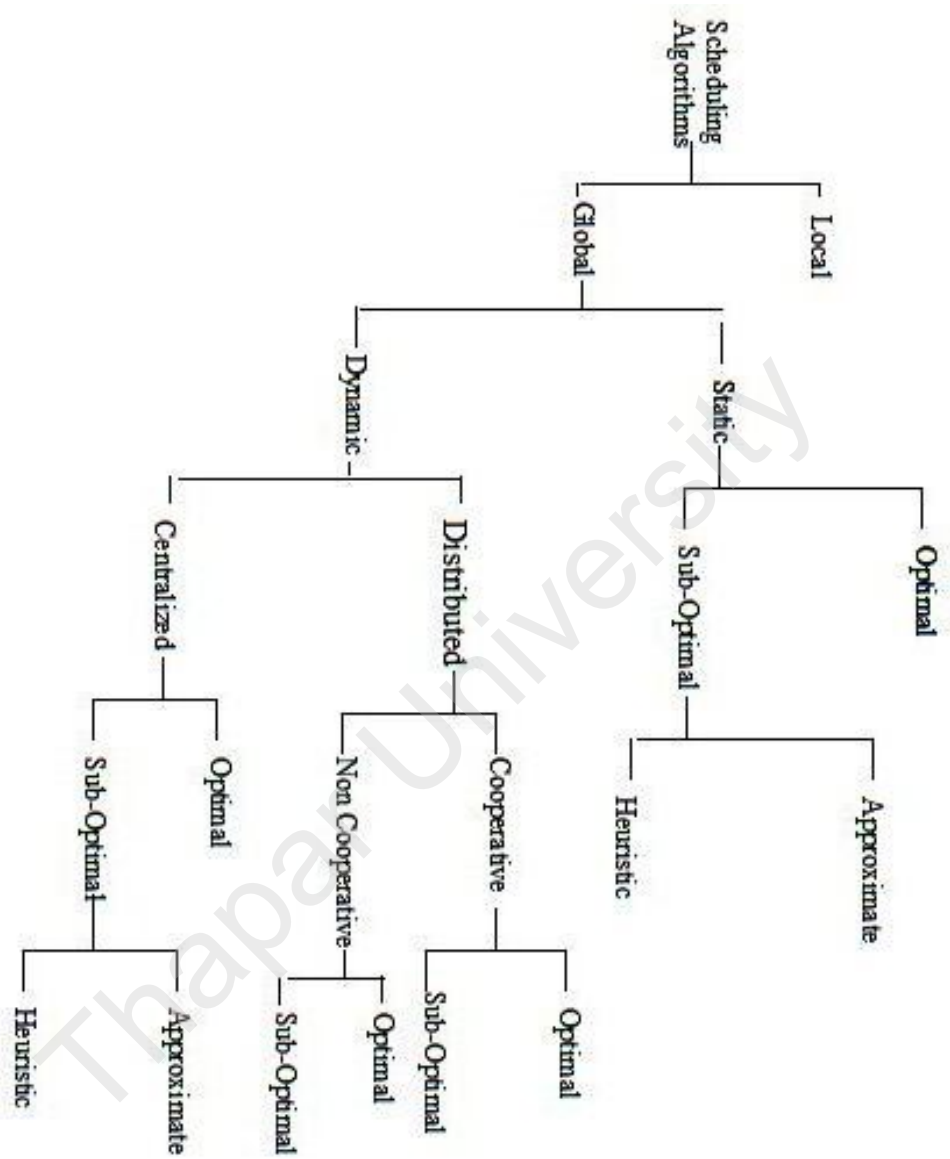


Figure 2.3: A Hierarchical taxonomy for scheduling algorithms

2.6.1 Local vs. Global

The local Scheduling discipline determines how the processes resident on a single CPU are allocated and executed; a global Scheduling policy uses information about the system to allocate processes to multiple processors to optimize a system wide performance objective. Grid Scheduling falls into the Global Scheduling [28].

2.6.2 Static vs. Dynamic

The next level in the hierarchy (under the Global Scheduling) is a choice between static and dynamic Scheduling. This choice indicates the time at which the Scheduling decisions are made. In case of static Scheduling, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. By contrast, in the case of dynamic Scheduling, the basic idea is to perform task allocation on the fly as the application executes.

2.6.3 Optimal vs. Suboptimal

In the case that all information regarding the state of resources and the jobs is known, an optimal assignment could be made based on some criterion function, such as minimum makespan and maximum resource utilization. But due to difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the optimality of an algorithm, current research tries to find suboptimal solutions, which can be further divided into the following two general categories.

2.6.4 Approximate vs. Heuristic

The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. The factors, which govern their decision, are:

- Availability of a function to evaluate a solution.
- The time required evaluating a solution.
- The ability to judge the value of an optimal solution according to some metric.
- Availability of a mechanism for intelligently pruning the solution space.

Heuristic represents the class of algorithms, which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It represents the solutions to the Scheduling problem, which cannot give optimal answers and require the most reasonable amount of cost and other system resources to perform their function. The evaluation of this kind of solution is usually based on experiments in the real world or on simulation. Heuristic algorithms are more adaptive to the Grid scenarios.

2.6.5 Cooperative vs. Non-cooperative

If a distributed Scheduling algorithm is adopted, the next issue that should be considered is whether the nodes involved in job Scheduling are working cooperatively or non-cooperatively. In the non-cooperative case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of system e.g. application-level schedulers. In the cooperative case, each Grid scheduler has the responsibility to carry out its own portion of the Scheduling task, but all schedulers are working toward a common system-wide goal.

2.6.6 Distributed vs. Centralized

In dynamic Scheduling scenarios, the responsibility for making global Scheduling decisions may lie with one centralized scheduler, or be shared by multiple distributed schedulers. The centralized strategy has the advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck.

Chapter 3
Proposed Grid Scheduling Algorithm

Grid Computing is a service aggregation of both the information and the computational resources in heterogeneous environment. To manage these resources efficiently a Grid Resource Management System is required. It deals with the process of identifying requirements, matching resources to applications, allocating, monitoring and scheduling those resources. Due to dynamicity and heterogeneity of resources as well as grid applications, scheduling becomes challenging. In general grid scheduling is a NP hard problem [19]. A problem is NP-hard if solving it in polynomial time would make it possible to solve all problems in class NP in polynomial time.

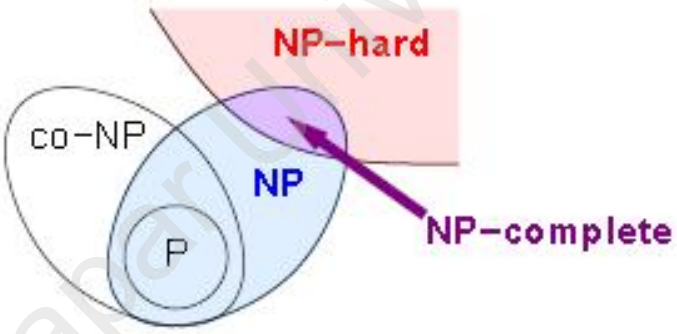


Figure 3.1: Relationship between NP-Hard, NP-Complete & NP Problems [19]

This chapter presents an efficient scheduling algorithm, which aims to optimize the makespan. The proposed algorithm is implemented using DAG approach and comparative results show that it optimizes makespan.

The problem of scheduling a set of tasks to a set of processors can be divided into two categories:

- *Job scheduling*,
- *Job mapping and scheduling*.

In the former category, independent jobs are to be scheduled among the processors of a distributed computing system to optimize overall system performance. In contrast, the mapping and scheduling problem requires the allocation of multiple interacting

tasks of a single parallel program in order to minimize the completion time on the parallel computer system. The mapping and scheduling problem can be addressed in both static as well as dynamic contexts. To generate the schedule, our technique is based on the traditional list scheduling approach in which we construct a list and schedule the nodes on the list one by one to the processors.

List scheduling is a class of scheduling in which tasks are assigned priorities and placed in a list, which is ordered in decreasing magnitude of priority. The selection of tasks to be processed is done on the basis of priority with higher-priority tasks being assigned to resources first [19]. Many list heurists have been invented, and some new proposals can be found in [20], [21] and [22] as well as the comparison of their algorithms with older ones

In this chapter we present a Grid Scheduling Algorithm, which optimizes makespan. *Makespan is defined as the maximum time taken for the completion of all the tasks in a given application*. For this purpose our implementation model consists of a Directed Acyclic Graph (DAG).

Assumptions:

Proposed scheduling algorithm makes simplifying assumptions about the architecture of parallel programs and target computing systems such as:

- Uniform computational cost of tasks
- Negligible communication cost between tasks
- Heterogeneous Grid Computing Environment, which has a finite number of computing resources in a fully connected topology.
- An application consists of finite number of tasks

The algorithm can handle multiple independent jobs with precedence constraints among job tasks and arbitrary job lengths. The algorithm is based on list scheduling concept. We assume that any machine can execute the task and communicate with other machine at the same time. Once a machine has started task execution it continues without interruption and after completing the execution it immediately sends the output data to all children tasks in parallel.

3.1 Problem Formulation

The implementation model consists of a Directed Acyclic Graph (DAG).

Directed Acyclic Graph (DAG): Directed acyclic graphs are directed graphs with no directed cycles that is, for any vertex *v*, there is no nonempty directed path that starts and ends on *v*.

- A finite DAG has at least one source and at least one sink.
- The depth of a vertex in a finite DAG is the length of the longest path from a source to that vertex, while its height is the length of the longest path from that vertex to a sink.

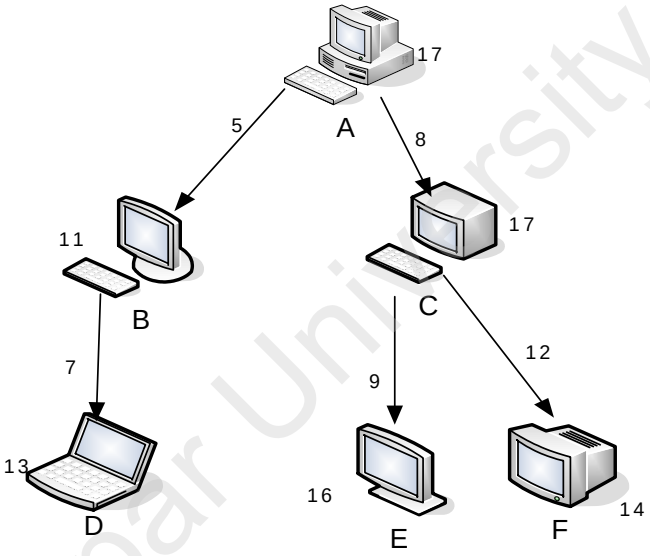


Figure 3.2 (a): Scheduling in Grids

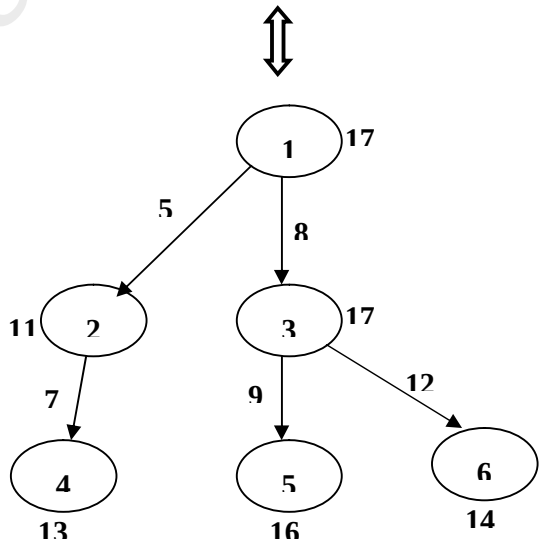


Figure 3.2 (b): Directed Acyclic Graph (DAG)

Scheduling in grids is similar to DAG. The nodes in the figure 3.2 (a) represent a scheduling scenario in grid environment. In figure 3.2 (a) A, B, C, D, E, F represent the resources on which tasks will be executed and the directed edges represent the execution dependencies as well as the amount of communication. This can be mapped to Directed Acyclic Graph (DAG) as shown in figure 3.2 (b). The numerical value besides each node denotes the computational cost of the task and the numerical value on each directed edge indicate communication cost between the pair of nodes connected. Scheduling is done in such a manner that the precedence constraints among the tasks are preserved.

Given *n* jobs to be scheduled on a set of *m* machines. Jobs have certain size that can be specified as Million Instruction (MI) and machines have processing speed, which can be specified in Million Instructions Per Second (MIPS). We consider the problem formulation in which an instance of the problem consists of the following.

- Given a *set of jobs* and *a set of machines*
- *Number of jobs* that must be scheduled with different processing length
- *Number of heterogeneous machines* to participate in the schedule.
- *Workload* of each task (MI).
- *Computing capacity* of each machine (in MIPS).
- *Total execution time* of the job on a given Grid Resource
- *Expected Time to Compute*, *ETC*, where ETC_{ij} indicates the expected execution time of task *i* in machine *j*.
- Jobs have *precedence constraints*.
- There may be *setup times* for different types of jobs.

Precedence constraints between jobs are given in the form of a partial order. If j < k, processing of job k cannot start until job j's execution is completed. The objective is to find a non-preemptive schedule to minimize the makespan of the schedule.

3.2 Proposed Makespan Optimization based Grid Scheduling Algorithm

It is assumed that the job consists of gridlets or tasks. The grid scheduler assigns these gridlets or tasks to resource. Also it is assumed that each computational resource can run one application at a time, and must run that application to completion.

Let *T* be a set of *n* gridlets and *m* is the number of computational resources in a grid. We define a schedule of *T* as follows: A schedule *S* of *T* onto a grid with *m resources* is a finite set of triples <*v, p, t*> where *v* is the schedule, *t* is the starting time and *p* is the resource. A triple <*v, p, t*> is called the *task instance.*

To generate the schedule, our technique is based on the traditional list scheduling approach in which we construct a list and schedule the nodes on the list one by one to the processors. The list is constructed by ordering the nodes according to the node priorities. The list is static therefore the order of nodes on the list will not change during the resource allocation process.

Given a DAG of v nodes $\{n_1, n_2,.., n_v\}$ and e directed edges, each of which is denoted by (n_i, n_j). We shall traverse DAG in breadth first order.

Let C_j denote the completion time of job j. The objective is to find a schedule that minimizes the quantity $C_{max} = \max(C_j)$, called the makespan of the schedule.

Makespan=∑ Actual CPU time taken by gridlets on grid resources.

We restrict ourselves to non-preemptive schedules where a job once started has to run to completion on the same machine.

Scheduler has information about all resources such as processing speed (in MIPS), processing cost per second, baud rate (communication rate), and resource load during peak hours and off-peak hours. Each grid resource can have multiple machines and each machine can have multiple processing elements. MIPS of a resource is sum of all its processing elements' MIPS. Expected completion time (ECT) is the function of resource MIPS, resource baud rate, gridlet length, gridlet input and output size in MI.

ECT=f (resource characteristics, gridlet characteristics)

After gathering the details of user jobs and the available resources, the system randomly creates jobs according to the given data. Eventually, the scheduler will submit the job according to proposed algorithm to the corresponding resources. The tests are conducted using four resources of different MIPS. The MIPS of each resource is computed as follows:

*Resource MIPS = Total_PE * PE_MIPS,*

Where *Total_PE* = Total number of Processing Elements of the resource, and

PE_MIPS = MIPS of Processing Element

Each resource has its own predefined cost rate for charges imposed on a Grid user for executing the user jobs on that resource. In the simulation, the total processing time is calculated in milliseconds. In real world, the overhead time for each job depends on the current network load and speed. The total processing cost is computed based on the actual CPU time taken by the grid resource for computing the gridlets and the cost rate specified by that grid resource, as summarized below:

Process_Cost = *T* * *C*, where

T = Total CPU Time for Gridlet execution, and

C = Cost per second for executing gridlet on the grid resource.

The proposed algorithm, comprises of two parts as explained below:

- *Task Ordering Procedure*, to get the scheduling list.
- *Resource Allocation Procedure*, which allocates resources to the jobs contained in scheduling list, generated by task ordering procedure.

3.2.1 Task Ordering Procedure:

Begin

Step 1: The list is initialized to be an empty list. Set *position* to 1.The first node in the list is the node, which does not have any predecessor. Set *position* to 2. Let n_x be the next node.

Repeat

Step 2: if n_x has all its parent nodes in the list then

Step 3: Put n_x at *position* in list and increment Position.

Step 4: Else

Step 5: Let n_y be the parent node of n_x which is not in the sequence and has the smaller level. Ties are broken by choosing node with larger gridlet length.

Repeat

Step 5: If n_y has all its parent nodes in the sequence then

Step 6: Put n_y at *position* in sequence and increment *position*.

Step 7: Else

Step 8: Include all the ancestor nodes of n_y in sequence so that the nodes with a smaller value of level are considered first.

Step 9: End if

Until all the parent nodes of n_x are in the list.

Step 10: Put n_x at *position* in the list.

Step11: End if

Step 12:Make n_x to be the next node

Until all the nodes are in the list.

Step13: Append all the nodes to the sequence in increasing order of level.

End

3.2.2 Resource Allocation Procedure:

Begin

Step 1: Scheduler collects resources and their characteristics like processing speed , processing cost per second, baud rate, and resource load during peak hours and off-peak hours.

Step 2: Discard resources whose processing speed is less than 1/m times the speed of the fastest machine where m is the number of machines.

Repeat

Step 3: For all gridlets i to be scheduled

Step 4: Repeat for all resources j

Step 5: ComputeECT_{ij}= ECT(gridlet i, resource j)

End loop

Step 6: Select best match such that ECT_{ij} is minimum

Step 7: For all gridlets i to be scheduled on resource j

Step 8: Calculate actual execution time for gridlet i on resource j

Step 9: Compute Makespan as sum of actual execution time for all submission

Step 10: Compute Processing cost as actual CPU time * cost per second

Until there are gridlets to be scheduled

End

3.3 Complexity Analysis of the Makespan Optimization Algorithm

The complexity is compressed in terms of:

- Number of gridlets, *n*
- Number of resources, *m.*

Complexity Analysis of the Task Ordering Procedure:

- Complexity of Step 1 in which the list is initialized with the node, which does not have predecessors *O (n).*
- The complexity of list creation phase is *{O (log n) * O (n + log n)}* which eventually becomes *O (n log n).*

Total complexity of Task Ordering Procedure= *{O (n)+ O (n log n)} ={O (n log n)}.*

Complexity Analysis of the Resource Allocation Ordering Procedure:

- complexity for step 1 = *O(n).*
- complexity for step 2 = *O(n) + O(n).*
- complexity for step 3,4,5 = *O(nm).*
- complexity for step 6 = *O(nm).*

Total complexity of Resource Allocation Procedure = *{O (n) + O (n) + O (n) + O(nm) + O (nm)} = {O (3n+2nm) } = O(nm)*

Total Complexity of the Makespan Optimization Algorithm:

{ O (n log n) + O (nm) }.

Chapter 4
Implementation of Proposed Makespan Optimization Algorithm

The management and scheduling of resources in Grid environment is complex and therefore, demands sophisticated tools for analysing the algorithms before applying them to the real systems. The proposed makespan optimization algorithm is implemented using GridSim Toolkit 4.1.

GridSim is a software platform that enables users to model and simulate the characteristics of Grid resources and networks with different configurations. The motivation for using simulation instead of directly using the Grid test-bed, comes from the following factors [23]:

- Setting up a Grid test-bed is expensive, resource intensive, and time consuming.
- The real test-bed does not provide a repeatable and controllable environment for experimentation and evaluation of scheduling strategies.
- Simulation works well, without making the analysis mechanism unnecessary complex, by avoiding the overhead of co-ordination of real resources.
- Simulation is also effective in working with very large hypothetical problems.

4.1 GridSim Architecture

GridSim is a open source JAVA based toolkit. It has a layered and modular architecture as shown in figure 4.1

The **first layer** is concerned with the scalable Java interface and the runtime machinery, called JVM (Java Virtual Machine), whose implementation is available for single and multiprocessor systems including clusters [23].

The **second layer** is concerned with a basic discrete-event infrastructure built using the interfaces provided by the first layer.

The **third layer** is concerned with modeling and simulation of core Grid entities such as resources, information services, and so on; application model, uniform access interface, and primitives application modeling and framework for creating higher level entities. The GridSim toolkit focuses on this layer that simulates system entities.

The **fourth layer** is concerned with the simulation of resource aggregators called Grid resource brokers or schedulers.

The **final layer** is focused on application and resource modeling with different scenarios using the services provided by the two lower-level layers for evaluating scheduling and resource management policies, heuristics, and algorithms.

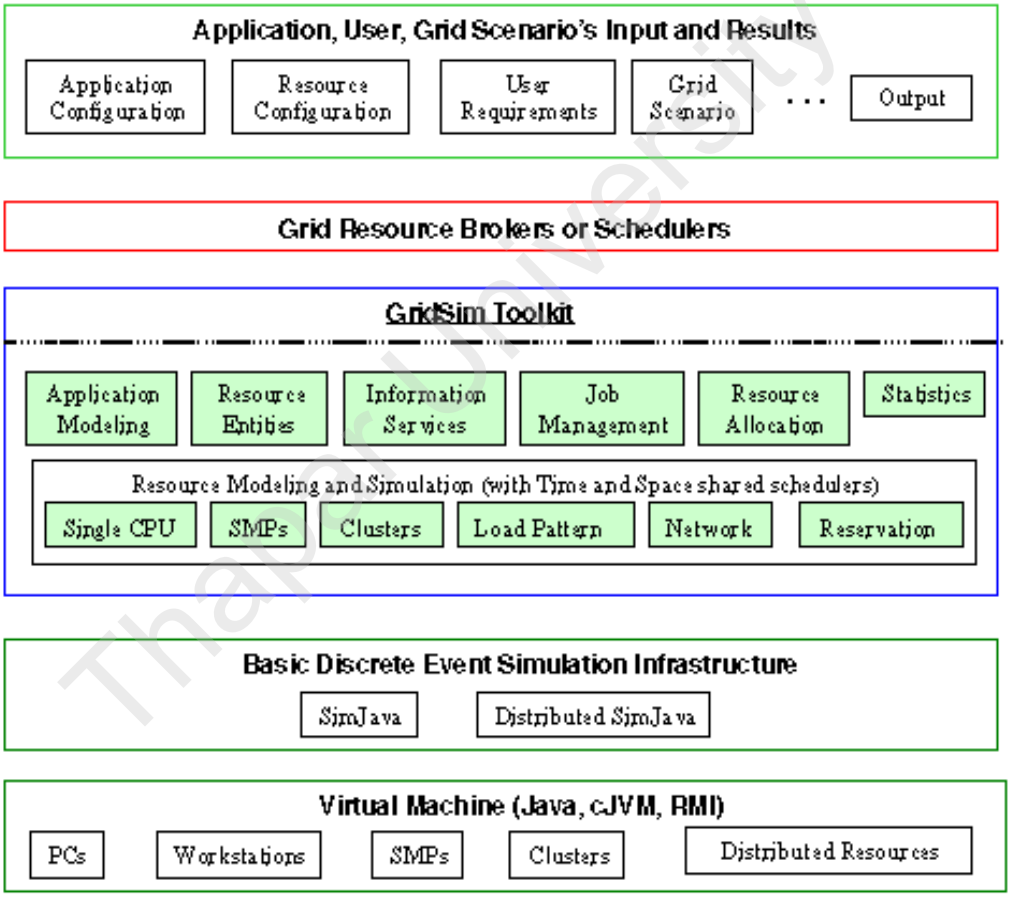


Figure 4.1. Modular Architecture for GridSim Platform and Components [23]

4.1.1. GridSim entities

GridSim supports entities for simulation of single processor and multiprocessor, heterogeneous resources that can be configured as time- or space-shared systems.

User: Each instance of the User entity represents a Grid user. Each user may differ from the rest of users with respect to the following characteristics:

- Types of job created, e.g. job execution time, number of parametric replications, etc;
- Scheduling optimization strategy, e.g. minimization of cost, time, or both activity rate, e.g. how often it creates new job
- Time zone
- Absolute deadline

Broker: Each user is connected to an instance of the Broker entity. Every job of a user is first submitted to its broker and the broker then schedules the parametric tasks according to the user's scheduling policy. Before scheduling the tasks, the broker dynamically gets a list of available resources from the global directory entity. Every broker tries to optimize the policy of its user and therefore, brokers are expected to face extreme competition while gaining access to resources. The scheduling algorithms used by the brokers must be highly adaptable to the market's supply and demand situation [23].

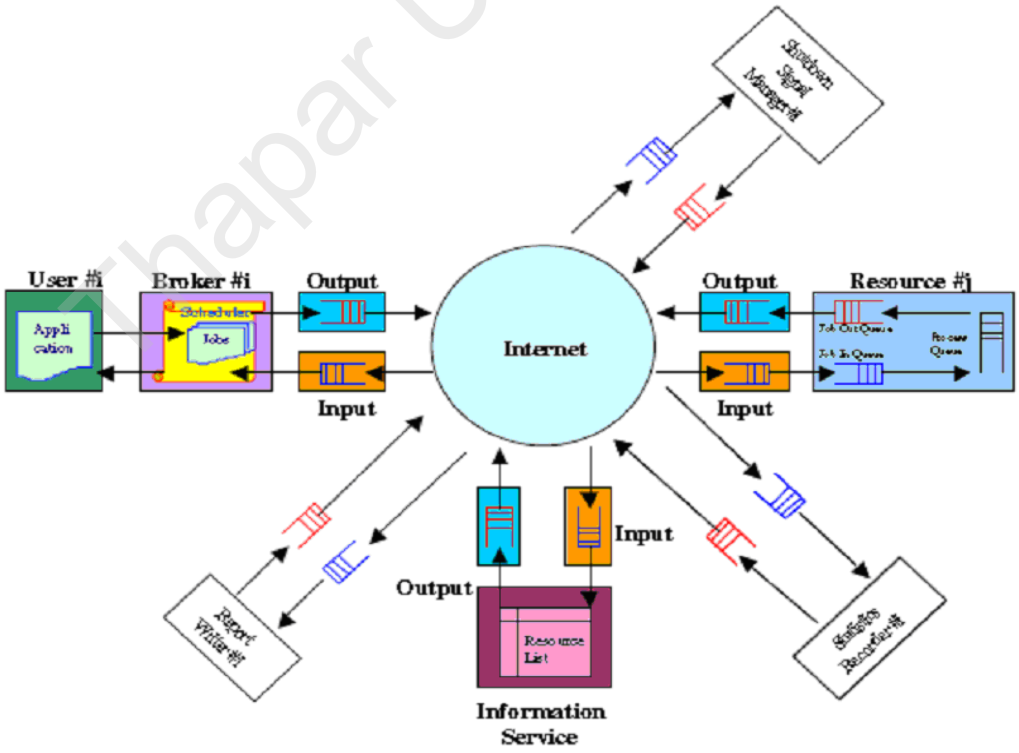


Figure 4.2. A flow diagram in GridSim based simulation [23]

Resource: Each resource may differ from the rest of the resources with respect to the following characteristics:

- number of processors
- cost of processing
- speed of processing
- internal process scheduling policy, e.g. time-shared or space-shared
- local load factor
- time zone

The resource speed and the job execution time can be defined in terms MIPS (Million Instructions Per Second) and SPEC (Standard Performance Evaluation Corporation) benchmark).

Grid Information Service: It provides resource registration services and keeps track of a list of resources available in the Grid.

Input and Output: The flow of information among the GridSim entities happens via their Input and Output entities. Every networked GridSim entity has I/O channels or ports, which are used for establishing a link between the entity and its own Input and Output entities.

4.2 Features of GridSim Toolkit

Salient features of the GridSim toolkit include the following.

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space- or time-shared mode.
- Resource capability can be defined (in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark).
- Resources can be located in any time zone.
- Weekends and holidays can be mapped depending on resource's local time to model non-Grid (local) workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.

- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.

Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

4.3 GridSim Toolkit 4.1 API Specification

Packages	
eduni.simjava	Classes that form the main simulation structure of SimJava.
eduni.simjava.distributions	Classes concerning random number sampling and random variate generation.
gridsim	Classes that form the main simulation structure of GridSim.
gridsim.auction	Classes that form the framework of auction model in GridSim.
gridsim.datagrid	Classes that form the framework of Data Grid model in GridSim.
gridsim.datagrid.filter	Classes that form the selection of File attributes in the Replica Catalogue.
gridsim.datagrid.index	Classes that form the structure of multiple regional Data GIS and Replica Catalogue entities.
gridsim.datagrid.storage	Classes that form the structure of Storage elements of

	a Data Grid resource.
gridsim.filter	Classes that form the selection of incoming events of an entity.
gridsim.index	Classes that form the structure of multiple regional GIS entities.
gridsim.net	Classes that form the network extension of GridSim.
gridsim.util	Classes that perform other important functionalities of GridSim

Table 4.1: GridSim Toolkit 4.1 API Specification [23]

4.4 Required Installations

4.4.1 Installation of Netbeans IDE 6.0.1

Netbeans IDE is freely available and can be downloaded from the following website:
http://www.**netbeans**.org
The IDE can be installed easily following the on screen messages. The next step is to configure the Netbeans IDE for the implementation of the algorithm.

4.4.2 Installation of GridSim Toolkit

In the current implementation of the Makespan Optimization Algorithm we have used GridSim Toolkit version 4.1. GridSim Toolkit 4.1 downloaded from the following website:
http://www.gridbus.org/gridsim/

Steps to be followed on Windows System for GridSim 4.1 installation are:

GridSim has been tested and ran on Sun's Java version 1.4.2 or newer. Older versions of Java are not compatible. If you have non-Sun Java version, such as J++, they may not be compatible. There is no need to compile GridSim source code. The JAR file is provided to compile and to run GridSim applications.

Description of the following jar files:

- gridsim.jar - contains both GridSim and SimJava v2.0 class files
- simjava2.jar - contains SimJava v2.0 class files only

Running GridSim of this version requires a lot of memory since there are many objects to be created. Therefore, it is recommended to have at least 512MB RAM or increase JVM heap size when running Java for large simulation experiments.

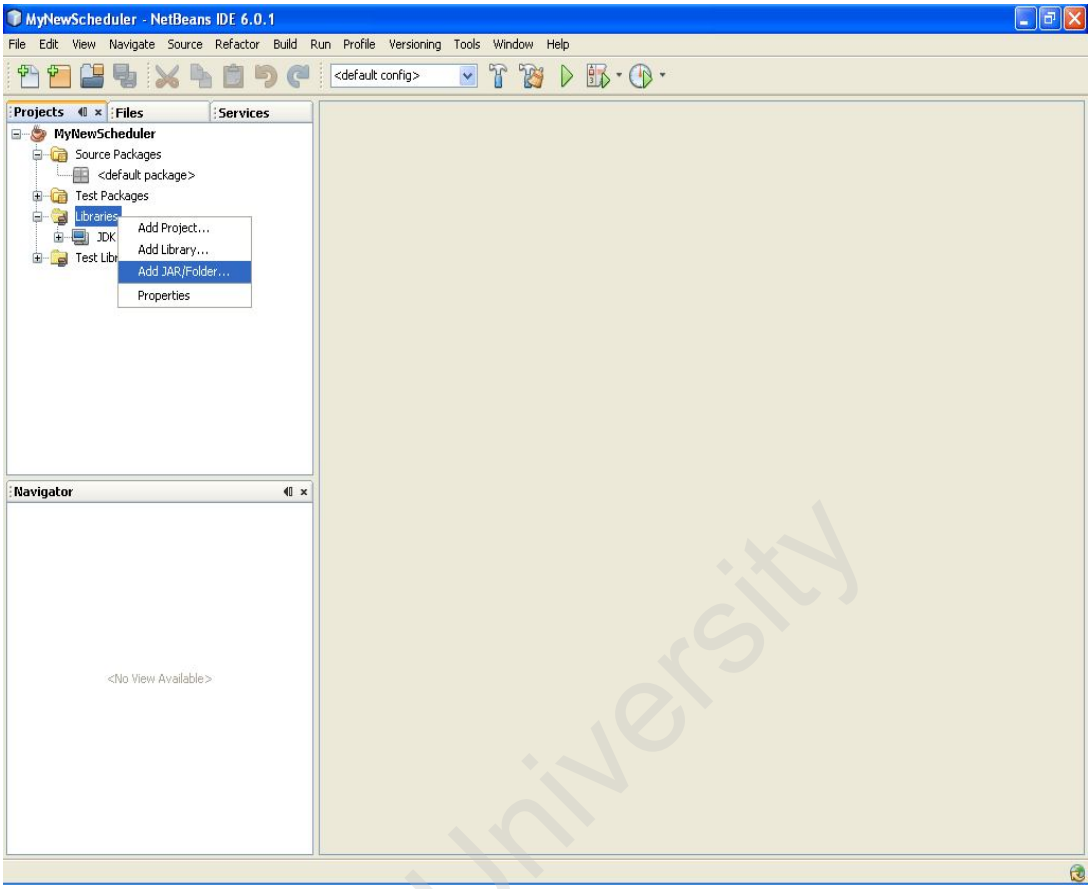
1. Unzip the downloaded archive to the location intended to be used for installation
2. Update the System Path variable to include the path to the gridsim.jar and simjava2.jar files, e.g. If the GridSim has been unzipped to the D:\gridsim-4.1 location then we add to the System Path variable
3. D:\gridsimtoolkit-4.1\jars\gridsim.jar and
4. D:\gridsimtoolkit-4.1\jars\simjava2.jar
5. The GridSim API is ready to be used in JAVA applications on the Windows system.
6. If you want to remove GridSim, then remove the whole $GRIDSIM directory.

The steps for installation on UNIX based Operating Systems are same with the exception of .tar file for GridSim Archive is to be downloaded and installed.

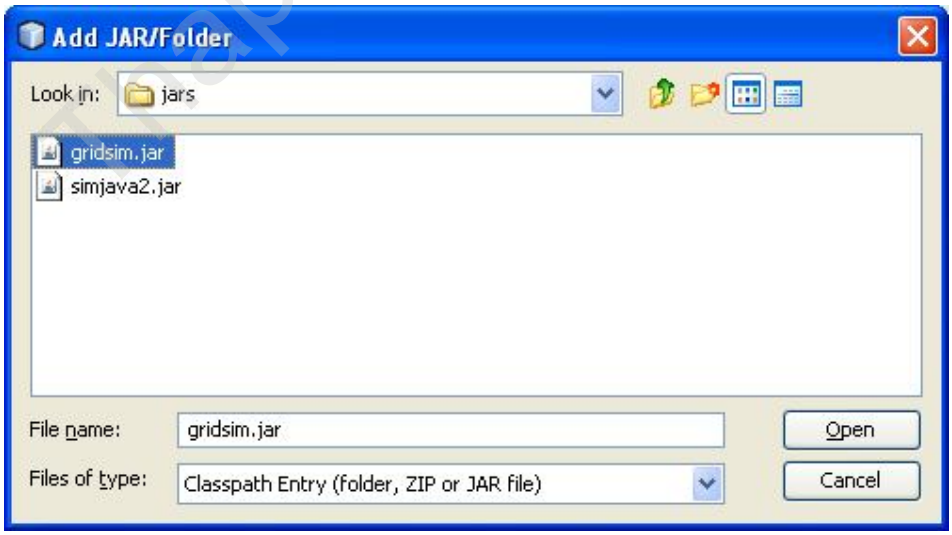
4.4.3 Importing GridSim package in Netbeans IDE.

The Proposed Makespan Optimization Algorithm has to be simulated on top of GridSim [23] API as a JAVA application built in Netbeans IDE. The steps to be followed are as:

1. Create a new Java Project in Netbeans
2. Create a new Package under this project.
3. To set the properties of the project go to the libraries and add jar/folder. Gridsim.jar and simjava2.jar are to be browsed and added accordingly as shown below.



Screenshot 4.1:Addition of external project dependencies



Screenshot 4.2: External project dependencies

4.5 Implementation of Proposed Makespan Optimization Algorithm in GridSim

To use the GridSim API for the implementation and simulation of the proposed Makespan Optimization Algorithm we need to follow below mentioned sequential steps; which when together coded in JAVA programming language simulate a heterogeneous Grid Computing Environment.

In our implementation, the simulation system accepts total number of user jobs, processing requirements or MI of those jobs, input size, output size, and the available Grid resources in the Grid environment. Details of the available Grid resources are obtained from Grid Information Service (GIS). GIS keeps track of the resources available in Grid environment. Each Grid resource is described in terms of their various characteristics, such as resource ID, name, total number of machines in each resource, total processing elements (PE) in each machine, MIPS of each PE, and bandwidth speed.

After gathering the details of user jobs and the available resources, the system randomly creates jobs according to the given data. Eventually, the scheduler will submit the job according to proposed algorithm to the corresponding resources. The tests are conducted using four resources of different MIPS. The MIPS of each resource is computed as follows:

*Resource MIPS = Total_PE * PE_MIPS,*
Where *Total_PE* = Total number of Processing Elements of the resource, and
PE_MIPS = MIPS of Processing Element

Each resource has its own predefined cost rate for charges imposed on a Grid user for executing the user jobs on that resource. In the simulation, the total processing time is calculated in milliseconds. In real world, the overhead time for each job depends on the current network load and speed. The total processing cost is computed based on the actual CPU time taken by the grid resource for computing the gridlets and the cost rate specified by that grid resource, as summarized below:

*Process_Cost = T * C*, where
T = Total CPU Time for Gridlet execution, and
C = Cost per second for executing Gridlet on the grid resource.

The proposed algorithm is compared with FCFS (first come first serve) algorithm Random Allocation algorithm, Smallest Job Slowest Resource Allocation Algorithm and Longest Job Slowest Resource Allocation Algorithm using simulation data Summation of actual CPU times gives the overall makespan i.e.

Makespan=Σ Actual CPU time taken by gridlets on grid resources.

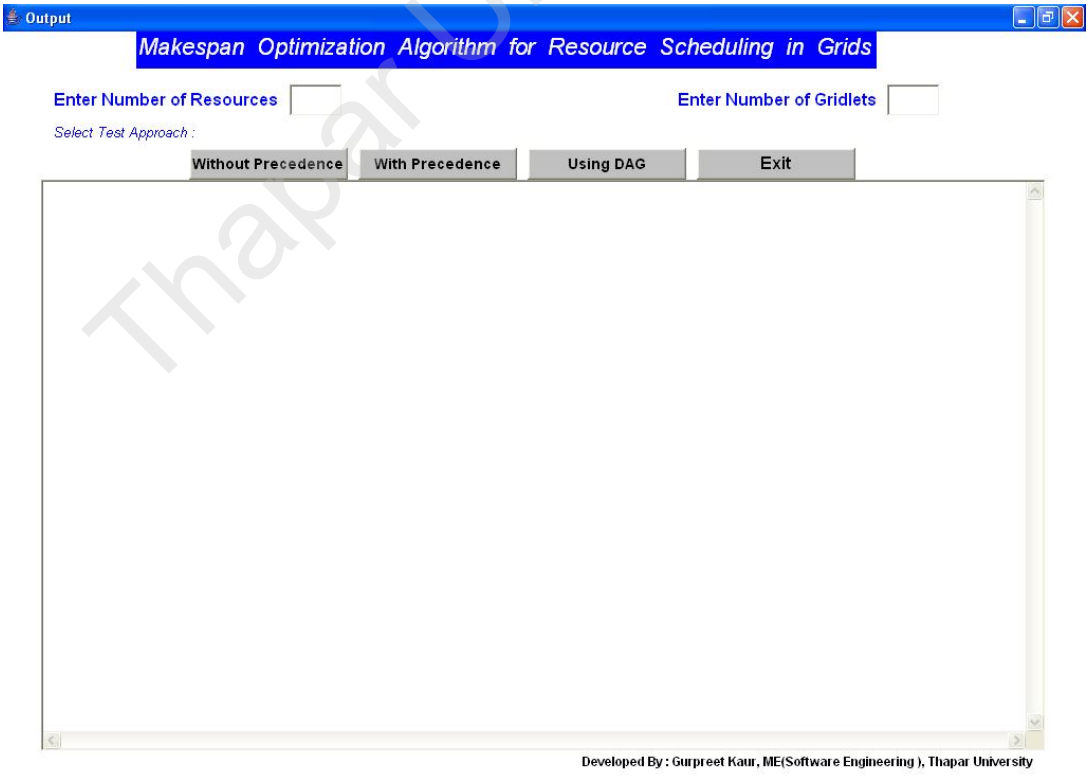
According to simulation results the proposed algorithm gives a significant improvement and hence optimizes the makespan.

Chapter 5
Experimental Results

This chapter describes the experimental results obtained after implementing the Makespan Optimization Algorithm. The algorithm is implemented in Java using Netbeans IDE and run on top of GridSim version 4.1 Toolkit. It takes as input the required set of heterogeneous resources and a set of Gridlets. The proposed algorithm is compared with FCFS (first come first serve) Algorithm and Random Allocation Algorithm, Smallest Job Slowest Resource Allocation Algorithm and Longest Job Slowest Resource Allocation Algorithm using simulation data. The simulation results are shown in the form of screen shots.

5.1 The User Input

In the starting window the user is prompted to enter the values for the number of Gridlets and the number of Grid Resources needed to be simulated.



Screenshot 5.1: User Interface for the Proposed Makespan Optimization Algorithm

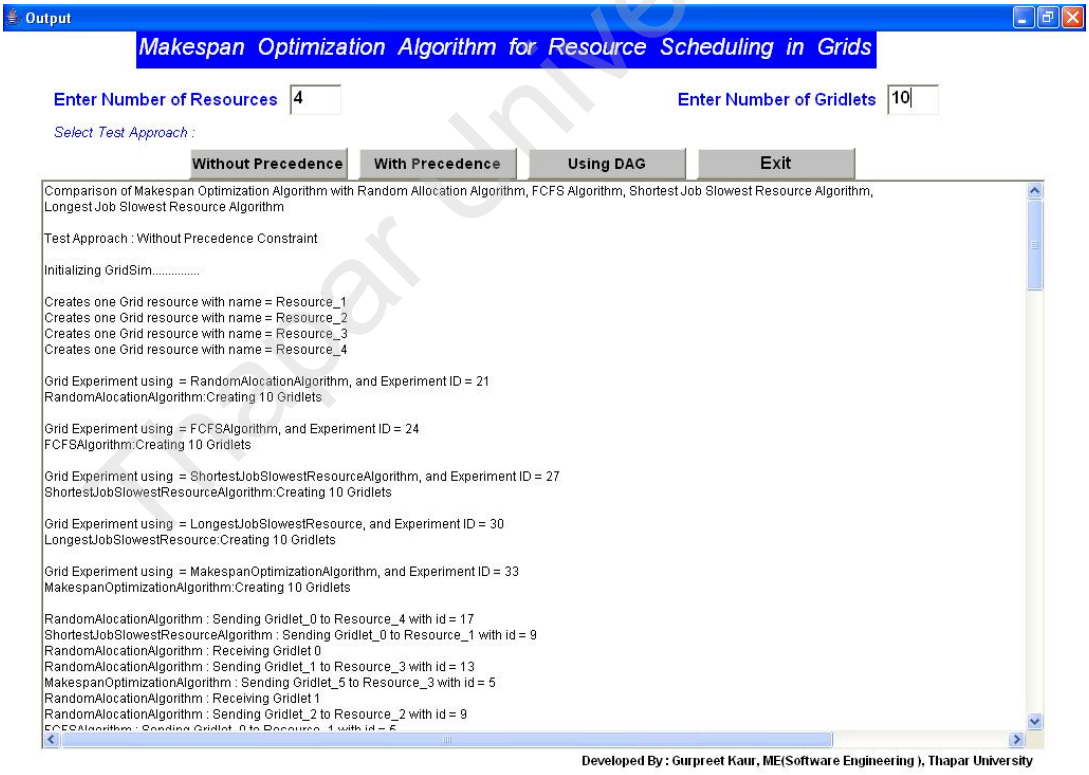
5.2 Test Approach Selection

In the second step the user is prompted to select for the following test approach:

I. Without Precedence: Test case in which gridlets (jobs) that are to be scheduled do not have any precedence constraint among themselves.
II. With precedence: Test case in which gridlets (jobs) have precedence constraints. Precedence constraints between jobs are given in the form of a partial order. If j < k, processing of job k cannot start until job j's execution is completed.
III. Using DAG: Test case in which DAG (Directed Acyclic Graph) approach is applied.

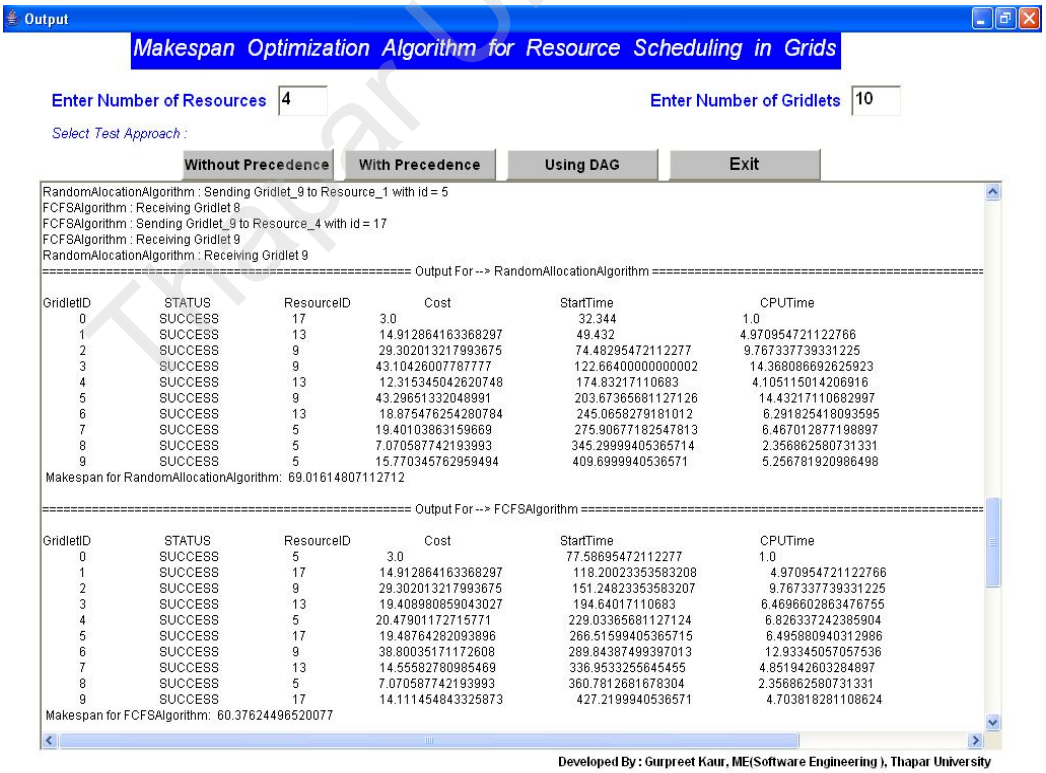
5.2.1 Test Approach: Without Precedence

In this test case gridlets do not have any precedence constraint among themselves.

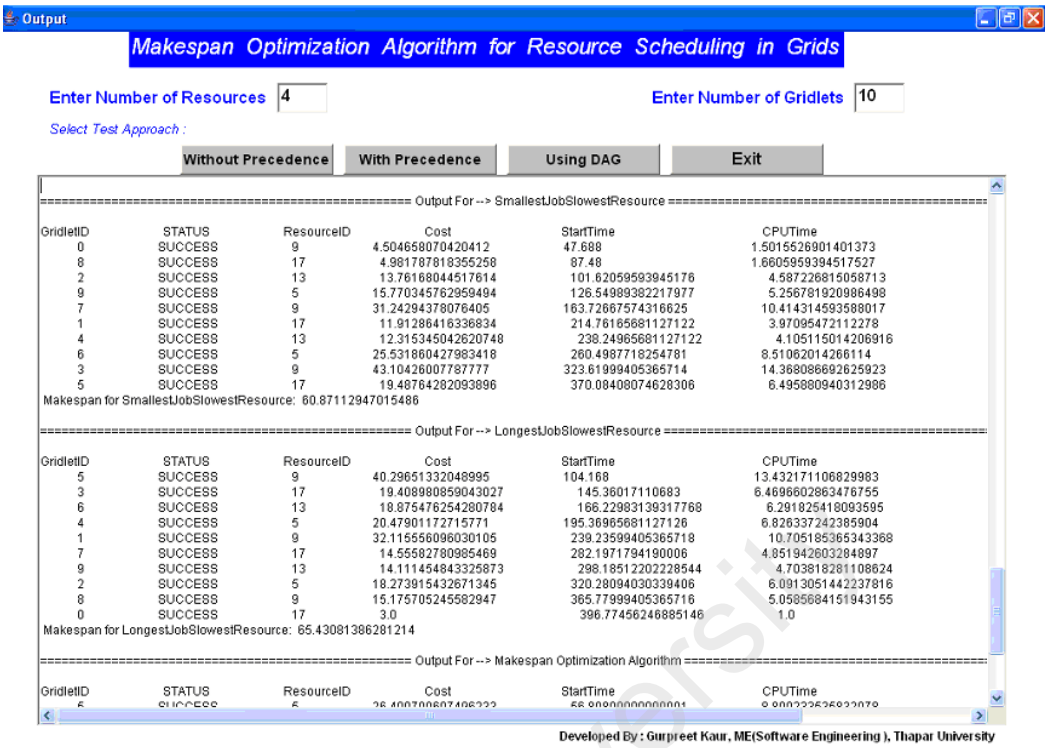


Screenshot 5.2:Test case considering Jobs Without Precedence Constraint (1/5)

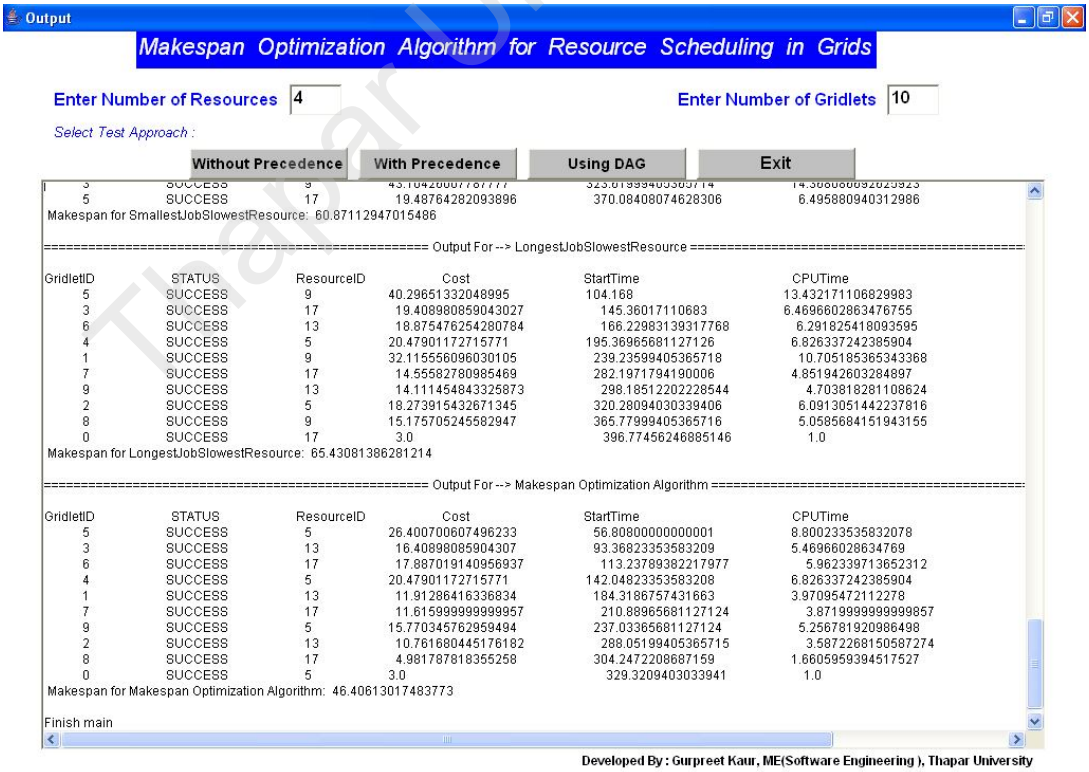
Screenshot 5.3:Test case considering Jobs Without Precedence Constraint (2/5)



Screenshot 5.4:Test case considering Jobs Without Precedence Constraint (3/5)



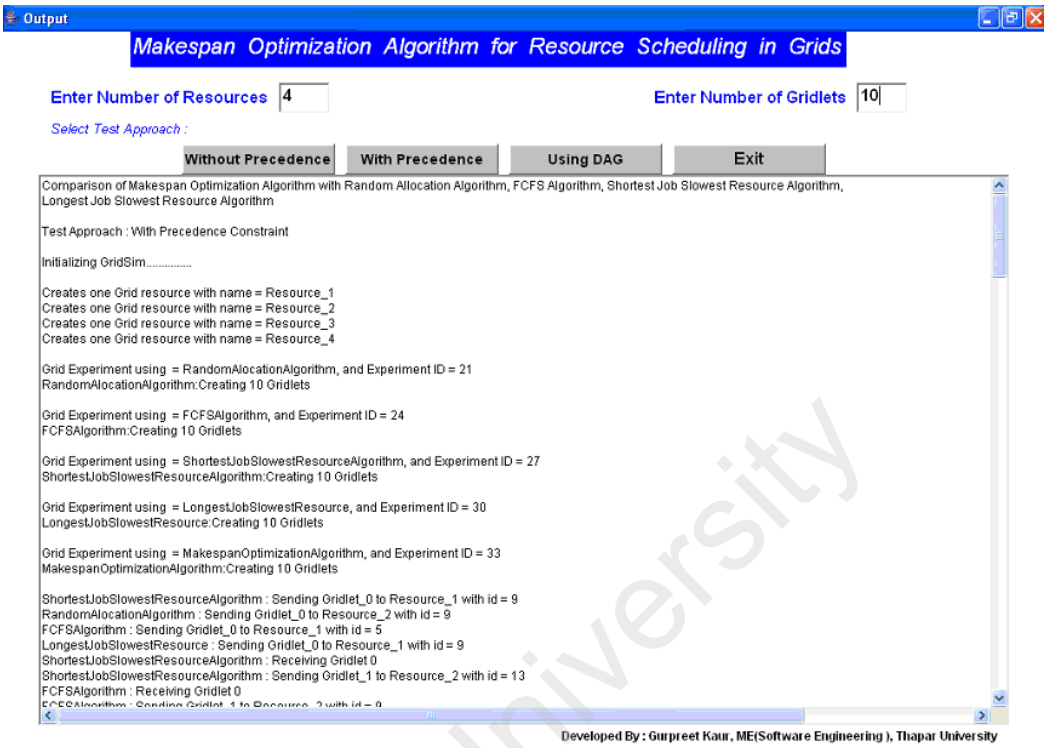
Screenshot 5.5:Test case considering Jobs Without Precedence Constraint (4/5)



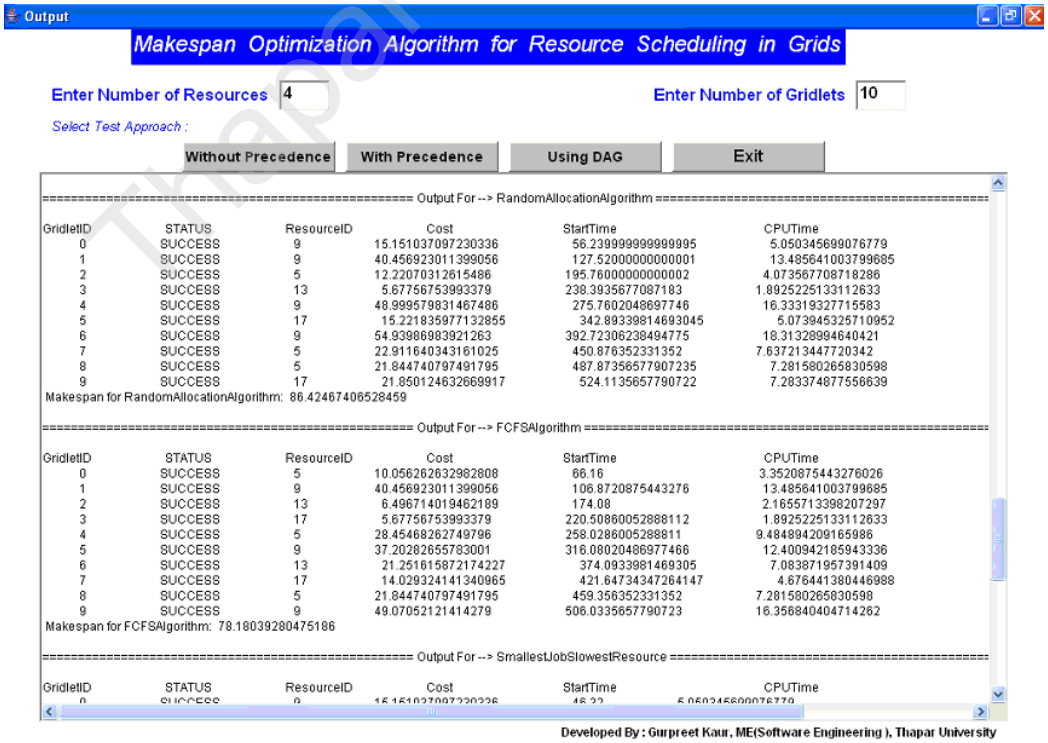
Screenshot 5.6:Test case considering Jobs Without Precedence Constraint (5/5)

5.2.2 Test Approach: With Precedence

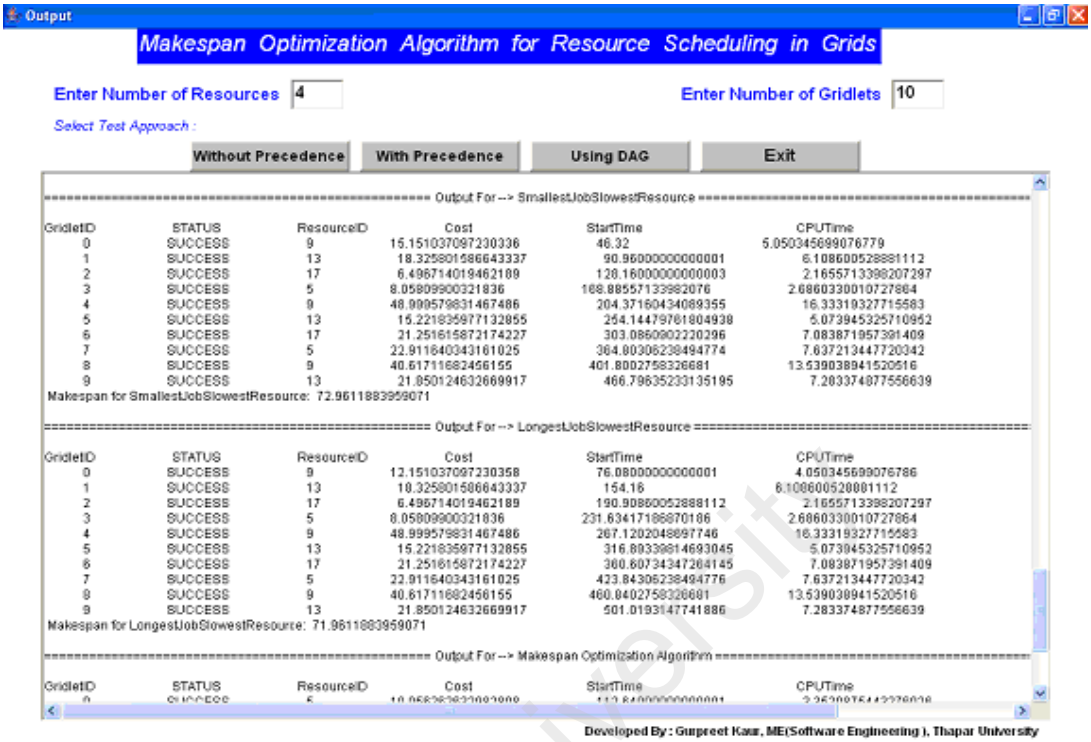
In this test case Gridlets have precedence constraint among themselves.



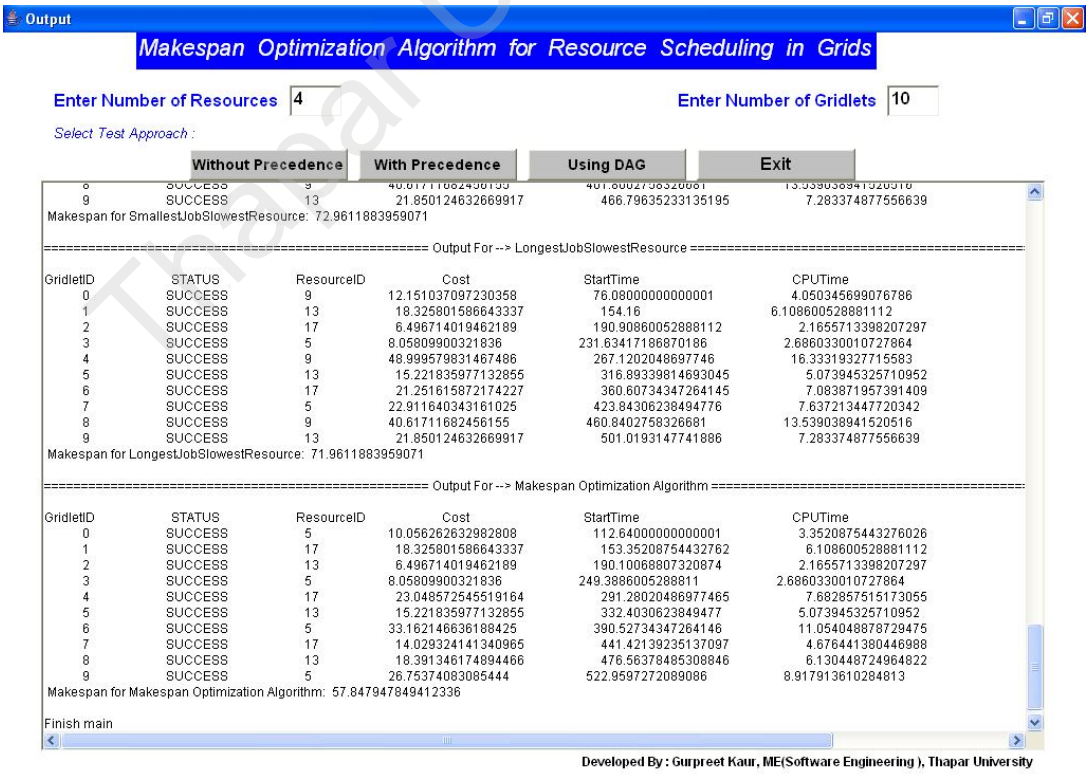
Screenshot 5.7:Test case considering Jobs With Precedence Constraint (1/4)



Screenshot 5.8:Test case considering Jobs With Precedence Constraint (2/4)

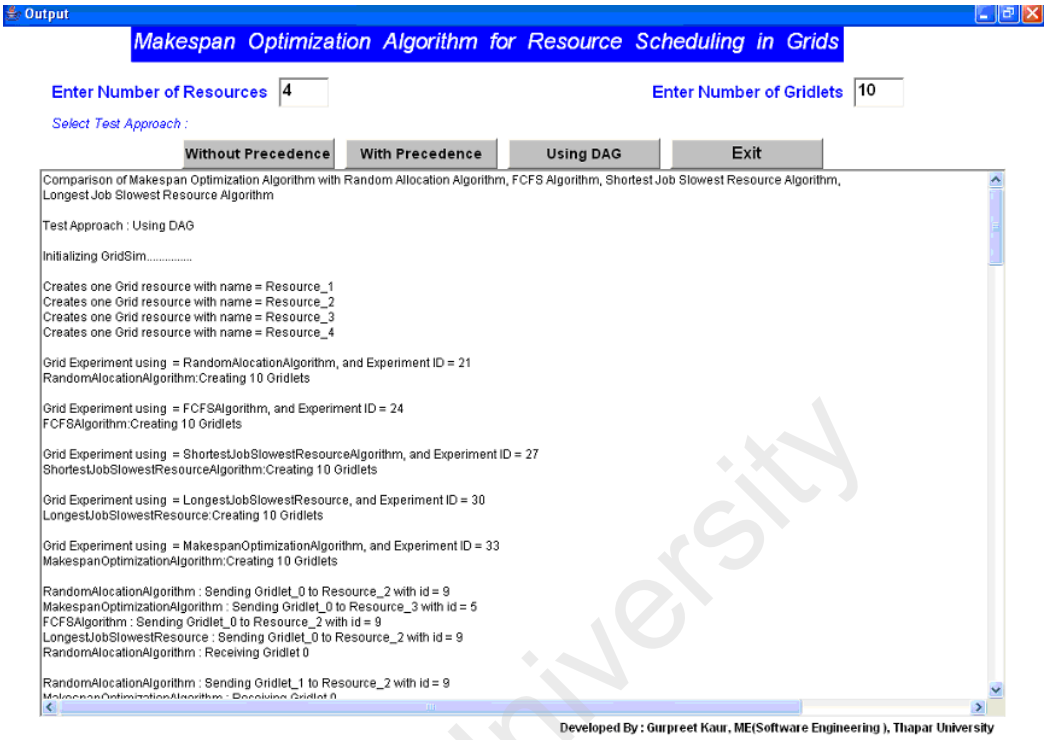


Screenshot 5.9:Test case considering Jobs With Precedence Constraint (3/4)

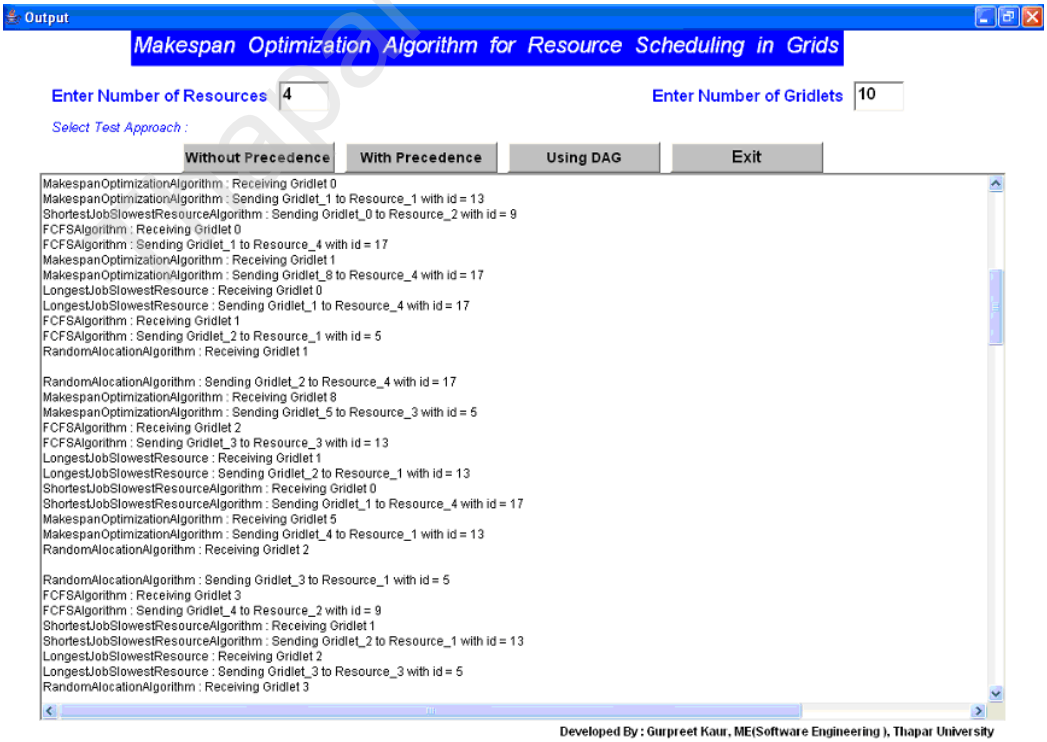


Screenshot 5.10:Test case considering Jobs With Precedence Constraint (4/4)

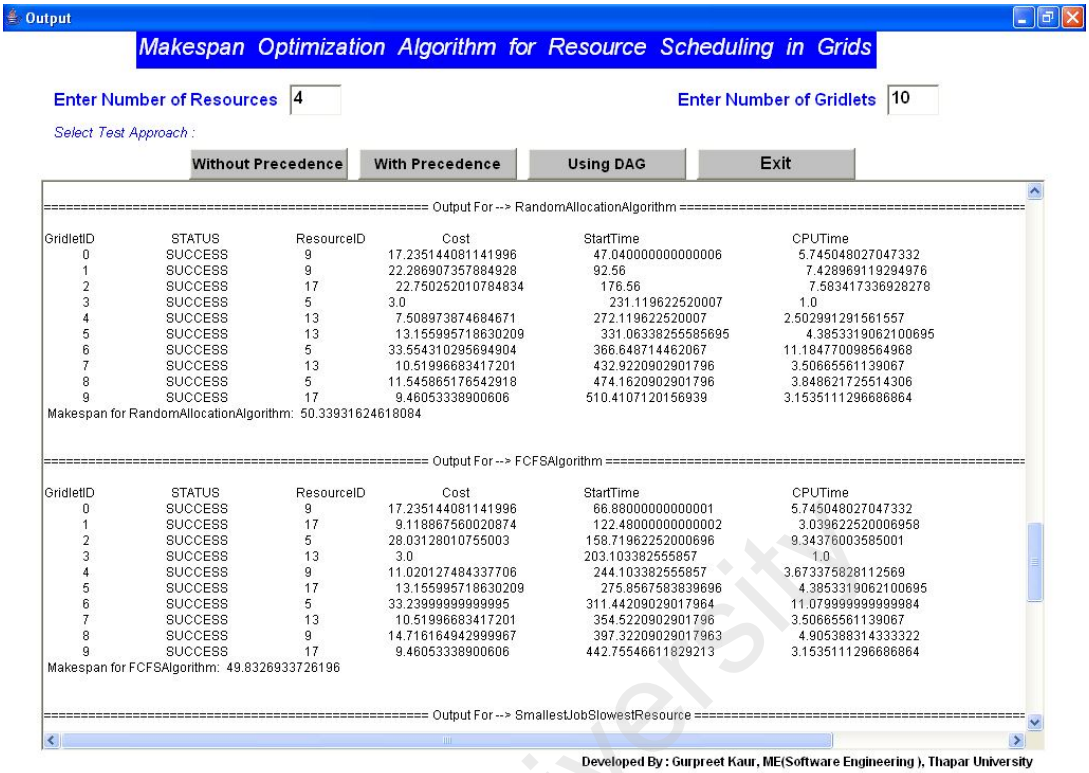
5.2.3 Test Approach: Using DAG



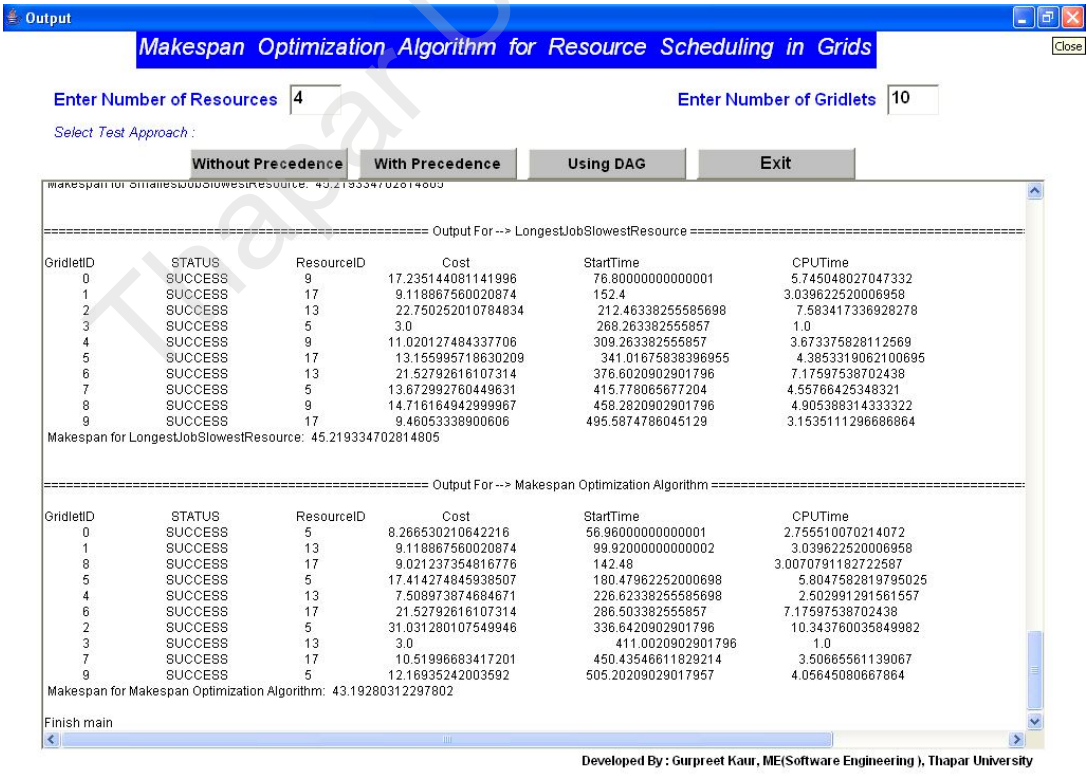
Screenshot 5.11:Test case considering Jobs Using DAG Approach (1/4)



Screenshot 5.12:Test case considering Jobs Using DAG Approach (2/4)



Screenshot 5.13:Test case considering Jobs Using DAG Approach (3/4)



Screenshot 5.14:Test case considering Jobs Using DAG Approach (4/4)

5.3 Graphical Analysis of Experimental Results

In this section we analyze the simulation results using graphs. Graphical data consists of 10 gridlets that are scheduled on 4 heterogeneous resources.

5.3.1 Graphical Results for the Proposed Makespan Optimization Algorithm

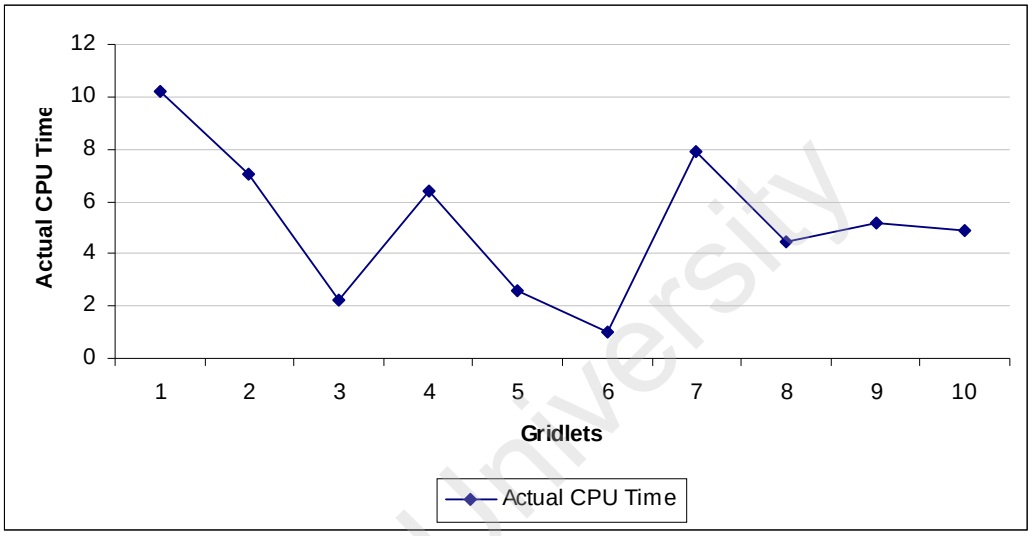


Figure 5.1: Line Chart showing Actual CPU Time for each Gridlet

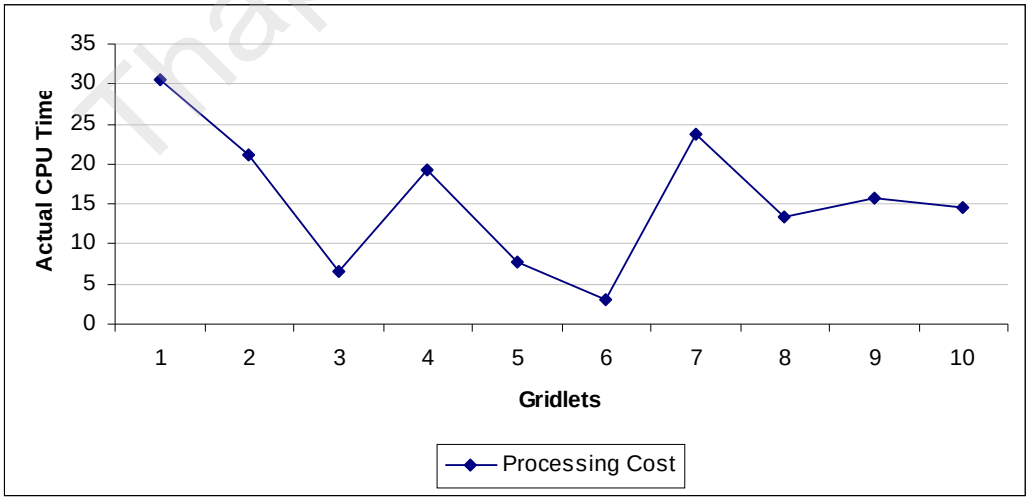


Figure 5.2: Line Chart showing Processing Cost for each Gridlet

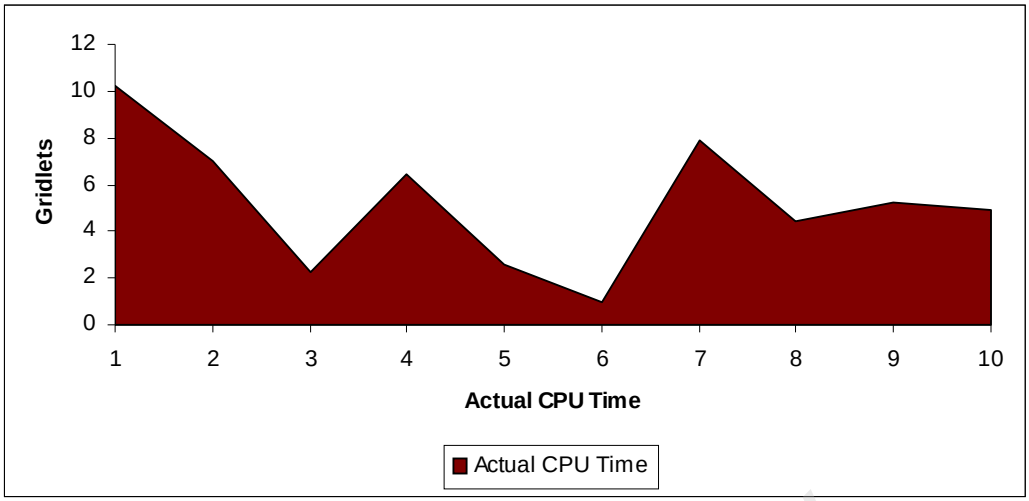


Figure 5.3: Stacked Area Curve for Actual CPU Time where the Area under the curve Shows the Makespan

5.3.2 Graphical Results for Without Precedence Test Approach

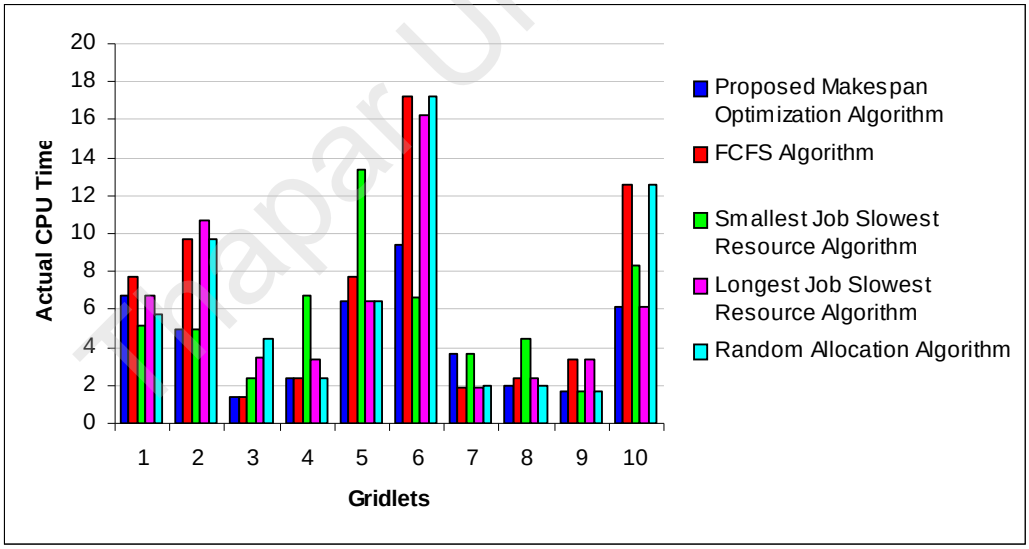


Figure 5.4:Graph showing Actual CPU Time for each Gridlet considering Jobs Without Precedence Constraint

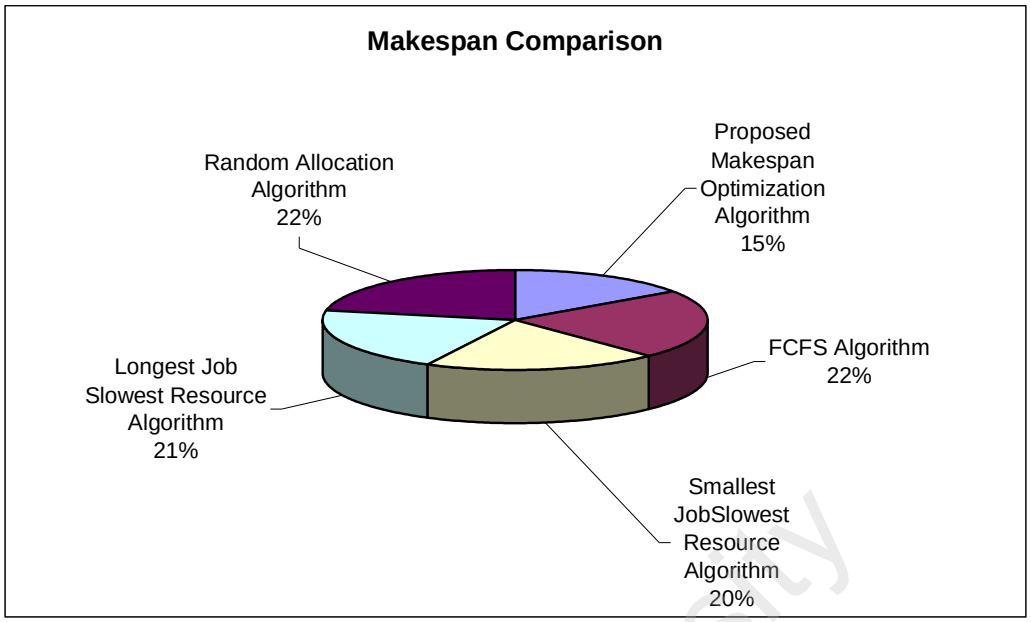


Figure 5.5:Pie chart Comparison of Makespan considering Jobs Without Precedence Constraint

5.3.3 Graphical Results for With Precedence Test Approach

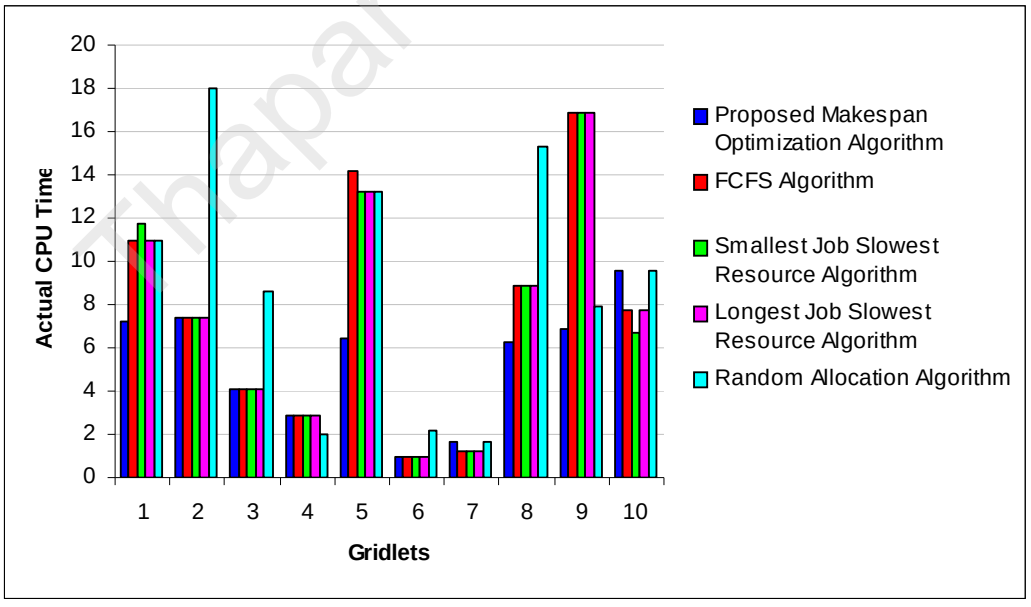


Figure 5.6:Graph showing Actual CPU Time for each Gridlet considering Jobs With Precedence Constraint

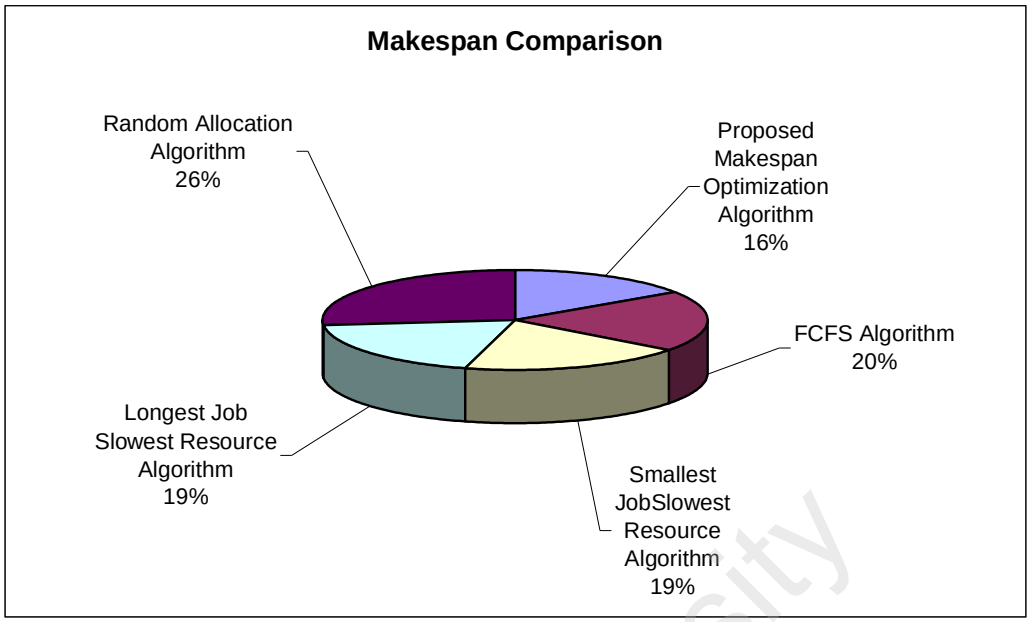


Figure 5.7:Pie chart Comparison of Makespan considering Jobs With Precedence Constraint

5.3.4 Graphical Results using DAG Approach

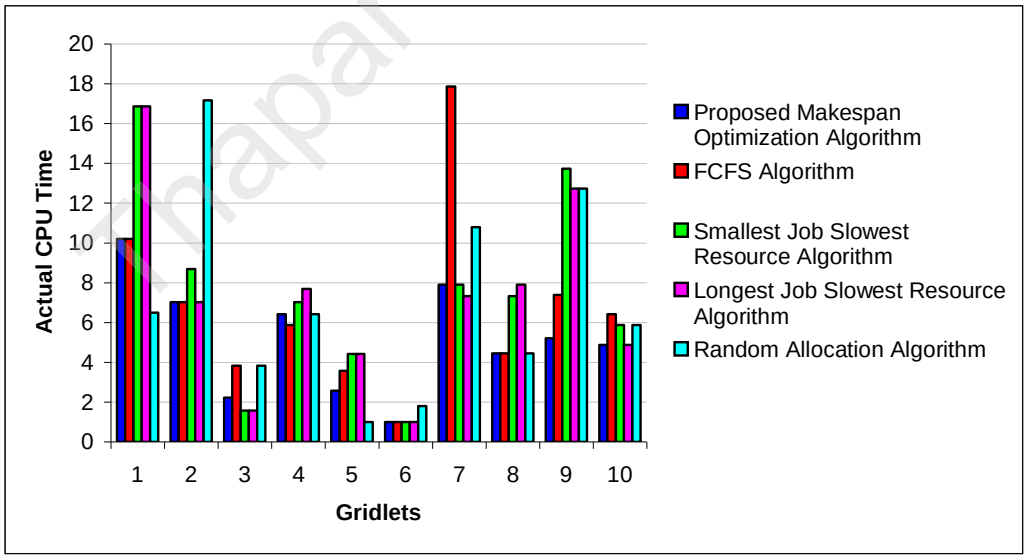


Figure 5.8:Graph showing Actual CPU Time for each Gridlet considering Jobs Using DAG Approach

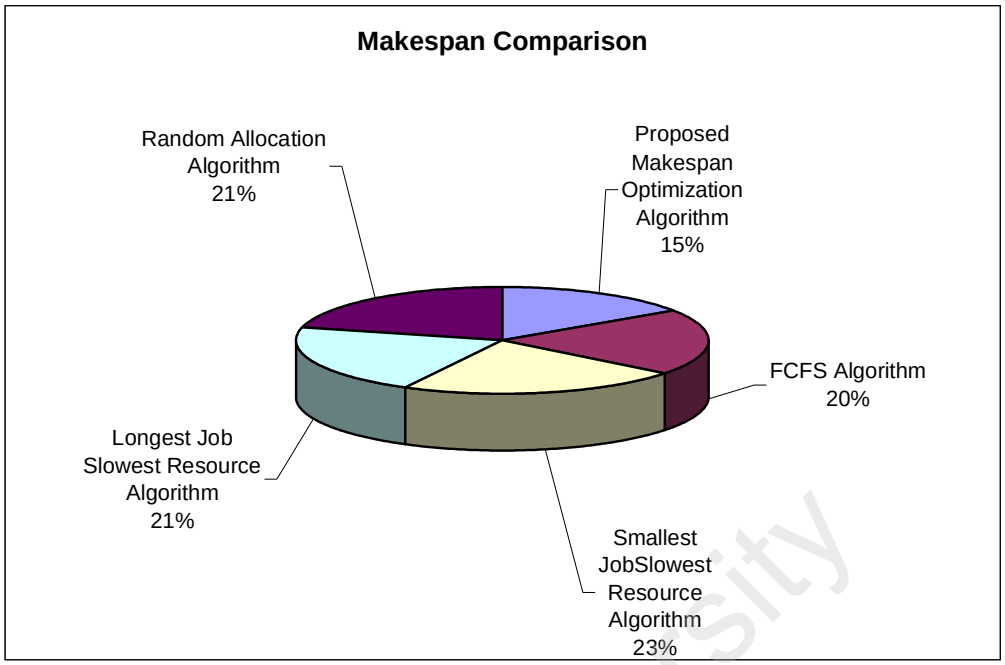


Figure 5.9:Pie chart Comparison of Makespan considering Jobs DAG Approach

Chapter 6

Conclusion and Future Scope

In this Thesis work a Makespan Optimization Algorithm for Grid Scheduling is proposed. The Algorithm is then implemented and successfully simulated on top of open source GridSim version 4.1 Toolkit. The corresponding results are compared with other known algorithms and comparative results show that it optimizes makespan.

The main highlights of the work are:

- The proposed Algorithm makes no assumptions about the architecture of target computing platforms.
- The algorithm is loosely coupled with the internal structure of the application, which makes it generic and scalable.
- The proposed Algorithm is compared with FCFS (first come first serve) Algorithm and Random Allocation Algorithm, Smallest Job Slowest Resource Allocation Algorithm and Longest Job Slowest Resource Allocation Algorithm using simulation data. The simulation results show that it optimizes the makespan for set of jobs.
- The proposed Algorithm assumes availability of finite number of resources in the Grid.
- The proposed Algorithm takes into account the communication costs between Grid Nodes.
- The proposed Algorithm is successfully implemented in JAVA programming Language for complete Platform Independence.
- The proposed Algorithm is simulated on top of GridSim 4.1 Toolkit.
- Data Analysis Visualization for the proposed Algorithm is done using Graphs.
- Use of Open Source Tools & Technologies throughout the design, development and simulation of Makespan Optimization Algorithm.

Future Scope

- The performance of the Algorithm can be improved by designing a mechanism that breaks down the application submitted to the Grid for Execution into a Directed Acyclic Graph.

- The basic structure of Grid Application assumed by the proposed Algorithm is of Directed Acyclic Graph, which is inherently Cycle Free, but still a proactive and dedicated Deadlock Prediction and Avoidance Mechanism can be designed.
- The improvements can be done to cater the dynamic behavior of the grid resources.
- Also we would like to extend this algorithm to include various parameters like option for advance reservation, preemptive jobs as well.
- The current Algorithm has been only simulated on a virtual Grid Environment simulated by GridSim, before all the design goals can be verified and validated this Algorithm should be implemented in a Third Party open source Grid Resource Broker like GridBus Broker, Nimrod-G or Condor-G for testing in a Real Grid Environment.

References

[1] "Grid Computing – Making the Global Infrastructure a Reality", F. Berman, G. Fox and T. Hey, John Wiley & Sons, Ltd, ISBN: 0-470-85319-0,2002.

[2] Moore's Law Explained by Intel <http://www.intel.com/research/silicon/mooreslaw.htm>

[3] "Gilder's law on network performance. Telecosm: The World After Bandwidth Abundance", Gilder, G. ed., Touchstone Books, ISBN: 0743205472, 2002

[4] The Global Grid Forum Web Site, <http://www.gridforum.org>

[5] The World Wide Web Consortium, <http://www.w3c.org>

[6] Protein Data Bank Worldwide Repository for the Processing and Distribution of 3-D Biological Macromolecular Structure Data, <http://www.rcsb.org/pdb/>

[7] MyGrid – Directly Supporting the e-Scientist, <http://www.mygrid.info/>

[8] Biomedical Informatics Research Network BIRN Grid, <http://www.nbirn.net/>

[9] NASA Information Power Grid, <http://www.ipg.nasa.gov/>

[10] Distributed Aircraft Maintenance Environment DAME, <http://www.iri.leeds.ac.uk/Projects/IAProjects/karim1.htm>

[11] IBM Red Book, Dec 2005 Edition. <http://www.redbooks.ibm.com>

[12] <http://www.globus.org>

[13] J. Joseph, M. Ernest, and C. Fellenstein, "Evolution of grid computing architecture and grid adoption models", IBM System Journals, Vol 43, Number 4, 2004

[14] Yanmin ZHU, "A Survey of Grid Scheduling", Department of Computer Science Hong Kong University of Science and Technology.

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

[16] <http://www.unix.cs.anl.gov/`schopf/ggf-sched/wd/scedwd.8.5.pdf>

[17] "Grid Computing: Making the Global Infrastructure a Reality ", Fran Berman, Geoffrey C.Fox, Anthony J.G.Hey, John Willey & Sons, Ltd, ISBN:0-470-85319-0, 2003.

[18] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An architecture for a Resource Management and Scheduling System in a global computational Grid",

in proceedings of the Fourth International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2000.

[19] "Task Scheduling in Parallel and Distributed Systems", H. El-Rewini, T. Lewis, H. Ali, PTR Prentice Hall, ISBN: 0130992356, 1994.

[20] H.Topcuoglu, S.Hariri, M.Y. Wu, "Performance-Effective and Low- Complexity Task Scheduling for Heterogeneous Computing", IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3, pp. 260 - 274, 2002.

[21] R.Sakellariou, H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems", in the Proc. of 18th International Parallel and Distributed Processing Symposium (IPDPS'04), pp.111-123, Santa Fe, New Mexico USA, April 2004

[22] A. Radulescu, A.J.C.Van Gemund, "The Complexity of List Scheduling Algorithms for Distributed Memory Systems", in Proc. Of 13th International Conference on Supercomputing, pp. 68-75, Portland, Oregon, USA, November 1999.

[23] Rajkumar Buyya, Manzur Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid Computing", Concurrency and Computation: Practice and Experience, Concurrency Computat: Pract. Expert, Vol.14, pp.1175–1220, 2002.

[24] Aridor Y, Factor M, Teperman A. "JVM: A single system image of a JVM on a cluster", Proceedings 29th International Conference on Parallel Processing (ICPP 99), Fukushima, Japan, IEEE Computer Society Press, September 1999.

[25] I. Foster and C. Kesselman, "The Globus Project: a Status Report", In Proc. of Seventh Heterogeneous Computing Workshop (HCW 98), IEEE Computer Society Press, March 1998.

[26] <http://compgeom.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/21-nphard.pdf/>.

[27] Jean Christophe Durand, "Grid Computing: A Conceptual and Practical Study", University de Lausanne, Nov 8, 2004.

[28] Fangpeng Dong and Selim G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", School of Computing, Queen's University Kingston, Ontario, Technical Report No. 2006-504, January 2006.

[29] "A Networking Approach To Networking Grid Computing", D. Manoli, John Willey & Sons, Ltd, ISBN-978-0-471-68756-6, 2005.

[30] <http://simos.stanford.edu/>

[31] <http://www.dcs.ed.ac.uk/home/simjava/>

[32] <http://matsu-www.is.titech.ac.jp/takefusa/bricks/>

[33] <http://jcharts.sourceforge.net/>

List of Papers Published/Accepted

- Gurpreet Kaur, Seema Bawa, "Makespan Optimization Algorithm for Grid Environment", 12th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2008), *Kunming, China, October 21-23, 2008.* **(Communicated)**.