Design and Development of Compile Time Task Scheduling Algorithm for Grids

A thesis
submitted in partial fulfillment of the requirements
for the award of degree

Master of Engineering
in
Software Engineering

By
Gaurav Vohra
(Roll No. 8053109)

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

MAY 2007

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004
INDIA

Dedicated to my Parents

Sh. Sat Bhushan Vohra
and
Smt. Swarn Vohra

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Design and Development of Compile Time Task Scheduling Algorithm for Grids"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. (Mrs.) Seema Bawa**.

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

(Gaurav Vohra)

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

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

Countersigned by

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

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

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. I am deeply indebted to my parents and friends for their inspiration and ever encouraging moral support, which enabled me to pursue my studies.

This work would not have been possible without the encouragement and able guidance of my supervisor **Dr (Mrs.) 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. Her feedback and editorial comments were also valuable for writing this thesis. Also I shall be failing in my duties if I do not express my deep sense of gratitude towards **Dr. Maninder Singh** who has been a constant source of inspiration for me throughout this work.

I would like to express my sincere gratitude towards 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.

(Gaurav Vohra)
(8053109)

Abstract

Grid Computing is an evolving computing infrastructure that promises to transform the way organizations and individuals compute, communicate and collaborate. It offers untapped processing cycles from networks of computers spanning vast geographical boundaries.

Grid Resource Scheduling falls under the realms of NP-Complete problems in Computer Science and Engineering. A wide variety of Grid Scheduling Algorithms have been proposed and implemented in the literature of context, each having its own pros and cons as only a heuristic and approximate solution is possible to NP-Complete problems.

Compile Time Task Scheduling follows a system-centric approach to Schedule Grid applications aiming for optimization of the overall performance of the Grid system. It follows a centralized and heuristic based approach to make Scheduling decisions under the Quality of Service constraints.

This thesis work focuses on the design, development and simulation of a Compile Time Task Scheduling Algorithm for Grid Computing Environment. The proposed Algorithm i.e. SCH takes into account the high degree of heterogeneity among the target Grid Computing Platforms and applications submitted to the Grid. It also takes into account the communication costs between Grid Nodes. The SCH Algorithm aims to minimize the job submission time to the Grid Nodes. The SCH Algorithm is simulated on Open Source GridSim Toolkit 4.0 and JCharts 0.7.5 API is employed for statistical trend visualization of simulation Data. The outcomes of the SCH Algorithm simulation on GridSim Toolkit under different test scenarios show that the Wait Time for job submission to the Grid can be optimized and thus become negligible by applying the heuristics of the SCH Algorithm.

Organization of Thesis

The First Chapter briefly introduces Grid Computing concepts. It explains concept of Grid Computing with relevant examples, Benefits of Grid Computing, Grid Topologies and Grid Scheduling as a thrust area in Grid Computing. Lastly it layouts the broad objectives of this Thesis work.

The Second Chapter presents Review of Literature in Grid Scheduling. It also covers Challenges in Grid Scheduling, Generic Grid Scheduler Architecture and Steps involved in Grid Scheduling. Besides this a Taxonomy of Current Grid Scheduling Approaches is also presented with summary of their respective advantages and disadvantages. Further motivation using for Grid Scheduling Simulation Tools and their Taxonomy is also covered.

The Third Chapter covers Problem Formulation for this thesis work and explains the Proposed Solution Design for the former. The proposed SCH Scheduling Algorithm is presented with complete terminology, pseudo code 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 4.0 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 Concept of Grid Computing

In today's complex world of high speed computing, computers have become extremely powerful, even home-based desktops are powerful enough to run complex applications. But still we have numerous complex scientific experiments, advanced modeling scenarios, genome matching, astronomical research, a wide variety of simulations, complex scientific & business modeling scenarios and real-time personal portfolio management, which require huge amount of computational resources. To satisfy some of these aforementioned requirements, Grid Computing is born.

The Grid is emerging as a wide-scale, distributed computing infrastructure that promises to support resource sharing and coordinated problem solving in dynamic, multi-institutional Virtual Organization [1].

Grid Computing is applying the resources of many computers in a network for a single problem at the same time - usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. Grid Computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing.

Grid Resources fall into the categories of computation (i.e. a machine sharing its CPU), storage (i.e. a machine sharing its RAM or disk space), communication (i.e. sharing of bandwidth or a communication path), software and licenses and special equipment (i.e. sharing of devices).

Example of a Grid Application

A typical example of a Grid Application is "weather prediction". This involves collaboration between several partners: TV stations that produce regular weather news reports, a Satellite Company that regularly provides space images of the earth, a super computing center that rapidly analyses the images and a visualization center that produces visual interpretations of the weather analysis (Figure 1.1). The smooth

running of this project for the timely production of regular weather reports crucially depends on appropriate schemas for securely sharing, exchanging, and coordinating information between these partners.

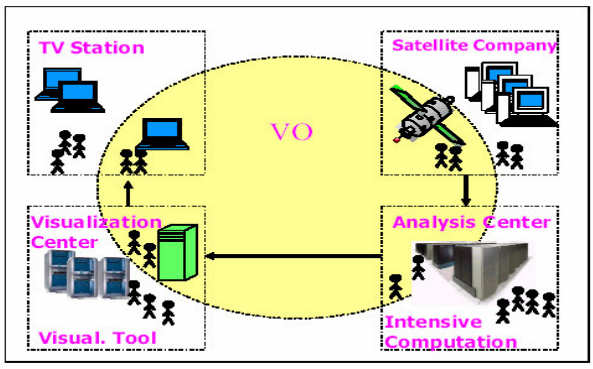


Figure 1.1: Weather prediction using Grid [4]

The power of Grid is particularly useful in areas involved in intensive processing such as life science research [4], financial modeling [4], industrial design [4] and graphics rendering [4].

1.2 Benefits of Grid Computing

1.2.1 Exploiting Unused and Underutilized Resources

Many organizations have a vast amount of compute resources available at their disposal. A large number of these resources are idle for long periods of time, especially the desktop machines. Grid Computing provides a framework that allows these resources to be exploited. This concept applies to any type of resource shared on the Grid, such as disk storage. If an application requires a large amount of disk storage, the Grid framework could be used to aggregate several disk stores into a single virtual store in order to increase storage space or improve performance [5].

1.2.2 Access to Additional Resources

A Grid allows its users to access resources that the user might not normally have access to. For example, an organization might have a Grid machine connected to an electron microscope or a telescope. If the appropriate policies are in place, Grid users who do not normally have access to such equipment might be given access to it, whether they belong to the organization or not [5].

1.2.3 Balanced Resource Utilization

The Grid framework can be used in order to improve resource utilization. For example, jobs can be scheduled to run on idle machines or machines with low activity levels. Grids also offer load balancing. If jobs running on the 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 [5].

1.2.4 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 [5].

1.2.5 Collaboration through Virtualization

Grid Computing supports collaboration by defining standards which allow groups of heterogeneous systems to band together and present themselves as a larger virtual organization. Grid users can be organized dynamically into a number of virtual organizations, each having their own policy requirements and sharing their resources as part of a larger Grid [1]. Resources are virtualized in order to increase interoperability between heterogeneous systems [5].

1.2.6 Increased Reliability

Grids offer a certain amount of reliability at a low cost; no expensive or proprietary hardware or software is required. A failure in one part of the Grid is unlikely to affect other parts of the Grid. If a job is unsuccessful due to a failure and the failure is detected, Grid Management software can automatically re-submit the job for execution at another site. Jobs can even be submitted multiple times in order to ensure that at least one copy of the job executes successfully [5].

1.3 Grid Architecture

The architecture of the Grid is often described in terms of layers, each providing a specific function. In general, the higher layers are focused on the user, whereas the lower layers are more focused on computers and networks [6].

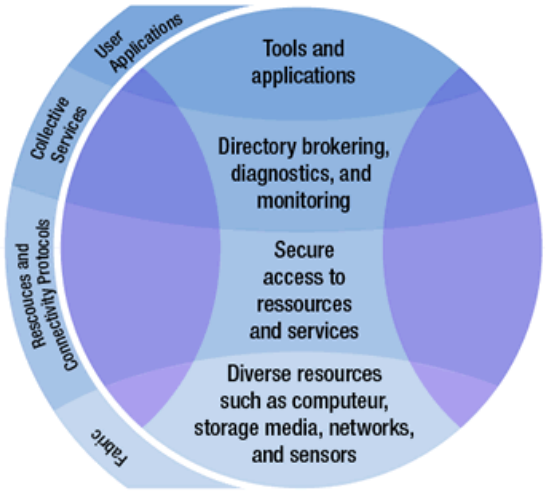


Figure 1.2: Layered Grid Architecture [6]

Fabric Layer

This layer represents all the physical infrastructure of the Grid, including computers and the communication networks. It is made up of the actual resources that are part of the Grid, such as computers, storage systems, electronic data catalogues and even sensors such as telescopes or other instruments, which can be connected directly to the network.

Resource and Connectivity Protocols

Resource and connectivity protocols handle all "Grid specific" network transactions between different computers and other resources on the Grid. Grids have to be able to recognize messages that are relevant to them, and filter out the rest. This is done with communication protocols, which let the resources speak to each other, enabling exchange of data, and authentication protocols, which provide secure mechanisms to verify the identity of both the users and the resources involved [6].

Collective Services

The collective services are also based on protocols: information protocols, which obtain information about the structure and state of the resources on the Grid, and management protocols, which negotiate access to resources in a uniform way. The services include:

- Keeping directories of available resources updated at all times
- Brokering resources
- Monitoring and diagnosing problems on the Grid

- Replicating key data so that multiple copies are available at different locations
- Providing membership/policy services for keeping track on the Grid of who is allowed to do what and when [6].

Applications Layer

The highest layer of the structure is the application layer, which includes all different user applications (science, engineering, and business, financial), portals and development toolkits supporting the applications. This is the layer that users of the Grid will see and interact with [6].

1.4 Grid Types

Grids can be classified on the basis of two parameters

- Grid Functionality
- Grid Topology

1.4.1 Functional Classification

According to the distinctly targeted application realms, Grid systems can be classified into two categories [7]. 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
- Data Grid

Computational Grid

A computational Grid is a system that aims at achieving higher aggregate computational power than any single constituent machine. A high throughput Grid aims to increase the completion rate of a stream of jobs through utilizing available idle computing cycles. [7].

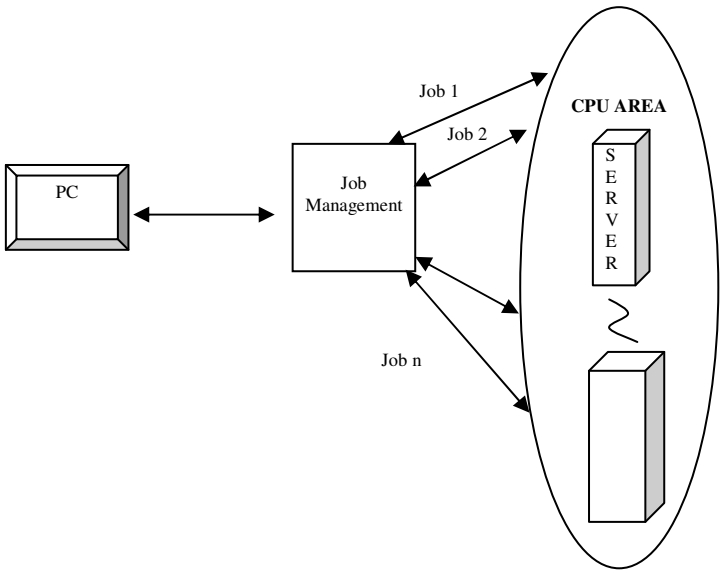


Figure 1.3: Visualization of a Computational Grid [9]

Data Grid

A Data Grid is responsible for housing and providing access to data across multiple organizations. Users are not concerned with where this data is located as long as they have access to the data.

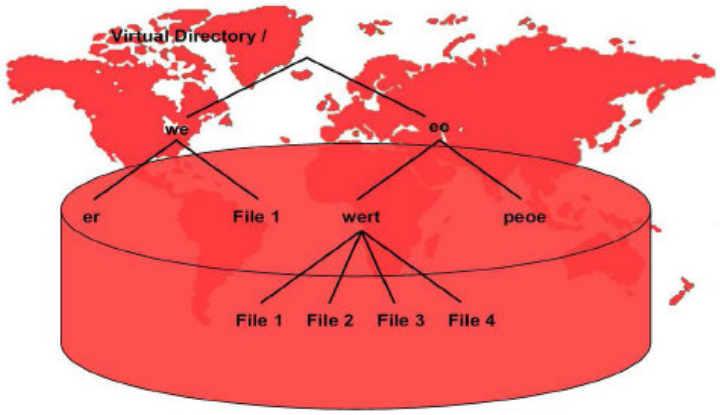


Figure 1.4: Visualization of a Data Grid [9]

A Data Grid would allow them to share their data, manage data and manage security issues [7].

1.4.2 Topology Classification

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 [7]; these are IntraGrid, ExtraGrid and InterGrid.

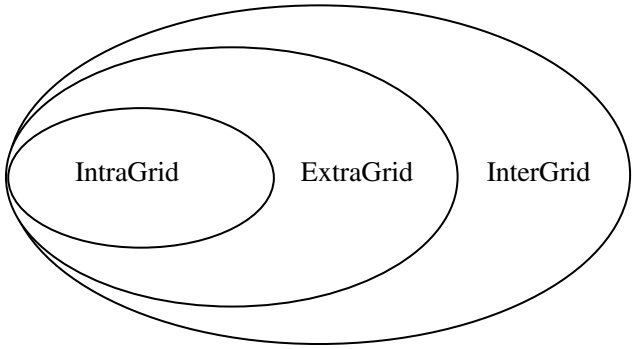


Figure1.5: Grid Types [2]

IntraGrid

A typical IntraGrid topology exists within a single organization. The single organization could be made up of a number of computers that share a common security domain, which are connected by a private high-speed local network.

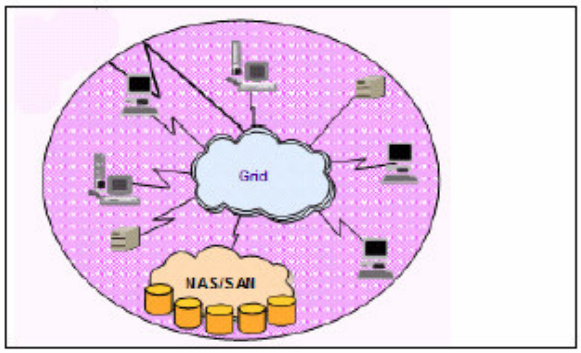


Figure 1.6: IntraGrid [8]

The primary characteristics of an IntraGrid are a single administrative domain and bandwidth guarantee on the private network. Within an IntraGrid, it is easier to design the scheduling system, since an IntraGrid provides a relatively static set of computing resources and communication capability between machines [7].

ExtraGrid

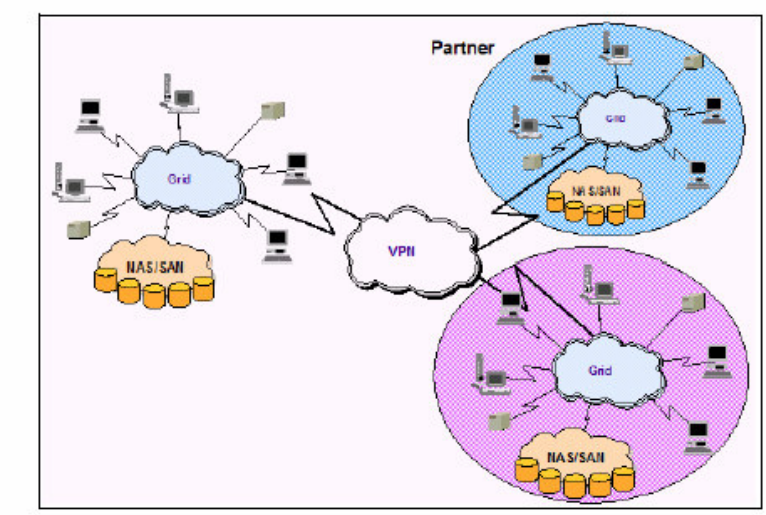


Figure 1.7: Extragrid [8]

An ExtraGrid couples two or more IntraGrids. The ExtraGrid typically involves more than one administrative domain, and the level of management complexity increases. The primary characteristics of an ExtraGrid are dispersed security, multiple domains, and remote/WAN connectivity. Within an ExtraGrid, the resources become more dynamic [7].

InterGrid

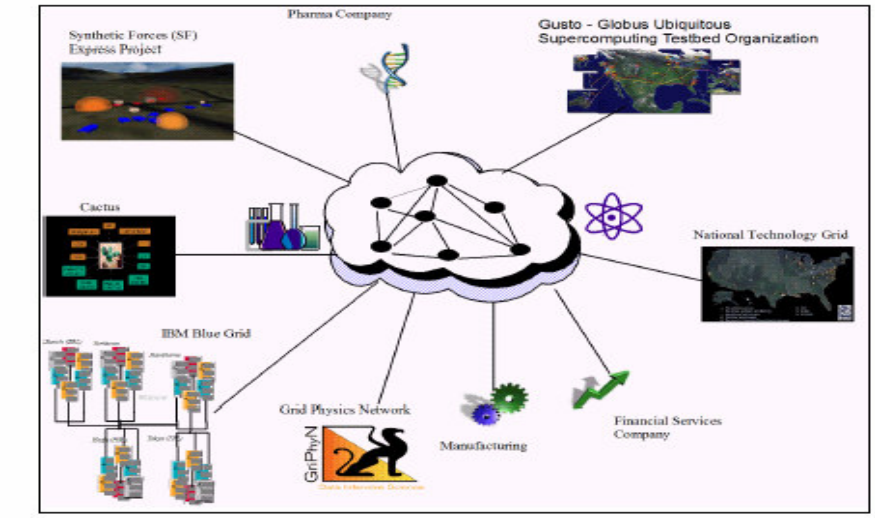


Figure 1.8: Intergrid [8]

An InterGrid has an analogy with the Internet. It is the most complicated form of Grid topology. The primary characteristics of an InterGrid include dispersed security, multiple domains and WAN connectivity [7].

1.5 Comparison with Other Distributed Computing Technologies

1.5.1 Comparison with Cluster Computing

- Resources in a Cluster lie in the same administrative domain, where as in a Grid it's hardly the case.
- Grids consist of heterogeneous resources, Clusters deal with homogenous resources that too only computational in nature.
- Grids are dynamic in nature, resources come and go in a Grid where as Clusters are static in nature.
- Grids are inherently distributed over a local, metropolitan, or wide-area network. Usually, clusters are physically contained in the same complex in a single location.
- Cluster interconnection technology delivers extremely low network latency, which causes problems if clusters are not close together and also makes Cluster scalability an issue. Grids are dynamic and hence more scalable.

1.5.2 Comparison with CORBA

CORBA shares surface level similarities with Grid Computing due to the strategic relationship between Grid Computing and Web Services in the Open Grid Services Architecture (OGSA). Both are based on the concept of Service-Oriented Architecture (SOA). A key distinction between CORBA and Grid Computing is that only CORBA assumes object orientation. In OGSA, there isn't a presumption of object-oriented implementation in the architecture. The architecture is message oriented; object orientation is an implementation concept.

1.5.3 Comparison with P2P

Grids usually have some form of centralized management and security which P2P systems lack, making them ideal for providing anonymity. Also:

- P2P systems are generally far more scalable than Grid Computing systems.
- P2P systems are generally more tolerant of single-point failures than Grids.

1.6 Thrust Areas in Grid Computing

Grid Computing has following six core thrust areas [9]:

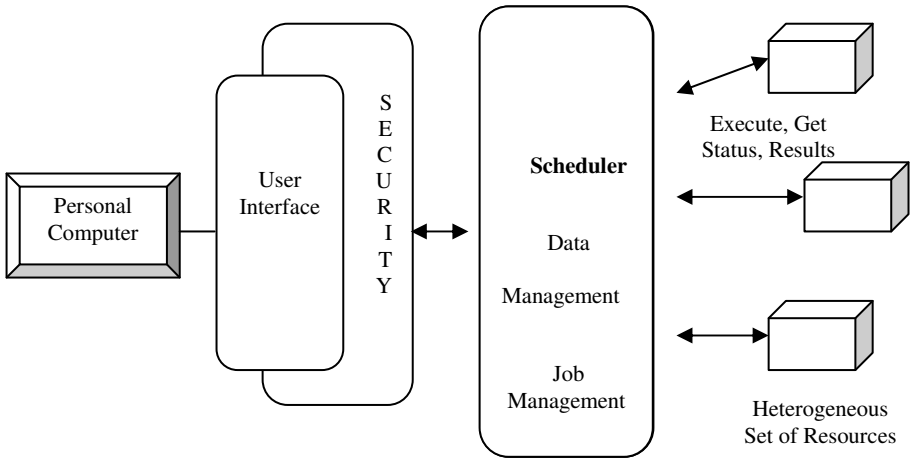


Figure 1.9: Components of Grid [9]

1.6.1 User Interface

Accessing information on the Grid is fairly important, and the user interface component handles this task for the user. It often comes in one of two ways:

- An interface provided by an application that the user is running.
- An interface provided by the Grid administrator, much like a Web portal.

1.6.2 Security

Computers on a Grid are networked and running applications; they can also be handling sensitive or extremely valuable data, so the security component of Grid Computing is of crucial concern. This component includes elements such as encryption, authentication, and authorization.

1.6.3 Workload management

Applications that a user wants to run on a Grid must be aware of the resources that are available; this is where a workload management service comes in handy. An application can communicate with the workload manager to discover the available resources and their status.

1.6.4 Data management

If an application is running on a system that doesn't hold the data the application needs, a secure, reliable data management facility takes care of moving that data to the right place across various machines, encountering various protocols.

1.6.5 Scheduler

A scheduler is needed to locate the computers on which to run an application, and to assign the jobs required. 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. In general the task of scheduling on a set of heterogeneous resources in Grid is NP-Complete [20] problem. The Grid Scheduling Algorithm is the crux of the Grid Scheduler. A large variety of Grid Scheduling Algorithms have been proposed in the literature of the context.

1.7 Objective of the Thesis

The objective of this thesis work is to design, develop, implement and subsequently simulate a Scheduling Algorithm for Grids. Main tasks undertaken and methodology followed for this virtue are:

- A thorough study of the current Grid Scheduling Algorithms and approaches.
- A survey of Grid Simulation Tools and Techniques.
- Design and Development of Grid Scheduling Algorithm aiming to overcome shortcomings found in first step.
- Implementation and Simulation of the designed Grid Scheduling Algorithm.
- Visualization of the Experimental Results and drawing appropriate conclusions henceforth.

Throughout the thesis work emphasis has been on the use of Open Source Tools & Technologies.

Chapter 2

Review of Literature: Grid Scheduling

Grid Scheduling is defined as the process of making Scheduling decisions involving resources over multiple administrative domains. 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.

2.1 A Generic Grid Scheduler Architecture

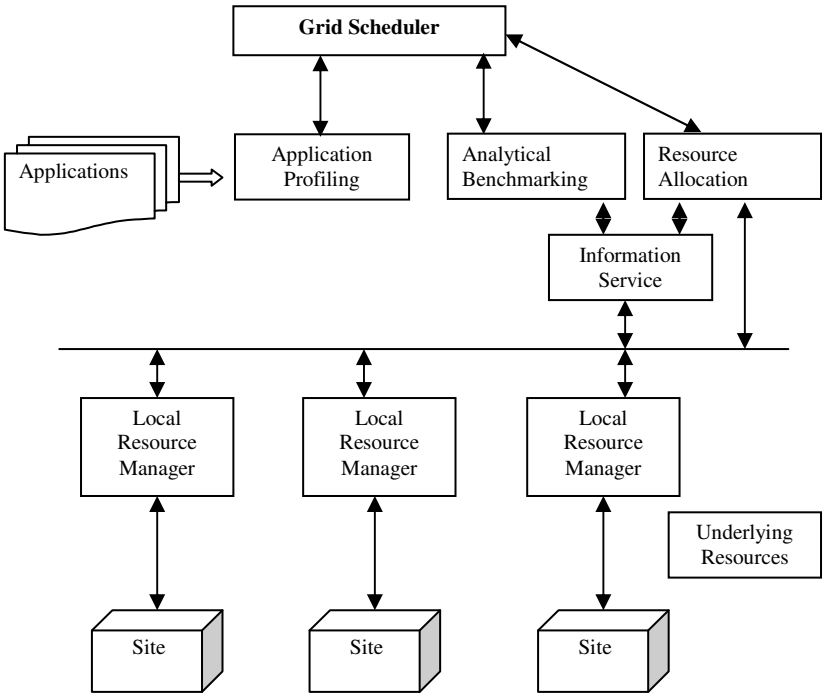


Figure 2.1: General Grid Scheduler Architecture [10]

The generic Grid Scheduler architecture provides a high-level view of Grid Schedulers. Not all existing Grid Scheduling systems have corresponding components that appear in the architecture. But every component seems necessary for any comprehensive Grid Scheduling System [10]

Real resources lie on the bottom of the architecture in the Figure 2.1 each being managed by a **Local Resource Manager (LRM)**.

An LRM is responsible for:

- Processing low level resource requests by executing a job that satisfies that request.
- Enabling remote monitoring and management of jobs created in response to resource.
- Updating the information service with information about the current availability and capabilities of the resources that it manages.

An **LRM** serves as the interface between a Grid Scheduler and a local autonomous site being able to process low level resource requests. Grid Resource Allocation Management **(GRAM)** from Globus [23] is a good example of LRM.

2.1.1 Information Service (IS)

This component is responsible for maintaining the current state information of the resources such as CPU capacity, memory size, network bandwidth, software reliability and so on, and answering to queries for acquiring information about Grid Resources. To overcome the heterogeneous and dynamic nature of Grids, efficient information service plays a particularly important role in the Grid Scheduling System. An adequate scheduler must incorporate the current information of resources when making Scheduling decisions. Globus Meta Director Service (MDS) [23] and Network Weather Service (NWS) [32] are two best examples that provide information service. [10].

2.1.2 Analytical Benchmarking (AB)

This component provides a measure of how well a computational resource performs on a given job. The execution time of a given job varies with the capability of resources. Thus basically, the AB component provides a performance estimation of a given job on a specific computational resource.

2.1.3 Grid Scheduler (GS)

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.

2.1.4 Resource Allocation (RA)

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 Challenges in Grid Scheduling

Although Grids fall into the category of distributed parallel computing environments, they have a lot of unique characteristics, which make the Scheduling in Grids highly difficult. An adequate Grid Scheduling system should overcome these challenges to leverage the promising potential of Grid Systems, providing High-Performance services [10].

2.2.1 Resource Heterogeneity

A Grid has mainly two categories of resources: networks and computational resources. Heterogeneity exits in both of the two categories of resources. **First**, networks used to interconnect these resources may differ significantly in terms of their bandwidth and communicational protocols. A wide area Grid may have to utilize the best effort services provided by the internet. **Second**, computational resources are usually heterogeneous in that these resources may have 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 can not be considered uniformly. An adequate Scheduling system should address the heterogeneity and further leverage different computing powers of diverse resources.

2.2.2 Site Autonomy

Typically a Grid may compromise multiple administrative domains. Each domain shares a common security and management policy. Each domain usually authorizes a group of users to use the resources in the domain. Thus applications from non-authorized users should not be eligible to run on the resources in some specific domains.

Furthermore, a site is an autonomous computational entity. A shared site will result in many problems. It usually has its own Scheduling policy, which complicates the prediction of a job on the site. 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.

2.2.3 Local Priority

It's another important issue. Each site within the Grid has its own Scheduling policy. Certain classes of jobs have higher priority only on certain specific resources. 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.

2.2.4 Resource Non-Dedication

Because of non-dedication of resources, resource usage contention is a major issue. Competition may exist for both computational resources and interconnection networks. Due to the non-dedication of resources, a resource may join multiple Grids simultaneously. The workloads from both local users and other Grids share the resource concurrently. The underlying interconnection network is shared as well. One consequence of contention is that behavior and performance may vary over the time; Contention free at the guaranteed level Schedulers must be able to consider the effects of contention and predict the available resource capabilities.

2.2.5 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. An adequate Scheduling system should be able to handle a variety of applications. [8]

2.2.6 Dynamic Behavior

In Traditional parallel computing environments such as a cluster, the pool of resources is assumed to be fixed or stable. In a Grid Environment, dynamics exists in both the networks and computational resources. **First**, a network shared by many parties cannot provide guaranteed bandwidth. This is particularly true when wide-areas networks such as the internet are involved. **Second**, both the availability and capability of computational resources will exhibit dynamic behavior. On one hand new resources may join the Grid and on other hand, some resources may become unavailable due to problems such as network failure. The capability of resources may vary overtime due to the contention among many parties who share the resources. 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 leverage the new resources in the later Scheduling decision making. When a computational resource becomes unavailable resulting from an unexpected failure, mechanisms such as check pointing or rescheduling should be used to guarantee the reliability of Grid systems.

These challenges pose significant obstacles on the problem of designing an efficient and effective Scheduling system for Grid Environments.

2.3 Grid Scheduling Procedure

A user goes through three stages to schedule a job when it involves multiple sites. Phase one is Resource Discovery, in which the user makes a list of potential resources to use. Phase two involves gathering information about those resources and choosing a best set to use. In phase three the user runs the job.

2.3.1 Phase 1: Resource Discovery

Resource discovery involves the user selecting a set of resources to investigate in more detail; in phase two information gathering. At the beginning of this phase, the potential set of resources is empty set and at the end of this phase, the potential set of

resources is some set that has passed a minimal feasibility requirement. Most users do this in three steps namely:

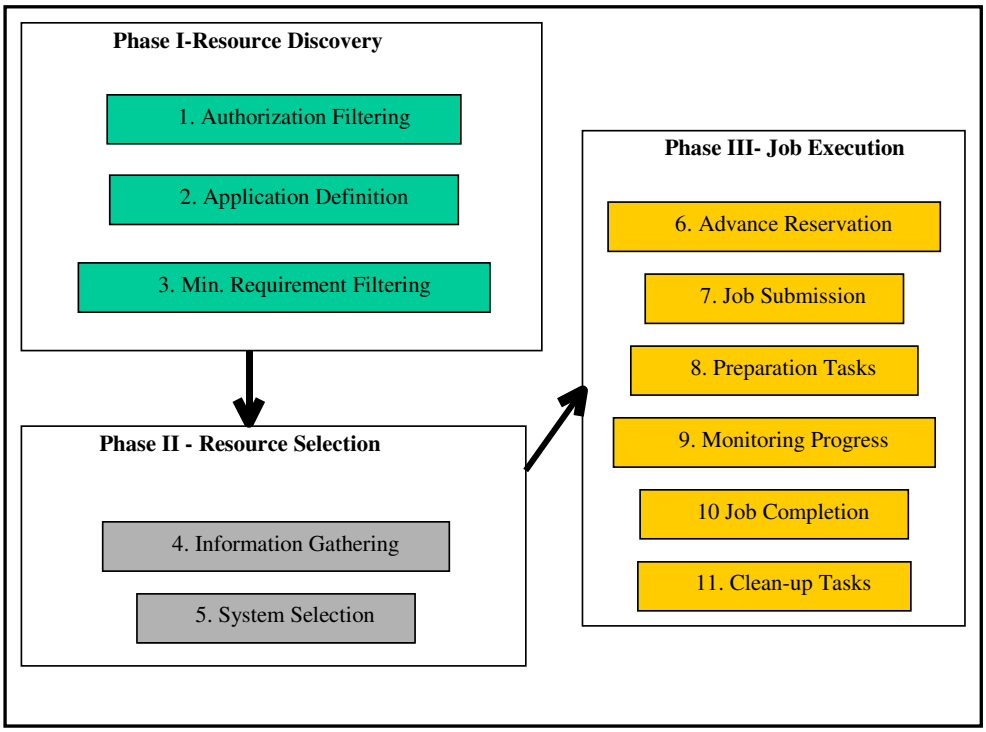


Figure 2.2: Grid Scheduling Process [11]

1. **Authorization filtering**

   It is generally assumed that a user will know which resources he 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 has access.

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. It may include static details such as operating system or hardware for which a binary of the code is available. Or that the code is best suited to a specific architecture. Dynamic details are also possible e.g. a minimum RAM requirement, connectivity needed. This may include any information about the job that should be specified to make sue that the job could be matched to a set of resources.

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 step is to filter out the resources that do not meet the minimal job requirements. The user generally does this step by going through the list of resources and eliminating the ones that do not meet the job requirements as much as they are known. It could also be combined with the gathering more detailed information about each resource.

2.3.2 Phase 2: System Selection

Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job. This is generally done in two steps:

1. **Gathering Information (QUERY)**

   In order to make the best possible resource match, a user needs to gather dynamic information about the resources in question. Depending on the application and resource in question, different information may be needed. Take for instance the simple case of finding the best single resource for a job to run on. A user might want to know the load on the various machine(s) and queue lengths if the machine has queues. In addition, physical characteristics and software requirements play a role, is the disk big enough for the data etc. then there are location/connectivity issues is the machine close enough to the data store. All of these issues are multiplied in the case of multiple resources. Making an advance reservation may or may not be a part of this step.

2. **Select the system(s) to run on**

   Given the information gathered by the previous step, a decision of which resource (or set of resources) should the user submit a job is made in this step. This can be done in variety of ways. Note that this does not address the situation of speculative execution, where a job is submitted to multiple resources and when one begins to run the other submissions is cancelled.

2.3.3 Phase 3: Run the Job

The third phase of Scheduling is running a job. This involves a number of steps:

1. **(Optimal) Make an Advance Reservation**

   It may be the case that to make the best use of a given system, part or all of the resources will have to be reserved in advance. Depending on the resource, this can be easy or hard to do, may be done with mechanical means as opposed to human means, and the reservations may or may not expire with or without cost.

2. **Submit Job to Resources**

   Once resources are chosen the application must be submitted to resources. This may be easy as running a single command or as complicated as running a series of scripts, and may or may not include setup or staging.

3. **Preparation Tasks**

   The preparation stage may involve setup, 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 America's National Aeronautics and Space Agency (NASA) [33] was considered unsuccessful because it did not address the need to stage files automatically.

4. **Monitor Progress**

   Depending on the application and its running time, users may monitor the progress of their application.

5. **Find out if Job is done**

   When the job is finished, the user needs to be notified.

6. **Completion Tasks**

   After a job is run, the user may need to retrieve files from that resource in order to do analysis on the results, break down the environment and remove temporary settings etc.

2.4 Grid Scheduling Taxonomy

There have been a number of efforts attempting to design Scheduling systems for Grids, each having its unique features. A comprehensive set of taxonomies is discussed in this section. The taxonomies are defined by many aspects of Grid Scheduling systems [10].

2.4.1 Knowledge of Application

- **Application Level Scheduling**

  The application level Scheduling scheme makes use of knowledge of applications as much as possible. Such kind of Scheduling results in custom schedulers for each application attempting to maximize application performance, measured as runtime or speedup, with little regard to overall system performance. The complexity of application level Scheduling is the order of the applications considered. *APPLeE* [31] is a Scheduling system that uses the application level Scheduling scheme.

- **Resource Level Scheduling**

  Resource level Scheduling does not use much knowledge of Grid applications. In this scheme, applications neither specify resource requirements nor provide application characteristics. Generally, a scavenging Grid aiming at leveraging the idle computing power will use this Scheduling scheme. Condor [22] is an example, which uses resource level Scheduling.

2.4.2 Inter-Job Dependency

Given an application, the constituent jobs may either be dependent or independent. The mapping algorithms for a set of independent jobs differ significantly from those for a set of dependent jobs. An application with a set of jobs is usually represented by a DAG. The mapping algorithms for a set of dependent jobs are more complicated.

2.4.3 Information Service

The scheduler determines the state information of all the resources in a Grid system through the information service before making a Scheduling decision. Different Scheduling systems construct quite different structures to provide information services.

- **Centralized**

  Under a centralized scheme, there exists a single centralized entity that maintains the state info. The centralized entity traverses every resource to get the most up to date state information periodically and keeps the information in its storage, waiting for queries issued by the schedulers. A centralized scheme is not scalable because it introduces the risk of single point of failure. Furthermore, a centralized entity may become the performance bottleneck

when the entity cannot afford a large number of resources and possible queries.

- **Decentralized**

  Within this scheme, every resource is responsible for maintaining its current state information locally, and answering queries from different clients. A decentralized scheme may not be efficient due to the amplified quantity of queries. However the decentralized scheme is more reliable because the risk of single point of failure is removed through distributing the responsibility evenly to every resource. The large overhead should be carefully considered. Example of decentralized scheme is NWS [32].

- **Hybrid**

  Using the hybrid scheme resources are categorized into several groups. Within each group centralized scheme is applied. Thus each group has representative entity; which is in charge of the information of all resources in its group. Over the groups, the decentralized scheme applies. This scheme is the most practical method for real wide area Grids.

2.4.4 Scheduler Organization

The Grid Scheduler organization can be organized into three categories: centralized, decentralized & hierarchical.

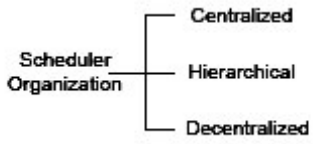


Figure 2.3 Scheduler Organization Types [21]

- **Centralized**

  In the centralized scheme, all user applications are sent to the centralized scheduler. There is a queue in the centralized scheduler for holding all the pending applications. When user application is submitted to the scheduler, it may not be scheduled at once. Instead it will be put in the queue, waiting for scheduling and resource allocation. Typically, each site neither maintains its queue nor performs any Scheduling decisions. A site receives jobs from the

scheduler and executes them. The centralized scheme is not very scalable with increasing number of resources.

The central scheduler may prove to be a bottleneck in some situation.

- **Decentralized**

  Decentralized scheme distributes the responsibility of Scheduling to every. Each site in the Grid acts as both a scheduler and a computational resource. User applications submitted to the local Grid scheduler where the applications originate. The local scheduler is responsible for Scheduling its local applications, thus it possibly maintains a local queue to hold its own pending applications. Meanwhile, it should be able to respond to other schedulers' requests by acknowledging or denying it.

- **Hierarchical**

  In the hierarchical scheme, different levels of schedulers share the Scheduling process. The higher-level schedulers manage larger sets of resources and lower level schedulers manage smaller sets of resources. A higher-level scheduler has no direct control of a resource if there is one lower-level scheduler between the higher–level scheduler and the resource. A higher-level scheduler can only consider the capability of the set of resources managed by a lower-level scheduler as a whole entity, and utilizes the capability through invoking the lower-level scheduler. Compared with the centralized Scheduling, hierarchical Scheduling addresses the scalability and the problem of single point failure issue. Nevertheless, it also retains some of the advantages of the centralized scheme.

2.4.5 Rescheduling

After an application was scheduled, the performance of the application may not approach the desired performance due to the dynamic nature of the resources. It may be profitable to reschedule the application during execution to maintain good performance. At the minimum, an adequate Grid scheduler should acknowledge the resource failure and resend lost work to a live computational resource. In summary, rescheduling is done to guarantee the job's completion and performance goal's achievement.

2.5 Grid Scheduling Approaches

The Scheduling policy determines how the Scheduling should be performed. The performance goal defined in the Scheduling policy plays a particularly important role in a Grid Scheduling system. According to the different performance goals, the Scheduling systems can be classified into three categories:

2.5.1 Application Centric

Scheduling systems that fall in the application centric category try to favor the 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 greedy mapping algorithm, which allocates the application to the resources that are likely to produce the best performance without considering the rest of pending applications.

2.5.2 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.

2.5.3 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 [25] is a Scheduling system which exploits idea of economy mechanism.

2.6 Taxonomy of Grid Scheduling Algorithms

It is well known that the complexity of a general Scheduling problem is NP-Complete

[20].The Scheduling problem becomes more challenging because of some unique characteristics belonging to Grid Computing as discussed earlier. The Grid Scheduling Algorithms can be classified as:

- **Local vs. Global**

  At the highest level, a distinction is drawn between local and global Scheduling. 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.

- **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 the 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. This is useful when it is impossible to determine the execution time, direction of branches and number of iterations in a loop as well as in the case where jobs arrive in a real-time mode. Both static and dynamic Scheduling are widely adopted in Grid Computing.

- **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 the NP-Complete nature of Scheduling Algorithms and the 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.

- **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. In the case where a metric is available for evaluating a solution, this technique can be used to decrease the time

taken to find an acceptable schedule. The factors which determine whether this approach is worthy of pursuit include

- Availability of a function to evaluate a solution.
- The time required to evaluate 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.

The other branch in the suboptimal category is called *heuristic*. This branch represents the class of algorithms which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It also represents the solutions to the Scheduling problem which cannot give optimal answers but only 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. Not restricted by formal assumptions, heuristic algorithms are more adaptive to the Grid scenarios.

- **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.

- **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.

2.7 Summary of Current Approaches to Grid Scheduling

The Following table summarizes properties of widely used the Grid Resource Management Systems with emphasis on their scheduling attributes [13]:

Table2.1 Summary of Current Approaches in Grid Scheduling [13]

System	Grid Type	Resources	Scheduling Approach
Condor [22]	Computational Flat	Extensible schema model, hybrid namespace, no QoS, network directory store, centralized queries discovery.	Centralized Scheduler
Globus [23]	Multiple Hierarchical	Extensible schema model, hierarchical namespace, soft QoS, network directory store, distributed queries discovery.	Decentralized Scheduler infrastructure, scheduling provided by external schedulers like Nimrod/G [25].
NetSolve [24]	Computational Hierarchical	Extensible schema model, hierarchical namespace, soft QoS, distributed queries discovery	Decentralized Scheduler, fixed application oriented policy
Nimrod/G [25]	High-Throughput Hierarchical	Extensible schema model, hierarchical namespace, relational network directory data store, soft QoS, distributed queries discovery	Hierarchical decentralized Scheduler, predictive pricing models, fixed application oriented policy

2.8 Challenges in Evaluation of Scheduling Algorithms

Besides the characteristics of Grid Environment which make the performance evaluation of Scheduling algorithms tedious; the following factors also make the task difficult:

- Performance prediction is difficult because end to end internet performance itself is extremely hard to analyze and predict.
- End to end performance observed on internet exhibits great diversity and thus different algorithms work more effectively for different topologies and also for different time periods on same topology.

2.8.1 Motivation for use of Grid Simulation Tools

Be it Application Developer, Tool Developer or Grid Developer all of them need to test and verify their applications, tools & services respectively for fulfillment of intended design goals before the end product or logic is put in Real-Time Grid Computing Environment. The motivation for using simulation instead of directly using the Grid test-bed, especially in analyzing models and algorithms can be drawn from the following factors [14]:

- Setting up a Grid test-bed is expensive, resource intensive, and time consuming. Even if one is set up, it is mostly limited to some local area environments.
- Using a real test-bed with real jobs is time consuming as well. Hours of real job time can be simulated in seconds provided the simulation is having sufficient processor power.
- 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 that would otherwise require involvement of a large number of active users, which is very hard to coordinate and build at a large-scale research environment for investigation purposes.
- Simulation allows analyzing existing as well as new economic models and scheduling algorithms.

2.8.2 Taxonomy of Grid Simulation Tools

A large variety of Simulation Tools have been developed and implemented for Parallel and Distributed systems, including Grids [15].

Table 2.2 Categorization of Simulation Tools based on their simulated systems [15]

Category	Tool	Organization	Key Features
Parallel Systems	SimOS [26]	Standford University, U.S.A	Simulates a complete multiprocessor system and studies all various aspects including hardware architecture, OS & application programs.
Distributed Systems	SimJava [27]	University of Ediburgh,U.K	Provides a core set of foundation classes for simulating discrete events, distributed hardware systems, communication protocols and computer architectures.
Grid Scheduling Systems	Bricks [28]	Tokoyo Institute of Technology, Japan	Provides simulation for resource allocation & scheduling algorithms for multiple clients and servers as in a Grid environment.
	GridSim [17]	University of Melbourne, Australia	Supports simulation of space-based and time-based, large-scale resources in the Grids. Simulates economy-based resource Scheduling systems in Grids.
	MicroGrid [29]	University of California, U.S.A	Runs emulations by executing actual application code on the virtual Globus Grid.
	SimGrid [30]	University of California, U.S.A	Simulates a single or multiple scheduling entities and timeshared systems operating in Grids. Simulates distributed Grid applications for resource Scheduling.

As this thesis work focuses on Grid Resource Scheduling we dig deeper into the features and design elements of the Simulation tools available for Grid Scheduling [15]:

Table 2.3 Feature Analysis of Grid Resource Scheduling Simulation Tools [15]

Design Element	**Bricks**	**GridSim**	**MicroGrid**	**SimGrid**
Simulated Systems	Grid, Resource Scheduling Systems	Grid, Resource Scheduling Systems	Grid, Resource Scheduling Systems	Grid, Resource Scheduling Systems
Usage	Simulator	Simulator	Emulator	Simulator
Simulation Engine	Serial, Event driven	Multithreaded, Event driven	Parallel, Event driven	Serial, trace Driven
Simulation	Static, discrete, deterministic	Static, discrete, deterministic	Dynamic, continuous, deterministic	Static, discrete, deterministic
Modeling Framework	Entity-based, Event-based	Entity-based, Event-Based	Entity-based, Event-Based	Entity-based, Event-Based
Programming Framework	Object-Oriented	Object-Oriented	Structured	Structured
Design Environment	Language	Library	Language	Library
User Interface	Non-Visual	Form	Non-Visual	Non-Visual
System Support	Statistics generation	Code generation, Statistics generation	N/A	N/A

Chapter 3

The Proposed Scheduling Algorithm

3.1 Introduction

Computational Grids that couple geographically distributed resources such as PCs, workstations, supercomputers, processors, clusters, and scientific instruments, have emerged as a next generation computing platform for solving large-scale problems in science, engineering, and commerce. However Resource Management and Scheduling in these heterogeneous & dynamic environments continues to be a tedious task. In Grid Computing Environment, 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.
In general, task Scheduling is an NP-Complete problem [20].

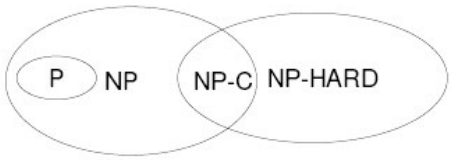


Figure 3.1: Relationship between NP-Hard, NP-Complete & NP Problems [20]

At the crux of the Grid Scheduling System lies the Scheduling Algorithm. As mentioned in the Chapter 2 a large category of Scheduling Algorithms have been proposed so far. In the current state of the art, Optimal polynomial-time algorithms exist for Scheduling tree-structure task graphs with uniform computational cost on a bounded number of machines, Scheduling a task graph with uniform computation cost to machines and Scheduling an interval ordered task graph with uniform task weights to a bounded number of machines [7]. In all these cases the communication cost among the tasks of the parallel application are assumed to be zero. All these shortcomings must be removed.

3.2 Design Goals

This thesis aims at design and development of a Compile Time Task Scheduling Algorithm for Grid Computing Environment which overcomes the shortcomings of the current state of the art in the context.

Our main Design Goals are:

- The Grid Scheduling Algorithm should not make any simplifying assumptions about the architecture of parallel programs submitted to the Grid.
- The Grid Scheduling Algorithm should not make any simplifying assumptions about the architecture of target computing platforms.
- The Grid Scheduling Algorithm should not assume availability of infinite number of resources in the Grid.
- The Grid Scheduling Algorithm should take into account the communication costs between Grid Nodes.

The algorithm is to be fully implemented on the GridSim [17] Toolkit 4.0 and all the statistical results are to be examined using JCharts [18] API to verify and validate intended design goals.

3.3 Proposed Grid Application Model

We propose a model for the application to be used for the Compile-Time Scheduling in Grid Computing environment. The application is represented by a Directed Acyclic Graph (DAG) [20] G (V, W, E, C).

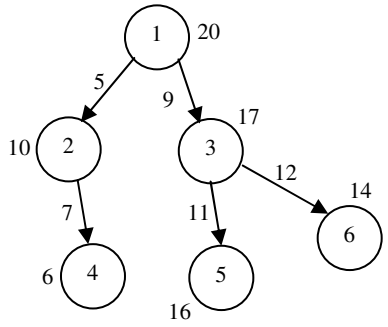


Figure 3.2: Directed Acyclic Graph (DAG)

The Figure 3.2 above shows a sample DAG with six nodes. The numerical value beside each node denotes the computation cost of the task the node denotes and the

numerical value on each directed edge indicated communication cost between the pair of nodes connected.

The terminology employed in the proposed algorithm is as follows:

V is the set of v nodes and each node $v_i \in$ V represents an application task, which is a sequence of instructions that must be executed serially on the same machine.

W is a $v*p$ computation cost matrix in which w_{ij} gives the estimated time to execute task v_i on machine p_j.

E is the set of communication edges. The directed edge e_{ij} joins nodes v_i and v_j where node v_i is called the parent node and node v_j is called the child node. This also means that v_j cannot start until v_i finishes and sends its data to v_j.

C is the communication cost and e_{ij} has a communication cost $c_{ij} \in$ C.

A task without any parent is called an entry task and a task without any child is called an exit task. If there is more than one exit (entry) task they may be connected to a zero-cost pseudo exit (entry) task with zero cost edges which do not affect the schedule.

We assume a heterogeneous Grid Computing Environment which has a finite number of computing resources in a fully connected topology. 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. An application consists of finite number of tasks v_i.

From now onwards in this thesis text the proposed Algorithm will be referred to as SCH Algorithm.

3.4 The Proposed Algorithm: SCH

The average earliest start time EST of v_i can be computed recursively by traversing the DAG downward starting from entry task v_{entry}.

$$EST(v_{entry}) = 0$$

The average latest start time LST of v_i can be computed by traversing the DAG upward starting from the exit task v_{exit}.

$$LST(v_{exit}) = 0$$

Average execution cost can be defined as

$$\overline{w_i} = \sum_{j=1}^{p} w_{ij} / p$$

The main objective of the SCH Algorithm is to determine the assignment of the tasks of a given application to a given machine set P such that Schedule length (makespan) is minimized while satisfying all the precedence constraints.

The algorithm is divided into two phases. The first phase is the Selection and second is the Resource Allocation phase.

3.4.1 Phase One: Selection

The basic idea is that delaying a critical task will delay all the children of the critical task. A critical task can be defined as the task that has zero difference between its EST and LST values. Mathematically for task v_{ct}

$$| EST_{vct} - LST_{vct} | = 0$$

Based on this fact the algorithm tries to add critical tasks, starting from entry task and ending on exit task, to the list (L) as early as possible. To achieve this goal the unlisted parents of each critical task should be added to L before adding the critical itself. Selecting the critical tasks to be added to L as early as possible divides the application DAG into a set of unlisted parent trees. The root of each parent tree is a critical task. To add a critical parent tree to L ; we will check the critical task's parents in depth first, starting from the critical task in question and add a parent to L if it does not have any unlisted parents. The heuristic applied at this stage is minimum value of LST for performance reasons.

The pseudo code for the listing phase is given next; it uses pseudo functions of stack push and pop. The push function inserts a value onto stack and pop function removes the last pushed valued from the stack.

```
BEGIN
 traverse DAG downward & compute EST for each task
 traverse DAG upward & compute LST for each task
 push critical tasks on the stack T in reverse order of their LST value
 while T is not empty
 do
        if there are unlisted parents of top(T)
```

then

push the parent with the smallest LST value on T

else

pop the top(T) and enqueue it to list L

END

3.4.2 Phase Two: Resource Allocation

The input to this phase of the SCH Algorithm is the list L generated in the previous phase. Instead of examining one task at each step, the mechanism in the machine assignment phase checks one task and one parent at a time which does not increase the complexity.

In the pseudo code below the values of *i* and *j* vary from one to the number of tasks in the application and the number of machines respectively.

BEGIN

Loop for i and j

Select the machine p_j *that minimizes the finish time of the current selected task* v_i *from the list L.*

Examine the idle time left on p_j *by* v_i *for replicating the parent of* v_i

Confirm the selection if it reduces the starting time of the v_i *otherwise loop again.*

END

3.4.3 Complexity Analysis of the Algorithm

The complexity is expressed in terms of the number of tasks *v*, number of edges of application DAG (*e)* and the number of machines *p*:

- The complexity of computing EST and LST values of all the tasks O (*e*+*v*).
- The complexity of listing phase of algorithm is O (*e*+*v*).
- The complexity of machine assignment phase is $O(pv^2)$.

The total algorithmic complexity is {O (*e*+*v*) + $O(pv^2)$ }. Taking into account the principles of graph theory for a DAG the *e* is $O(v^2)$, the total complexity of the proposed algorithm becomes $O(pv^2)$.

3.5 Performance Evaluation Metrics

The SCH Algorithm tries to optimize the following performance parameters.

- **Makespan**

  We define makespan as the maximum time taken for the completion of all the tasks in a given application. The time taken to complete the tasks on the critical path is lower bound on the value of makespan.

- **Schedule Length Ratio (SLR)**

  SLR is defined as the makespan divided by the sum of the minimum computation costs of the tasks on the critical path. The SLR of DAG cannot be less than one, because in the fraction generated, the denominator is the lower bound.

Chapter 4

Implementation Details and Experimental Setup

Throughout the implementation of the SCH Algorithm emphasis has been on the use of Open Source Technologies and Toolkits. The SCH Algorithm has been implemented in JAVA programming language on top of GridSim [17] Toolkit 4.0 and JCharts [18] API has been used for Graph based Statistical analysis, aiding in better visualization of trends. The steps followed for the implementation are described below in detail with appropriate screenshots and relevant API usage details.

4.1 Installation of Pre-requisites and Necessary Components

4.1.1 Installation of Eclipse IDE

Eclipse is an Open Source community whose projects are focused on providing a vendor-neutral Open development platform and application frameworks for building software. The Eclipse Foundation is a not-for-profit corporation formed to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an Open Source community and an ecosystem of complementary products, capabilities, and services. The Eclipse [19] IDE can be downloaded from the homepage of Eclipse Foundation [19]

The IDE can be installed easily following the on screen messages. The next step is to configure the Eclipse IDE for the implementation of the algorithm. The steps to be followed are:

- Make sure JDK 1.5 is installed on the system.
- Change the runtime environment to JRE 1.5 from that of default Eclipse.
- Set the Java Virtual Machine Parameters as "–Xmx350M" to increase the heap size for the Java Virtual Machine as the GridSim execution is taxing on the underlying system.

4.1.2 Installation of JCharts API

The JCharts is completely open source JAVA based API for Graph generation. The current implementation of the proposed algorithm uses JCharts [18] API version 0.7.5. It can be downloaded from JCharts Homepage [18].

The precise steps to be followed on Windows System for installation are as:

- Unzip the downloaded archive to the location intended to be used for installation.
- Update the System Path variable to include the path to the JCharts-0.7.5.jar file, e.g. If the GridSim has been unzipped to the D:\JCharts-0.7.5 location then we add to the System Path variable
- D:\JCharts-0.7.5\JCharts-0.7.5.jar
- The JCharts API is ready to be used in JAVA applications on the Windows system.

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

4.1.3 Installation of GridSim Toolkit

In the current implementation of the SCH Algorithm we have used GridSim Toolkit version 4.0. This open source JAVA based toolkit can be downloaded from the homepage of Grid-Bus Society [21].

The GridSim [17] toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains distributed computing systems such as clusters and Grids.

Salient Features of GridSim [17]

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space-or time-shared mode.
- Resource capability can be specified in the form of MIPS (Million Instructions Per Second) or as per SPEC (Standard Performance Evaluation Corporation) .
- 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.
- 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. 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.

GridSim Architecture

GridSim has employed a layered and modular architecture for Grid simulation to leverage existing technologies and manage them as separate components. A multi-layer architecture and abstraction for the development of GridSim platform and its applications is shown in Figure 4.1.

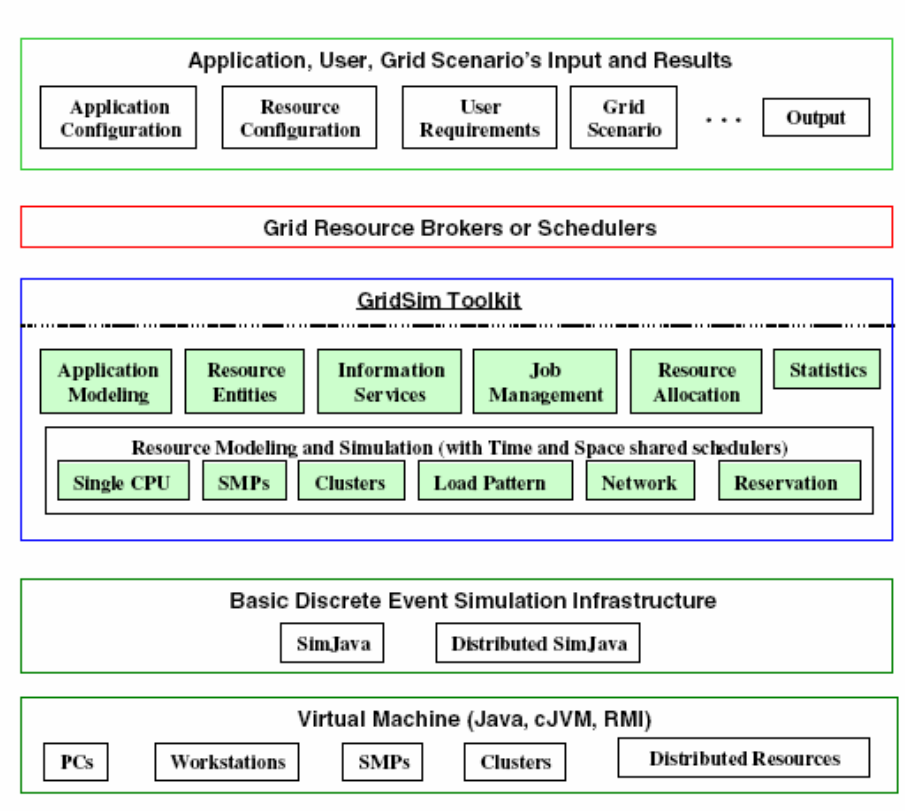


Figure 4.1 Architecture for GridSim platform and components [17]

The **first layer** is concerned with the scalable Java interface and the runtime

machinery, called JVM (Java Virtual Machine). The **second layer** is concerned with a basic discrete-event infrastructure built using the interfaces provided by the first layer. One of the popular discrete-event infrastructure implementations available in Java is SimJava [16]. 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 using the discrete-event services offered by the lower-level infrastructure. 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.

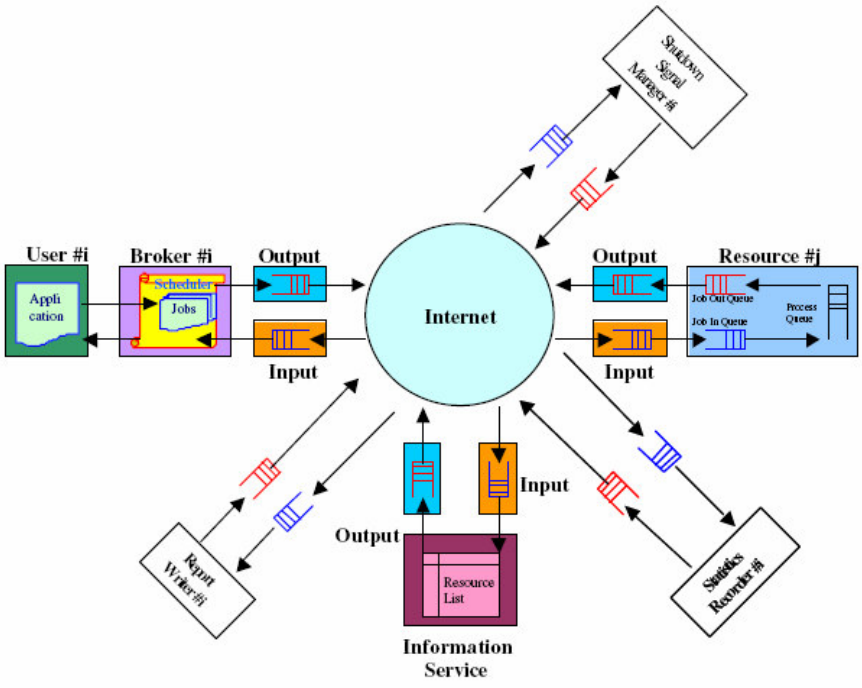


Figure 4.2 A Flow Diagram for GridSim based Simulations [17]

Steps to be followed on Windows System for GridSim 4.0 installation are:

- Unzip the downloaded archive to the location intended to be used for installation
- 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-4.0 location then we add to the System Path variable

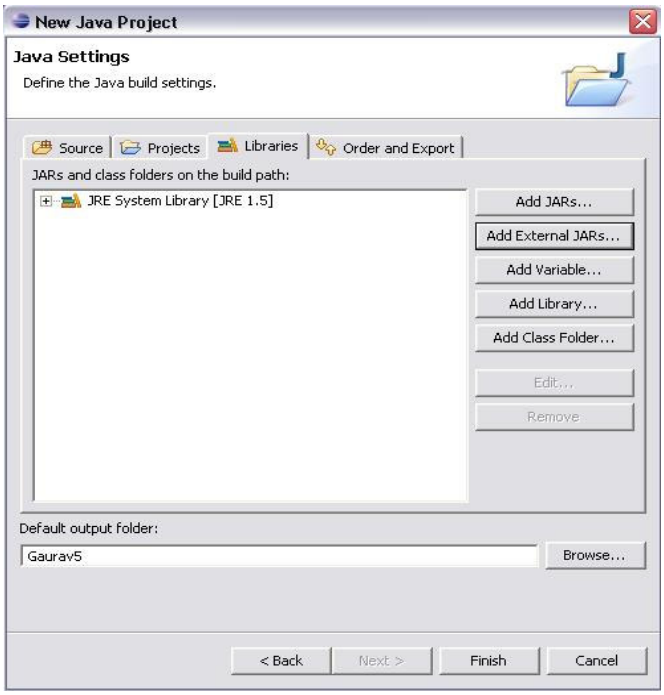
- D:\gridsimtoolkit-4.0\jars\gridsim.jar and
- D:\gridsimtoolkit-4.0\jars\simjava2.jar
- The GridSim API is ready to be used in JAVA applications on the Windows system.

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.

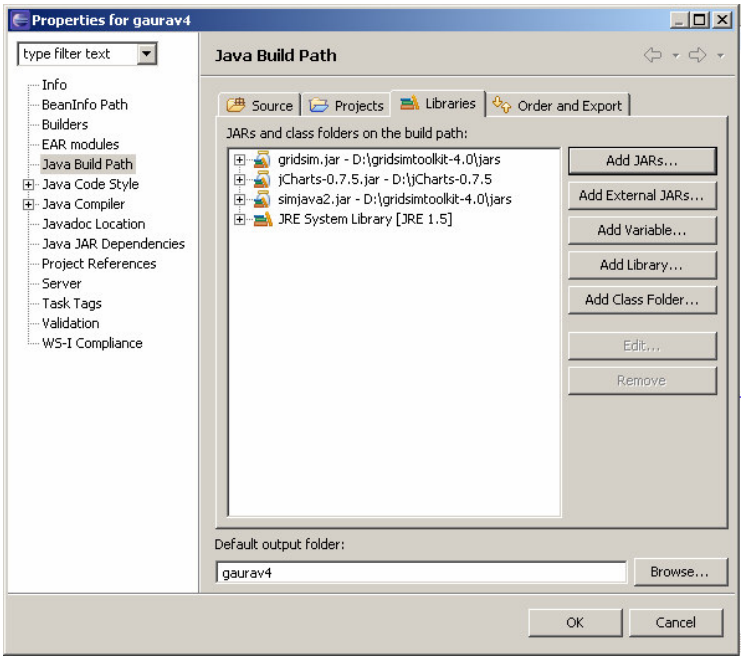
4.1.4 Importing GridSim & JCharts Jars in Eclipse

The SCH Algorithm has to be simulated on top of GridSim [17] API as a JAVA application built in Eclipse [19] IDE. The steps to be followed are as:

- Create a new Java Project in Eclipse
- Create a new Package under this project.
- To set the properties of the project go to the properties tab and then to build path option. In the configure build path option add the external jars stored on the local machine i.e. gridsim.jar, simjava2.jar and JCharts-0.7.5.jar are to be browsed and added accordingly as shown below.



Screenshot 4.1: Before addition of external project dependencies



Screenshot 4.2: After addition of external project dependencies

4.2 Implementation of SCH Algorithm in GridSim

To use the GridSim API for the implementation and simulation of the SCH Algorithm we need to follow below mentioned sequential steps; which when together coded in JAVA programming language simulate a heterogeneous Grid Computing Environment. All the type names referenced here are part of GridSim API.

4.2.1 Creating a Grid Resource in GridSim

A Grid Resource simulated in GridSim contains one or more machines. Similarly a machine contains one or more PEs (Processing Elements or CPUs). Below are the steps to be followed in JAVA code to create a Grid Resource.

1. Create an object of type MachineList to store one or more Machines

   MachineList mList = new MachineList();

2. A Machine contains one or more PEs/ CPUs. So we create an object of type PEList to store these PEs before creating a Machine.

   PEList peList1 = new PEList();

3. Create PEs and add these into the object of PEList created in step 2. We have to specify the unique ID of the PE as first parameter and its MIPS (Millions of Instruction Per Second) rating as second parameter.

   peList1.add(new PE(0, 377));

4. Create a Machine with its unique ID and the PEList asscociated with it.

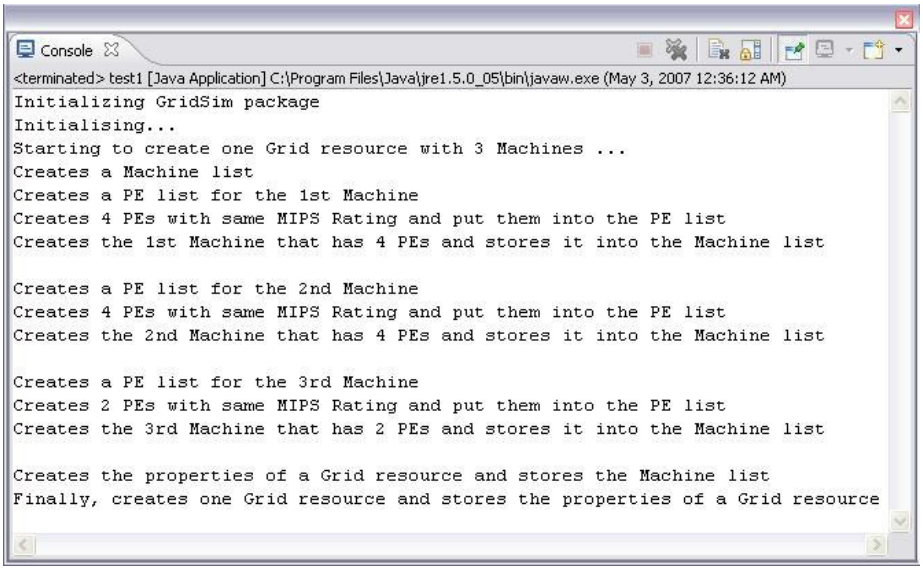
   mList.add(new Machine(0, peList1));

5. Repeat the steps from step 2 to step 4 to create additional number of machines.
6. Create a Resource Characteristics object which will store the properties of a Grid Resource, its architecture, Operating System, List of Machines, Allocation Policy which can be Time Shared or Space Shared, Time Zone and Cost of the Resource in terms of $ per PE Time Unit.

   ResourceCharacteristics resConfig = new ResourceCharacteristics(
   arch, os, mList, ResourceCharacteristics.TIME_SHARED,
   time_zone, cost);

7. In the final step we create an object of Grid Resource specifying its name, communication speed, peak load, off-peak load, holiday load and list of holidays along with the Resource Characteristics object which was created in the step 6.

   GridResource gridRes= new GridResource(name, baud_rate, seed,
   resConfig, peakLoad, offPeakLoad, holidayLoad, Weekends,Holidays);



Screenshot 4.3: Grid Resource creation in GridSim

4.2.2 Creating Grid User(s) in GridSim

The steps to create Grid User(s) in GridSim are straightforward, but each User must have a unique ID. In the next step when the Gridlets (jobs) are created at that time we need to specify the User ID to which the Gridlet (job) belongs, therefore

Grid user creation precedes Gridlet (job) creation. The steps to create a Grid User are as follows:

1. Create an object of type ResourceUserList.

ResourceUserList userList = new ResourceUserList();

2. Add to this list Grid users specifying a unique ID, first user has to have ID equal to 0. Just keep on adding Grid users to this list if we wish to create to more of them.

userList.add(0);

Screenshot 4.4: Grid Resource creation in GridSim

4.2.3 Creating Gridlets (jobs) in GridSim

In the terminology of GridSim, a job which can run sequentially and independently on a Grid Resource is called a **Gridlet**. After the creation of Grid Resource we create Gridlets in the GridSim which can then be submitted to the latter. We need to specify the length of Gridlet, its output file size, its input file size, its unique ID for simulation. Gridlet creation can be done in two modes first is the manual option and second is with the use GridSim Random functions to take care of the statistical needs of highly unpredictable Grid environment simulation. The steps to create Gridlet(s) in both modes are:

Manual Mode:

1. Create an object of type GridletList

GridletList list = new GridletList();

2. Create an object of type Gridlet specifying its unique id, length, input file, output file size in types integer, double, long integer and long integer respectively.

Gridlet gridlet1 = new Gridlet(id, length, file_size, output_size);

3. Add the object created in step 2 to the Gridlet list created in step 1.

list.add(gridlet1);

4. To create more Gridlets repeat from step 1.

Random Mode: Use of GridSimRandom and GridSimStandardPE Classes.

1. Create an object of type Random.
2. Create an object of type GridletList

   GridletList list = new GridletList();

3. Set the MIPS rating in the GridSimStandardPE.set Rating method.
4. Set the minimum and maximum ranges in the double types. These values can't be same at same time.
5. Set the length, file size and output size using the random functions in a newly created object of type Gridlet as:

   *length = GridSimStandardPE.toMIs(random.nextDouble()*output_size);*

   file_size = (long) GridSimRandom.real(100, min_range, max_range, random.nextDouble());

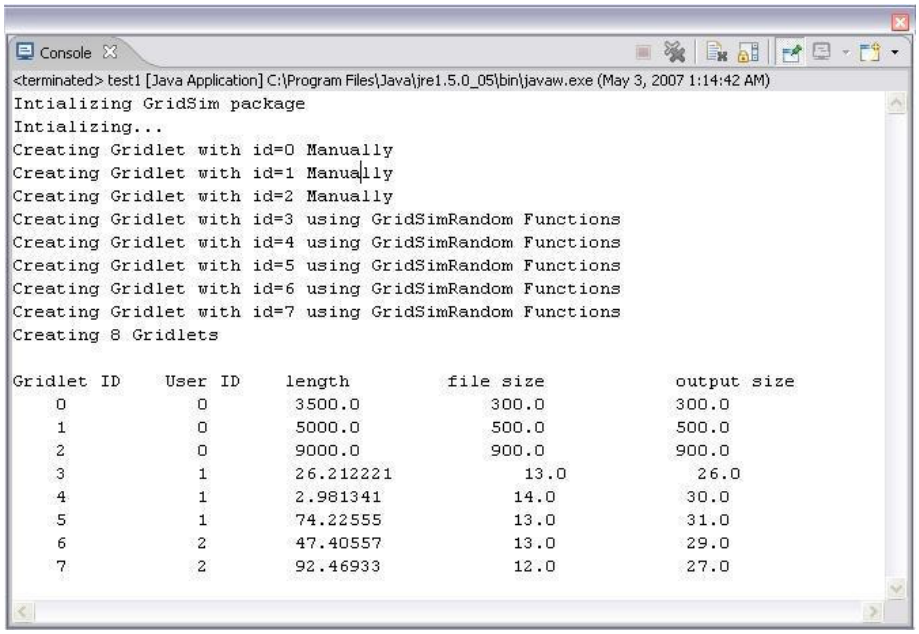
   output_size = (long) GridSimRandom.real(250, min_range, max_range, random.nextDouble());

   Gridlet gridlet = new Gridlet(id, length, file_size, output_size);

6. Add the Gridlet object created in the step 5 to the Gridletlist type object created in step 2.

   list.add(gridlet);

7. Repeat steps 5 and 6 if we intend to create more Gridlets.



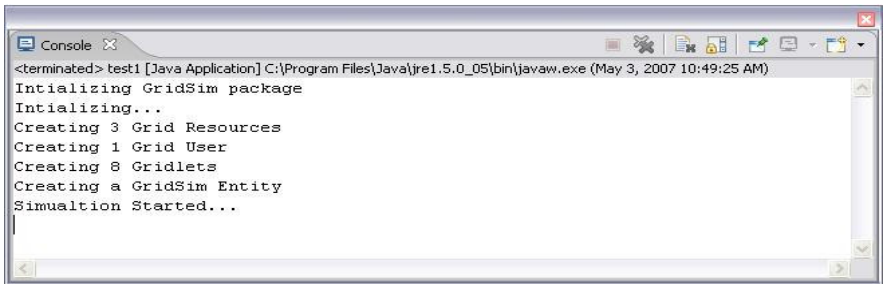
Screenshot 4.5: Gridlet creation in GridSim

4.2.4 Creating Grid Entities in GridSim to start the actual simulation

Till now we have successfully created Grid Resource(s), Grid User(s) and Gridlet(s) in GridSim. To actually start the simulation in GridSim we need to create GridSim entity without this we will get a Null pointer Exception at runtime In this we use the method GridSiminit() which takes as parameters the

1. Calendar Type Instance.
2. Trace Boolean Flag with which we can control the tracing of GridSim Events.
3. List of files or processing names to be excluded from any statistical measures and report name.

```
String[] exclude_from_file = { "" };
String[] exclude_from_processing = { "" };
String report_name = null;
GridSim.init(num_user, calendar, trace_flag, exclude_from_file,
    exclude_from_processing, report_name);
```



Screenshot 4.6: Grid Entity creation in GridSim

4.2.5 Submission of Gridlet(s) to Grid Resource(s) in GridSim

The steps to be followed to submit the Gridlet(s) to Grid Resource(s) are:

1. Since GridSim package uses multi-threaded environment the request for list of Grid Resources might arrive earlier before one or more Grid Resource Entities manage to register themselves to GridInformationService (GIS) entity. Therefore we wait for some micro seconds using the Hold method of GridSim Class.
2. The GIS will return only the Grid Resource IDs.
3. Using the ID obtained in step 2 retrieve Resource Characteristics from Resource Entity.

```
super.send(resourceID, GridSimTags.SCHEDULE_NOW,
```

GridSimTags.RESOURCE_CHARACTERISTICS, this.ID_);

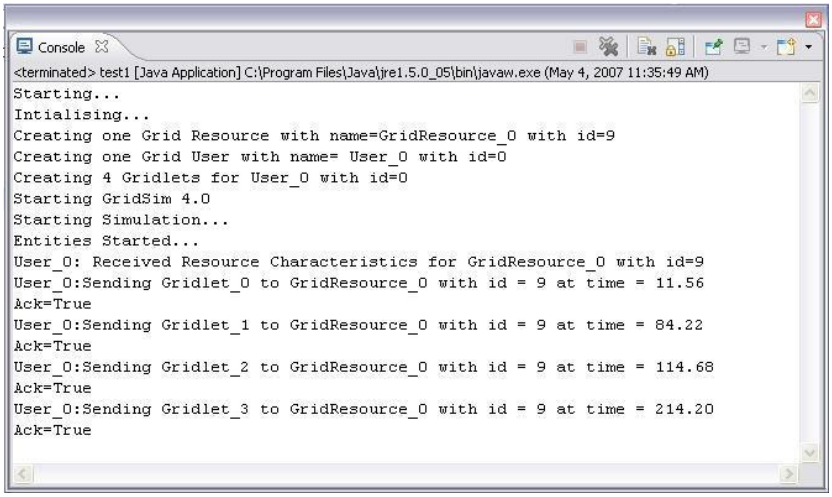
resChar = (ResourceCharacteristics) super.receiveEventObject();

resourceName = resChar.getResourceName();

4. From the Gridlet list take a Gridlet at a time and then submit to the Grid Resource Entity with the Grid Resource ID to which we intend to submit. The last parameter in the method call below signifies that we need an acknowledgement for Gridlet successful submission.

   supergridletSubmit(gridlet, resourceID[id], 0.0, true);

5. Record the statistics in the GridSim report.
6. Repeat the steps 5 and 6 according to the number of Grid Resources and Gridlets.
7. Create a list of type GridletList to keep a record of Gridlets submitted.



Screenshot 4.7: Gridlet Submission in GridSim

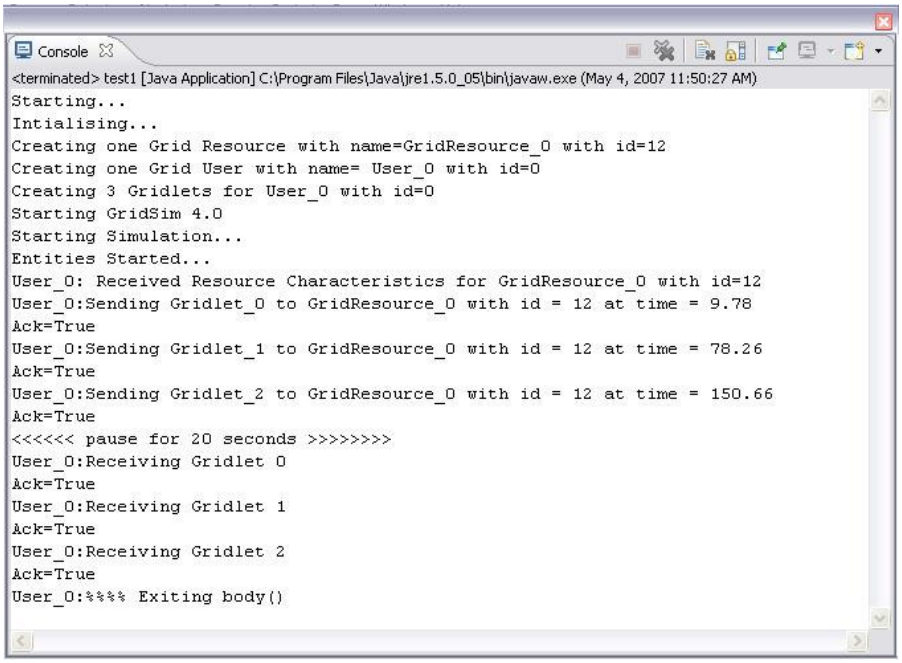
4.2.6 Retrieving back of Gridlet(s) from Grid Resource(s) in GridSim

The steps to receive back finished Gridlet(s) from Grid Resource(s) are as follows:

1. Create a empty List of type GridletList named as Received list
2. For each Gridlet submitted call the method

   gridlet = (Gridlet) super.receiveEventObject();

3. Add this Gridlet to the received List created in step 1
4. Repeat the steps 2 and 3 for all the Gridlets submitted.
5. Then compare the received list with the submitted list.

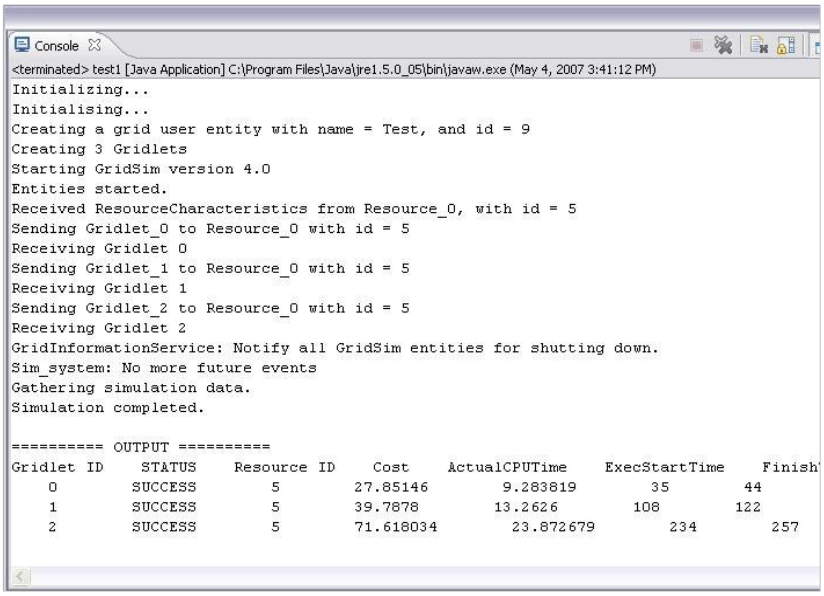


Screenshot 4.8: Gridlet Retrieval in GridSim

4.2.7 Retrieving Statistical Data from Gridlets in GridSim

For each Gridlet in the list of received Gridlets we can use the following methods with appropriate return type.

- Gridlet.getProcessingCost()
- Gridlet.getActualCPUTime()
- Gridlet.getLength()
- Gridlet.getCostPerSec()
- Gridlet.getWaitTime()
- Gridlet.getResourceID()
- Gridlet.getStatus()
- Gridlet.getExecStartTime()
- Gridlet.getExecFinishTime()

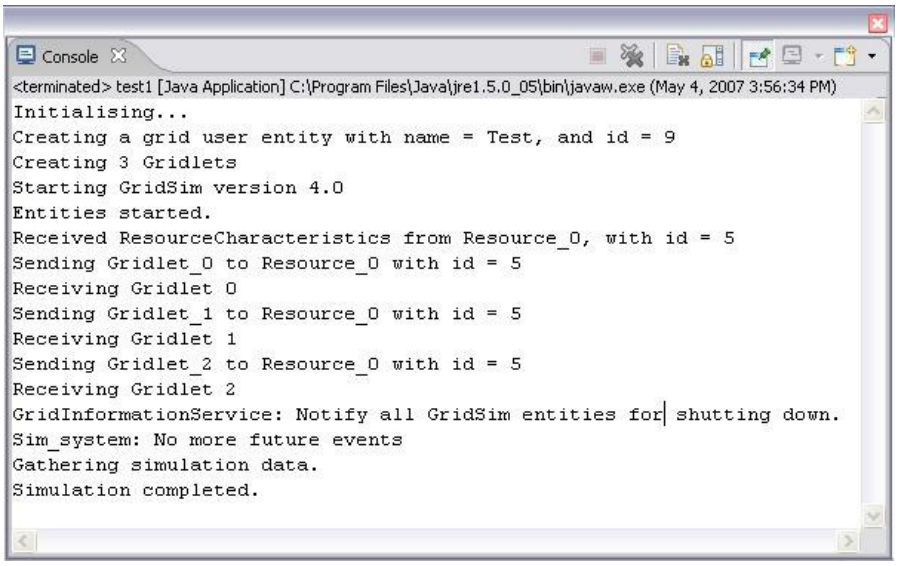


Screenshot 4.9: Gridlet Info Retrieval in GridSim

4.2.8 Shutting Down Simulation in GridSim

To shut down the simulation in GridSim three methods are used, they internally clear all the buffers, release all the files and reports written.

- GridSim.shutdownGridStatisticsEntity();
- GridSim.shutdownUserEntity();
- GridSim.terminateIOEntities();



Screenshot 4.10: Shutting Down GridSim

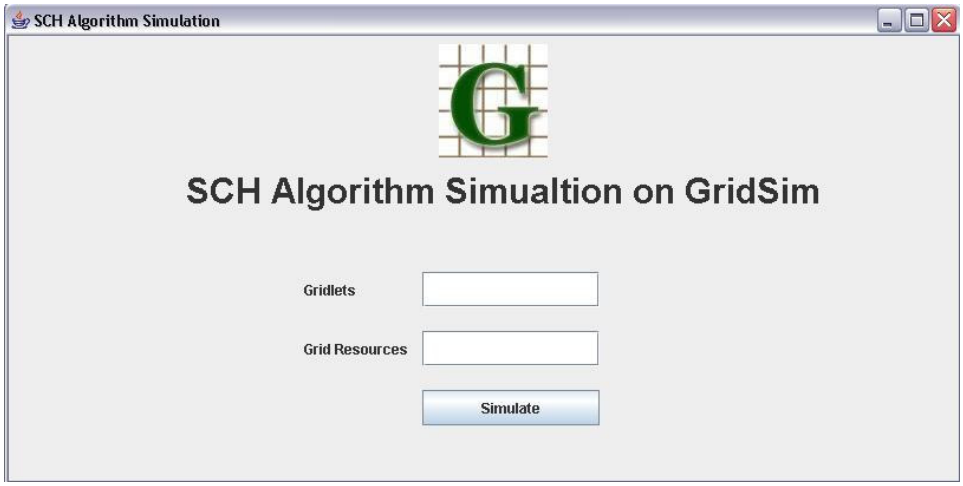
Chapter 5

Experimental Results

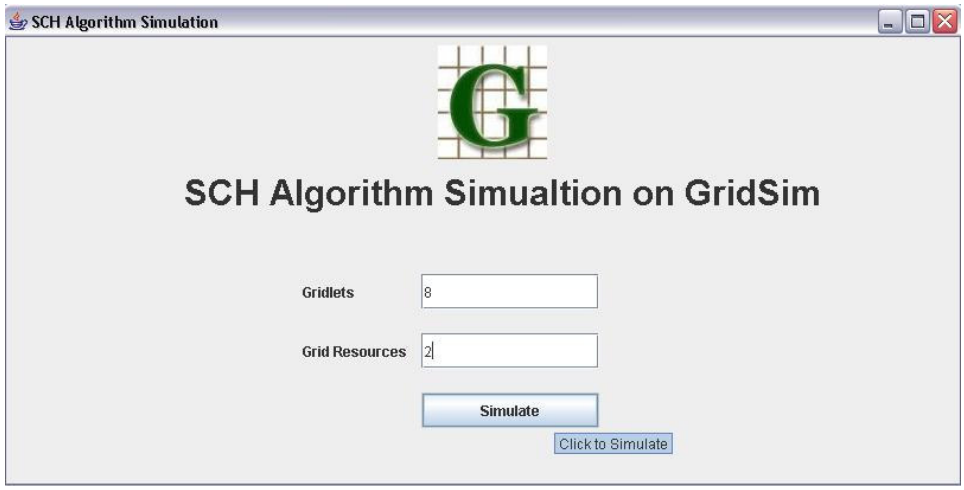
By following the steps mentioned in the chapter 4 the complete SCH algorithm is implemented in JAVA and run on top of GridSim version 4.0 Toolkit. In this chapter we explain and discuss the results obtained in the statistical light. For a better visualization of the trends JCharts version 0.7.5 API has been used.

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.



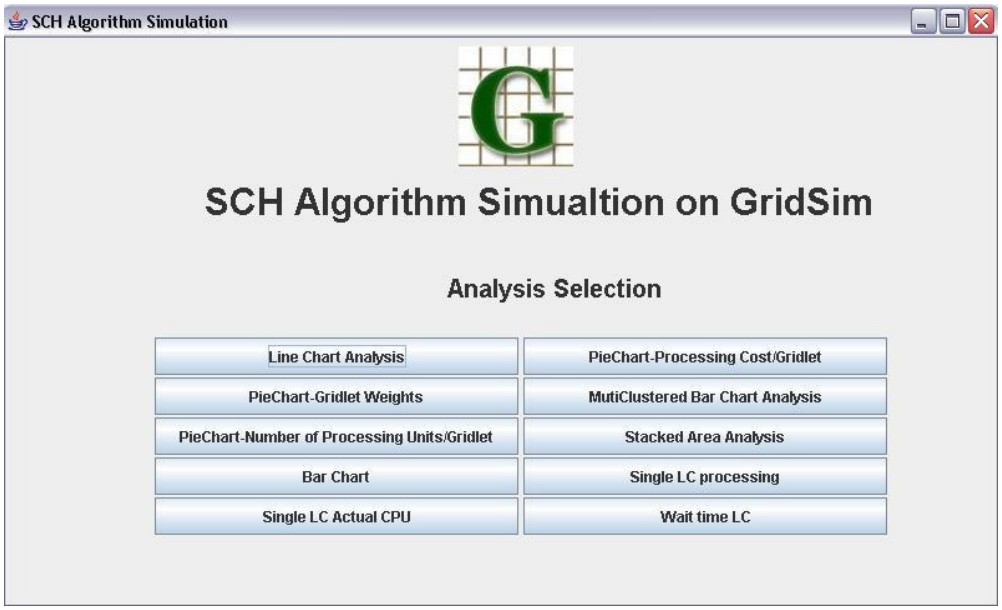
Screenshot 5.1: Opening Window before Data Entry by user



Screenshot 5.2: Opening Window after Data Entry by user

5.2 Analysis Selection

In the second step the user is prompted to choose for the analysis selection type, in the meanwhile the simulation has been completed in the background as explained in chapter 4.



Screenshot 5.3: Analysis Selection Window

5.3 The Output

The parameters of our interest for graphical analysis are:

- Processing Cost/Gridlet
- Gridlet Weight i.e. Size
- Cost/Sec for each Gridlet
- Actual CPU Time taken by each Gridlet
- Latest Execution Start Time of each Gridlet
- Earliest Execution Finish Time of each Gridlet

Starting with a relatively smaller set of data for easy trend visualization i.e. number of Gridlets to be simulated is eight and numbers of Grid Resources are two (**Test Input-I**) as shown in the Screenshot 5.1. The following are the results we obtain Graphically Figure 5.1 to Figure 5.8.

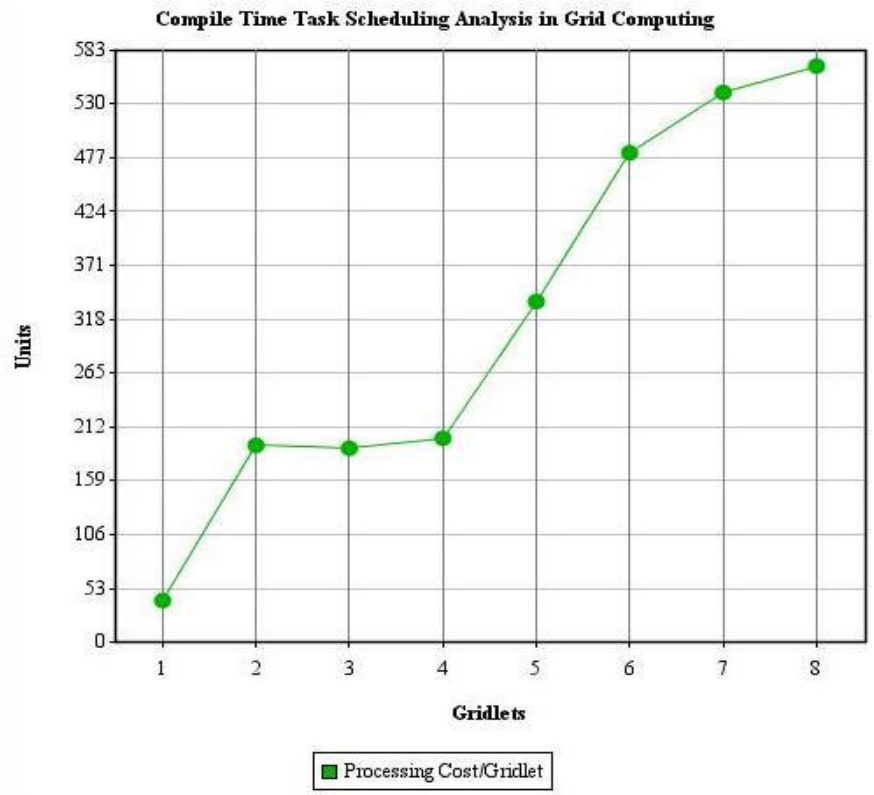


Figure 5.1: Line Chart showing Processing Cost/Gridlet for Test Input-I

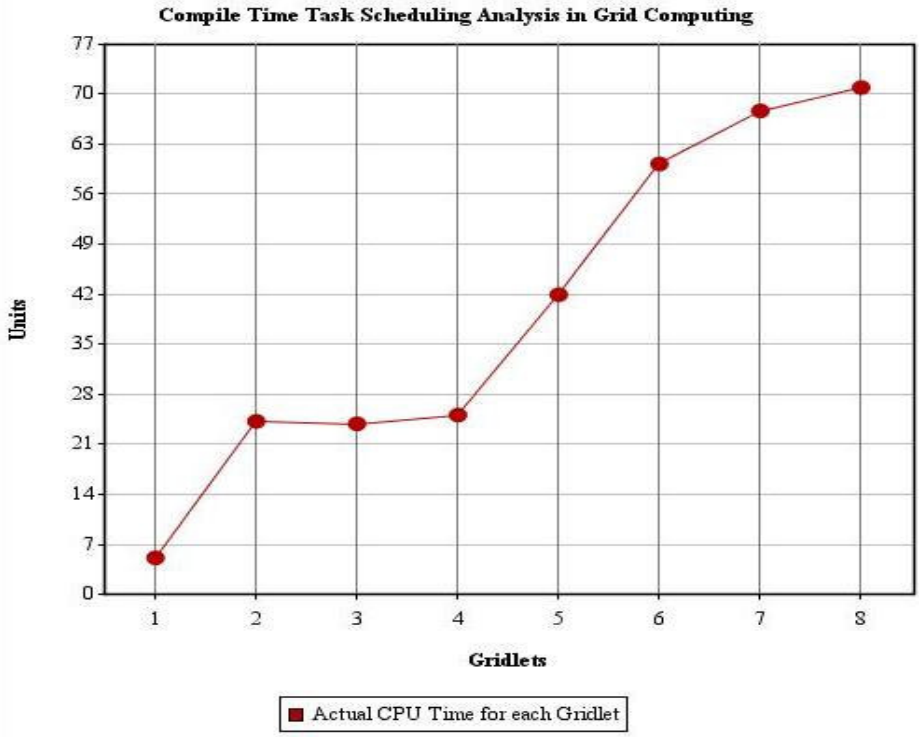


Figure 5.2: Line Chart showing Actual CPU Time for each Gridlet for Test Input-I

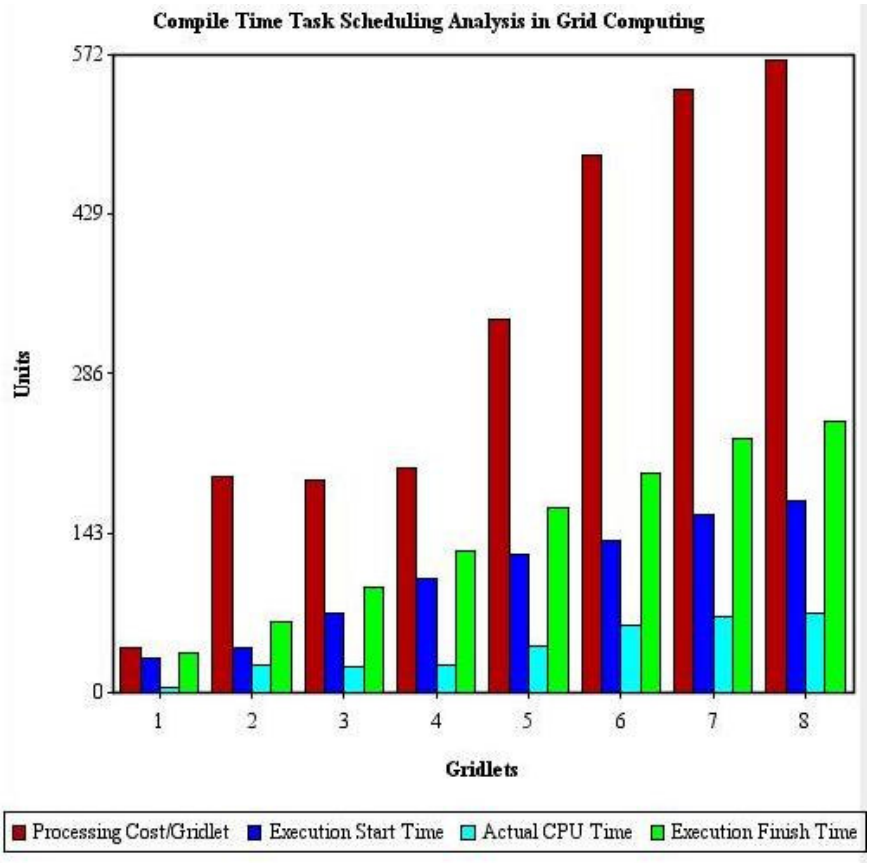


Figure 5.3: MutiClustered Bar Chart Comparing Gridlet Statistics for Test Input-I

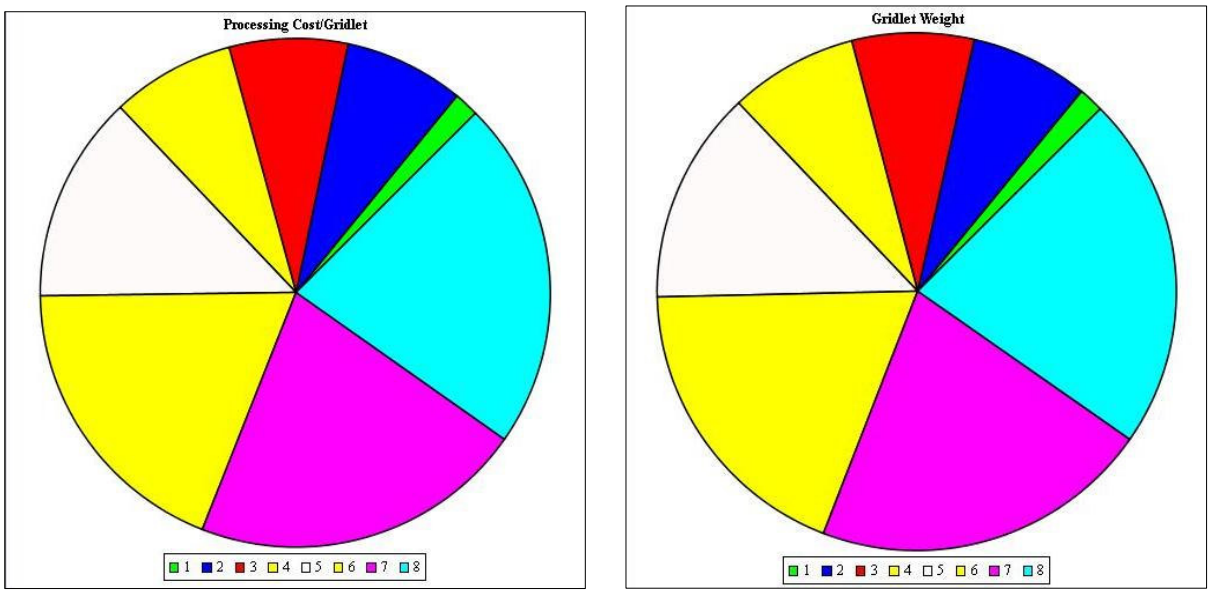


Figure 5.4: Pie Chart Comparison of Processing Cost/Gridlet with respective Gridlet Weighs for Test Input-I

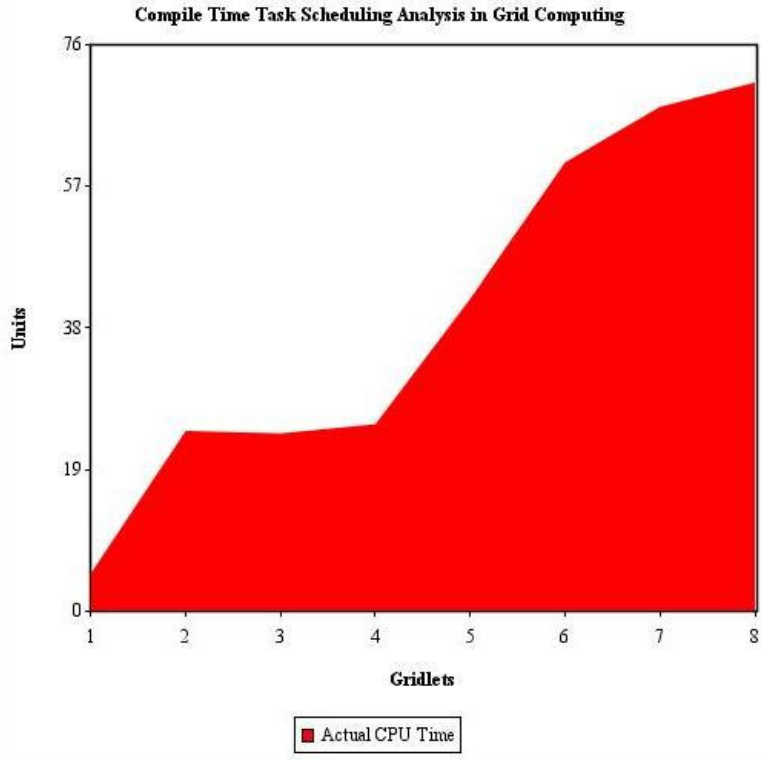


Figure 5.5: Stacked Area Curve for Actual CPU Time where the Area under the curve gives total execution time for Test Input-I

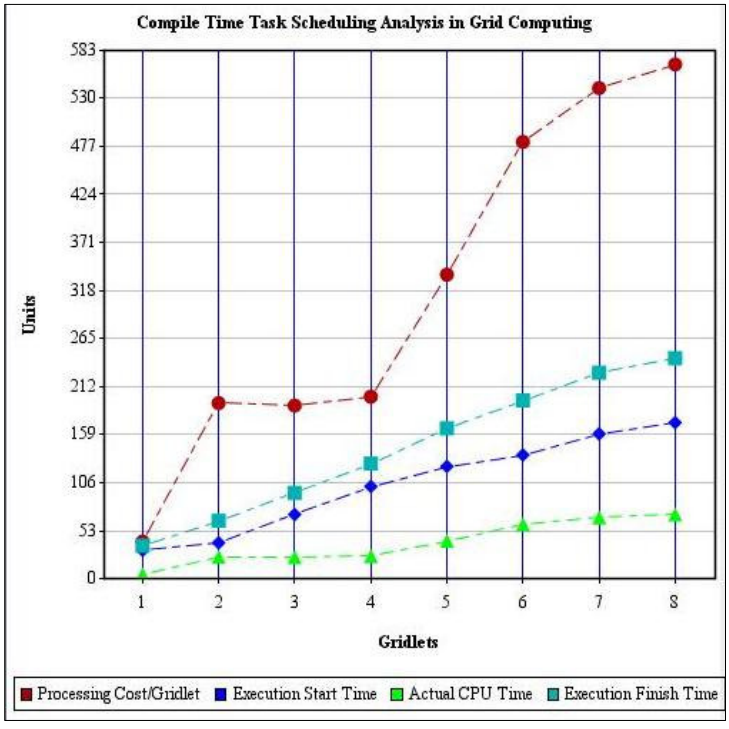


Figure 5.6: MutiLine Chart Comparing Gridlet Statistics for Test Input-I

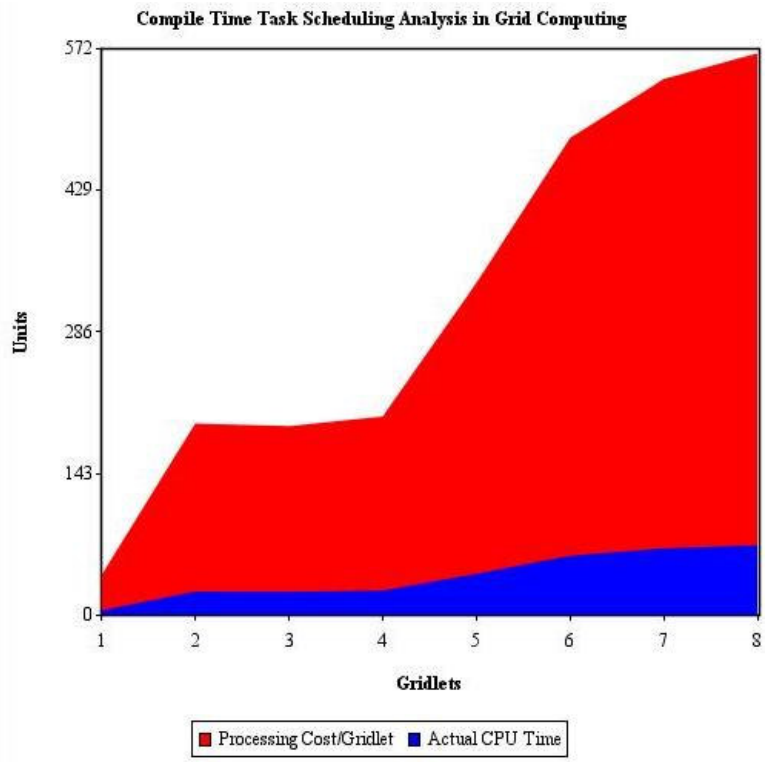


Figure 5.7: Stacked Area Curve of Actual CPU Time versus the Processing Cost/Gridlet for Test Input-I

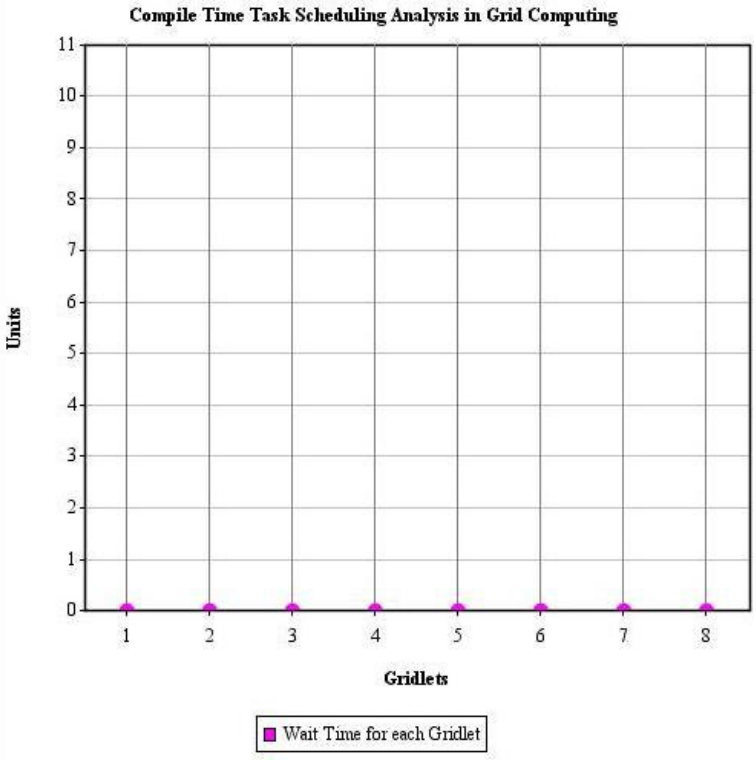


Figure 5.8: Line Chart showing Zero Waiting Time for each Gridlet in Test Input -I

Now we increase the input data slightly to 50 Gridlets and 5 Grid Resources (**Test Input-II**), Visualizing Trends Graphically again from Figure 5.9 to Figure 5.11.

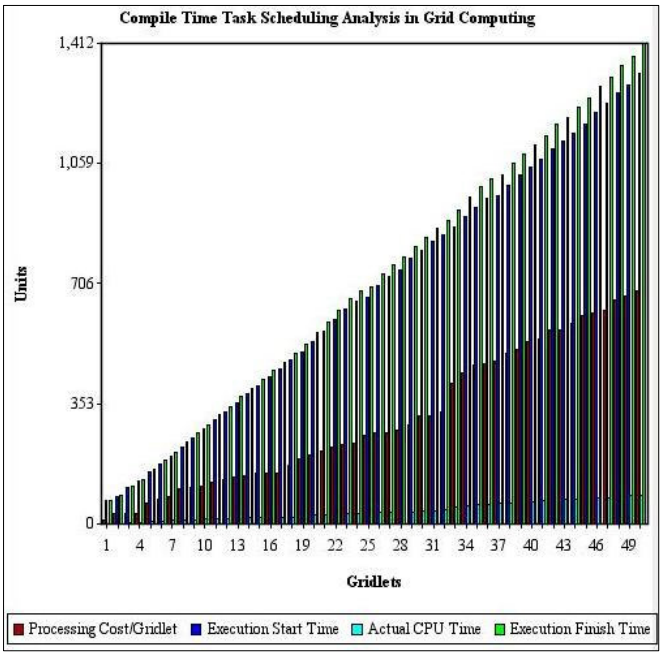


Figure 5.9: MutiClustered Bar Chart Comparing Gridlet Statistics for Test Input-II

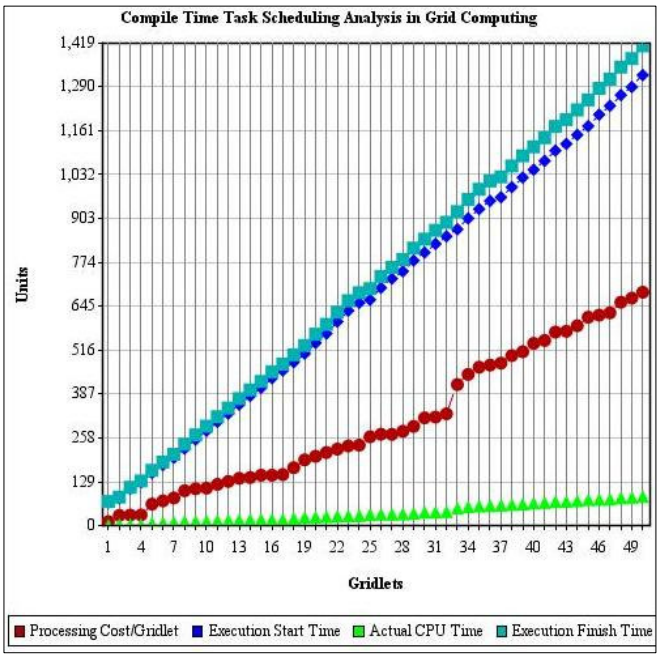


Figure 5.10: MutiLine Chart Comparing Gridlet Statistics for Test Input-II

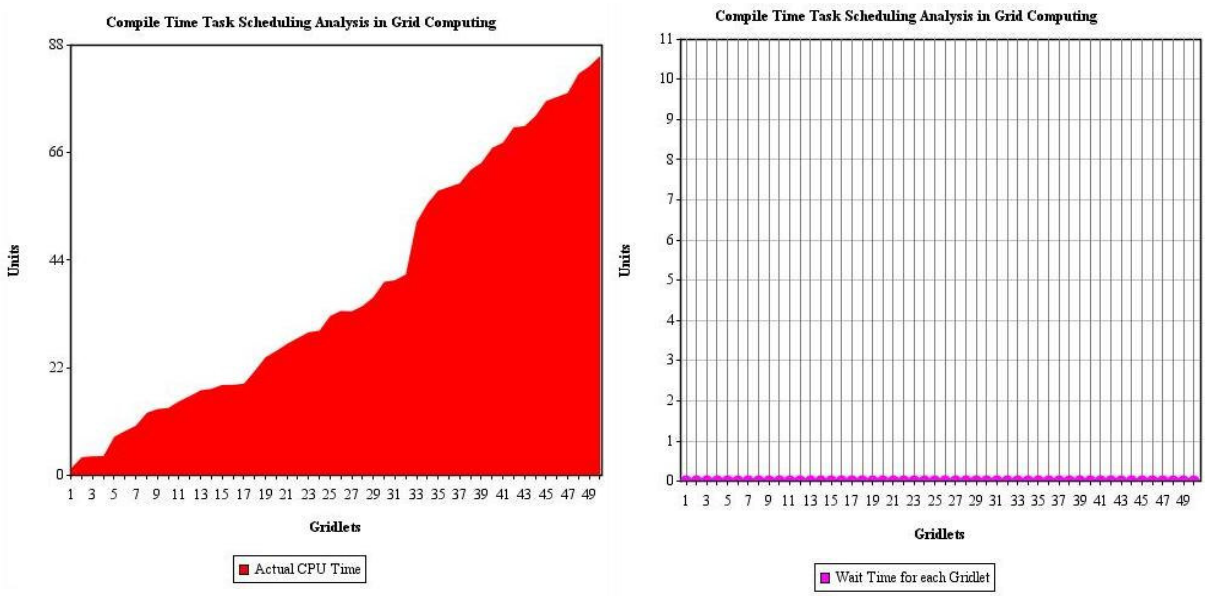


Figure 5.11: Stack Area Curve on the Left shows total time for application execution for Test Input-II and On Right the Wait Time Graph for each Gridlet

On increasing the number of Gridlets, still the waiting for each Gridlet is negligible. We increase the Gridlet count to 100 with 10 Grid Resources (**Test Input-III**) we get the following trends Figure 5.12 and Figure 5.13.

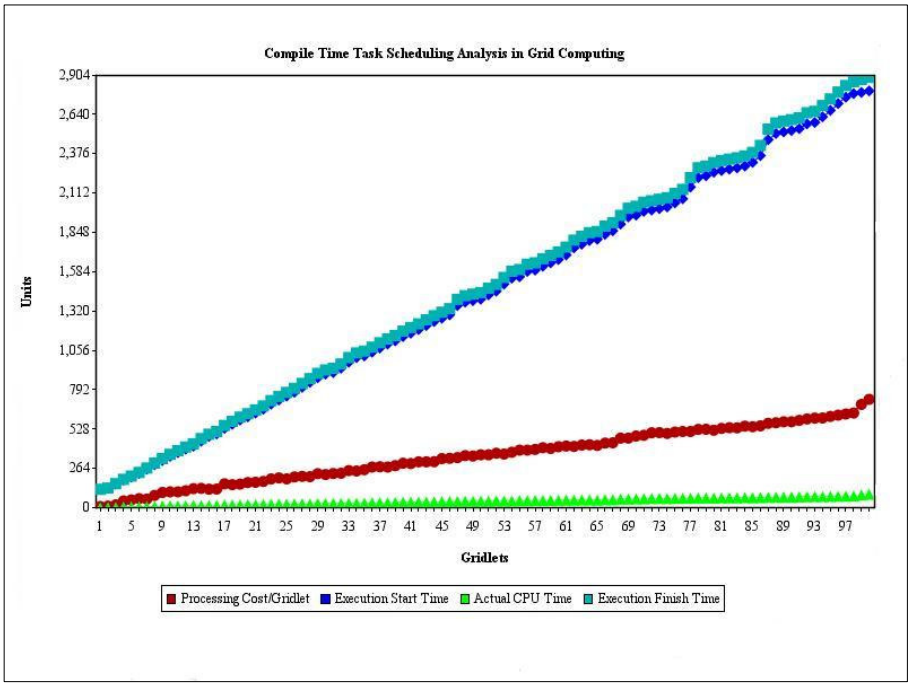


Figure 5.12: MutiLine Chart Comparing Gridlet Statistics for Test Input-III

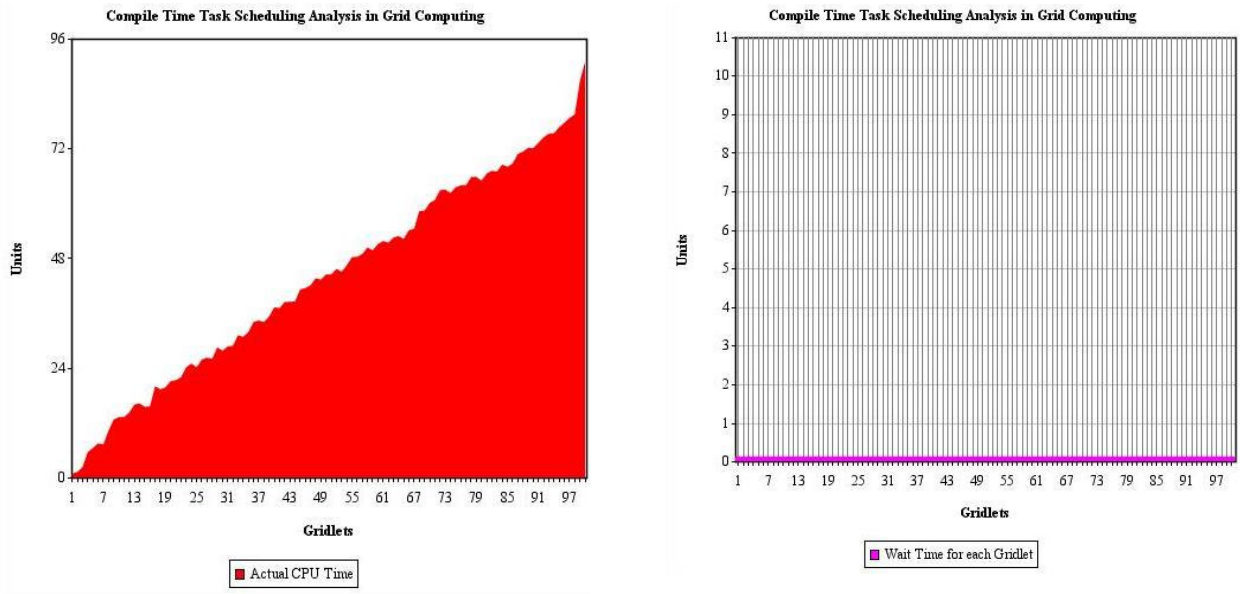


Figure 5.13: Stack Area Curve on the Left shows total time for application execution for Test Input-III and On Right the Wait Time Graph for each Gridlet.

5.4 Inferences Drawn

- The zero value of Wait Time shows that Makespan of the Schedule is optimized, Figures 5.8, 5.11 and 5.13.
- Even after increasing the number of Gridlets to 50 and then 100 with corresponding increase in the number of Grid Resources to 5 and 10 respectively, Wait Time for Gridlet submission remains negligible.
- The Gridlet Processing time is proportional to Gridlet Weight, Figure 5.1, 5.2, 5.10 and 5.12.
- Successful implementation of SCH Algorithm on top of GridSim 4.0 Toolkit and use of JCharts-0.7.5 API for Graphical data visualization.

Chapter 6

Conclusion and Future Scope

The work presented in this Thesis provides an insight into the world of Grid Computing, the current state of the art in Grid Scheduling Algorithms along with pros and cons of each approach.

The Thesis work focuses primarily on Compile Time Task Scheduling in Grid Computing Environment and tries to propose a Compile Time Task Scheduling Algorithm, SCH Algorithm based on Directed Acyclic Graphs. The SCH Algorithm is then implemented and successfully simulated on top of open source GridSim version 4.0 Toolkit together with JCharts 0.7.5 Graph Visualization API in a test environment.

The main highlights of the work are:

- The proposed SCH Algorithm makes no assumptions about the architecture of parallel programs submitted to the Grid.
- The proposed SCH Algorithm makes no assumptions about the architecture of target computing platforms.
- The proposed SCH Algorithm assumes availability of finite number of resources in the Grid.
- The proposed SCH Algorithm takes into account the communication costs between Grid Nodes.
- By applying the heuristics of SCH Algorithm job submission time to the Grid i.e. Wait Time can be reduced to negligible values.
- The proposed SCH Algorithm is successfully implemented in JAVA programming Language for complete Platform Independence.
- The proposed SCH Algorithm is simulated on top of GridSim 4.0 Toolkit.
- The proposed SCH Algorithm simulation uses JCharts 0.7.5 API for Statistical Data Analysis Visualization using Graphs.
- Use of Open Source Tools & Technologies throughout the design, development and simulation of SCH Algorithm.

Future Scope

- The performance of the SCH 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 & dedicated Deadlock Prediction and Avoidance Mechanism can be designed.
- The current proposed SCH Algorithm takes into consideration Resource Information which is made available at Compile Time, for a Heterogeneous & Dynamic Environment like that of Grids, we need to account for Dynamic Information about Resources.
- 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] Kleinrock, L. *MEMO on Grid Computing*, University of California, Los Angeles, 1969. <http://www.lk.cs.ucla.edu/LK/Bib/REPORT/press.html>

[2] Foster, I. and Kesselman, C. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 677 pp, 1999. <http://www.globus.org/research/papers/chapter2.pdf>

[3] Foster, I., Kesselman, C. and Tuecke, S. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of High Performance Computing Applications, Vol. 15, No. 3, pp. 200-222, 2001. <http://www.globus.org/research/papers/anatomy.pdf>

[4] *IBM Grid Solutions*: <http://www-1.ibm.com/grid/solutions/index.shtml>

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

[6] Franccois Grey, Matti Heikkurinen, Rosy Mondardini, *Grid Cafe: A Palace for Everybody to Learn About Grid*, <http://Gridcafe.web.cern.ch/Gridcafe/Gridwork/architecure.html >

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

[8] 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>

[9] *Jean-Christophe Durand*, *Grid Computing: A Conceptual and Practical Study*, Universite de Lausanne, Nov 8, 2004.

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

[11] Jarek Nabrzyski, Jennifer M. Schopf & Jan Welglarz, *Grid Resource Management– State of the art and Future trends*, Kluwer Academic Publisher.

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

[13] Klaus Krauter, Rajkumar Buyya and Muthucumaru Maheswaran, *Taxonomy and survey of Grid Resource Management Systems for Distributed Computing*. Software Practice and Experience Softw. Pract. Exper. 2002.

[14] Manzur Murshed Gippsland, Rajkumar Buyya , "*Using the GridSim ToolKit for Enabling Grid Computing Education*"

[15] Anthony Sulistio, Chee Shin Yeo and Rajkumar Buyya , *A Taxonomy of Computer Based Simulations and its mapping to Parallel and Distributed Systems Simulation Tools*, Grid Computing and Distributed Systems (GRIDS) Laboratory, University of Melbourne, Australia

[16] Howell F, McNab R., *SimJava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling*. Proceedings of the 1st International Conference on Web-based Modelling and Simulation, San Diego, CA. Society for Computer Simulation, 1998.

[17] Rajkumar Buyya and Manzur Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, The Journal of Concurrency and Computation: Practice and Experience, Volume 14, Issue 13-15, Wiley Press, Nov.-Dec., 2002.

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

[19] <http://www.eclipse.org/>

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

[21] <www.gridbus.org>

[22] <www.cs.wisc.edu/condor>

[23] <www.globus.org>

[24] <www.cs.utk.edu/~casanova/NetSolve>

[25] <www.csse.monash.edu.au/~davida/nimrod/nimrodg.htm>

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

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

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

[29] <http://www-csag.ucsd.edu/projects/grid>

[30] <http://grail.sdsc.edu/projects/simgrid/>

[31] <www.iges.org/grads/>

[32] <http://nws.cs.ucsb.edu>

[33] <www.nasa.gov>

Bibliography

- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, *Operating System Principles*, Seventh Edition, John Wiley Publications, ISBN-9812-53-176-9
- Herbet Schildt, *The Complete Reference JAVA-2*, Fifth Edition, Tata Mc-Graw Hill, ISBN-0-07-049543-2
- Kai Hwang*, Advanced Computer Architecture- Parallelism, Scalability and Programmability*, McGraw-Hill International Editions, ISBN-0-07-113342-9
- Thomas H. Cormen. Charles E. Leiserson, Ronald L. Rivest, Cliford Stein, *Introduction to Algorithms*, Second Eastern Economy Edition, Prentice Hall India, ISBN-81-203-2141-3
- Luciano Tarricone and Alessandra Esposito, "*Grid Computing for Electromagnetics*", Artech House, Boston. London, www.artechhouse.com, ISBN-1-58053-777-4.

List of Papers Published/Accepted

1. Gaurav Vohra, Seema Bawa, "*Compile Time Task Scheduling in Alchemi: A Grid Middleware*", **IEEE Student Chapter Conference**: **ELECTICA-2007**, pp 25-28, February 9-11, 2007 at Punjab Engineering College (Deemed University), Chandigarh, INDIA. [*Published].

   *** Awarded the Second Best Paper Prize**

2. Gaurav Vohra, Seema Bawa, "*A Compile Time Task Scheduling Algorithm in Grid Computing Environment*" **International Conference on High Performance Computing, Networking and Communication Systems (HPCNCS-07)** July 9-11, 2007, Orlando, Florida, USA [Accepted].