Efficient Grid Scheduling Algorithm based on Priority Queues

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

Master of Engineering
In
Software Engineering

Thapar University, Patiala

By:
SATYARTH KUMAR SINGH
(80631021)

Under the supervision of:
Mrs. Damandeep Kaur

JUNE 2008

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"An Efficient Grid Scheduling Algorithm based on Priority Queues"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mrs. Damandeep kaur and refers other researcher's works which are duly listed in the reference section.

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

(**Satyarth Kumar Singh)**

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

(**Mrs. Damandeep Kaur)**
Computer Science and Engineering Department
Thapar University, Patiala

Countersigned by

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

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

Acknowledgement

First and foremost, I would like to express my sincere gratitude to my guide **Mrs. Damandeep Kaur**, Lecturer, Computer Science and Engineering Department for immense help, guidance, stimulating suggestions and encouragement all the time with this thesis work. This work would have not been possible without her encouragement. She always provided a motivating and enthusiastic atmosphere to work with; it was a great pleasure to do this thesis under her supervision. The successful completion of this thesis is a direct consequence of the moral and material support extended by **Centre of Excellence in Grid Computing**, TU throughout this thesis.

I am equally grateful to **Dr. Seema Bawa**, Professor and Head, Computer Science and Engineering Department, **Dr. Maninder Singh**, Asst Professor, Computer Science and Engineering Department, **Ms. Inderveer Chana**, Lecturer, Computer Science and Engineering Department. 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.

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

I 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.

SATYARTH KUMAR SINGH
(80631021)

Abstract

Computational Grids are a new trend in distributed computing systems. They allow the sharing of geographically distributed resources in an efficient way, extending the boundaries of what we perceive as distributed computing. Various sciences can benefit from the use of grids to solve CPU-intensive problems, creating potential benefits to the entire society. With further development of grid technology, it is very likely that corporations, universities and public institutions will exploit grids to enhance their computing infrastructure. In recent years there has been a large increase in grid technologies research, which has produced some reference grid implementations. Task scheduling is an integrated part of parallel and distributed computing.

In the distributed environment every node has its own operating system, own resources, own processing speed, so the responsibility of communication between different platforms have been done by middleware. For this purpose, middleware takes the jobs from users and according to the job specification resource discovery has been done by middleware as a first step then assign the job to the particular resource, which has been discovered. So the middleware is responsible for the resource discovery, resource allocation and job will be executed successively in the grid environment. Due to this, performance of the job execution is affected. To reduce the load from the middleware, this thesis proposed the approach, which is “An efficient grid scheduling based on the priority based multiple queues”.

An efficient Grid scheduling system is an essential part of the Grid. Even though middleware support for grid computing has been the subject of extensive research, scheduling policies for the grid context have not been much studied. In addition to processor utilization, it is important to consider the waiting time, throughput, and response times of jobs in evaluating the performance of grid scheduling strategies. This thesis propose distributed scheduling algorithm that use priority based multiple queue without much loss of performance.

Table of Contents

List of Figures

List of Figures

Figure No. **Title** **Page No.**

List of Tables

Table No. **Title**

Organization of Thesis

The **first chapter** briefly introduces Grid computing concepts, an example of Grid computing, Grid architecture and its comparison with other distributed technologies. It also gives insight into the thrust areas of Grid computing, brief introduction of Grid Scheduling. Basic objective of the thesis is also explained at the end of this chapter

The **second chapter** gives the literature reviewed. It explains what Grid scheduler is, its architecture and various challenges in grid scheduling. It also describes the Grid scheduling process, types of Grid scheduling and various approaches associated with it. Later on it explains how distributed algorithms can be used for Grid scheduling, comparison among various existing Grid scheduling approaches and a brief introduction about Grid simulation.

The **third chapter** is proposed scheduling algorithm which contains problem formulation; approach followed for implementing the algorithm**,** and describes the proposed Grid scheduling algorithm in detail.

The **fourth chapter** describes implementation detail and provides the experimental result of the priority based multiple queues scheduling algorithm.

The **fifth chapter** concludes the thesis work, gives the contribution of thesis and what future work can on done on it.

Chapter 1

Introduction

1.1 Introduction to 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 [1] 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 [2].

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 [3] 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).

1.1.1 Grid Application: An Example

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.

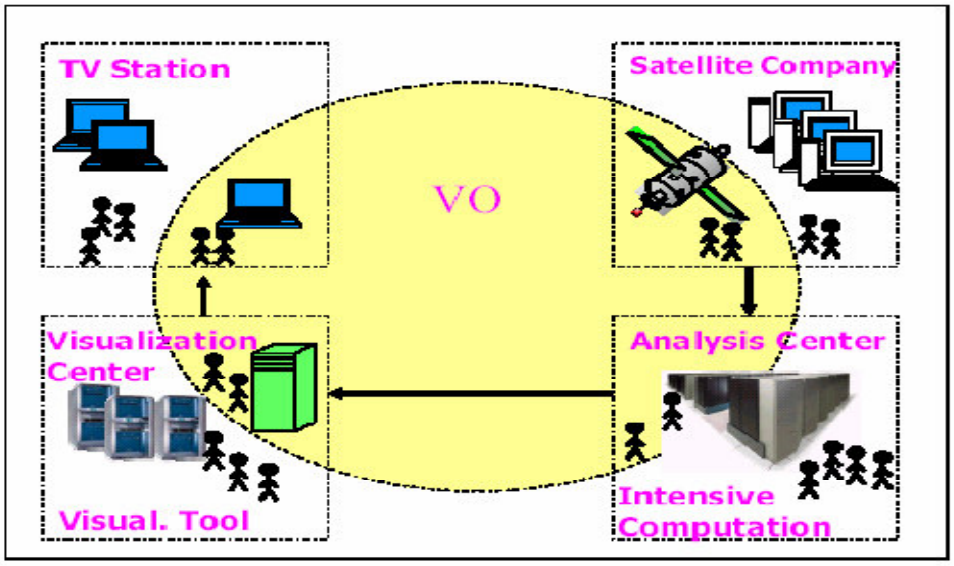


Figure 1.1: Weather prediction using Grid [4]

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 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].

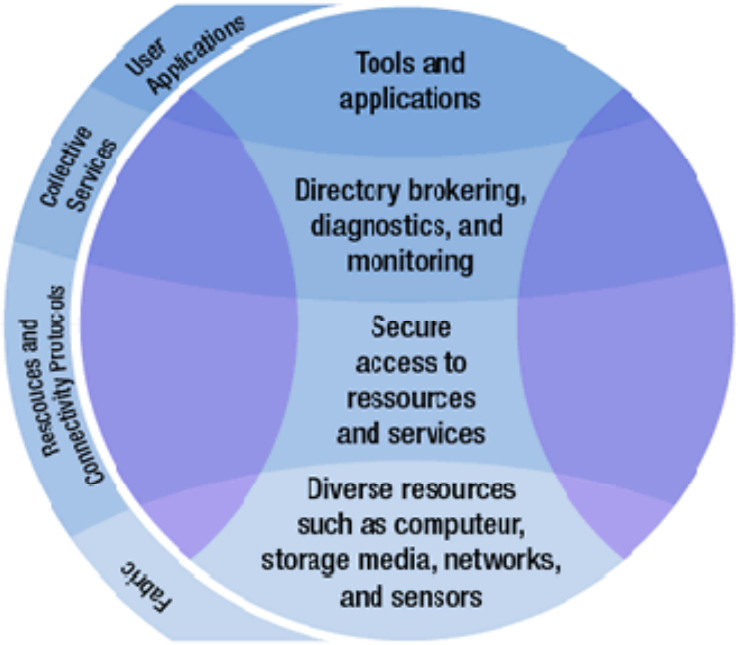


Figure 1.2: Layered Grid Architecture [6]

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].

1.3 Comparison with Other Distributed Computing Technologies

1.3.1 Comparison with Cluster Computing

When two or more computers are used together to solve a problem, it is called a computer cluster. Grid computing is something similar to cluster computing, it makes use of several computers connected is some way, to solve a large problem [7]. 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.

1.3.2 Comparison with CORBA

Computational Grids infrastructures are aimed at allowing the programmer to aggregate powerful and sophisticated resources scattered around the globe. To enable this goal, the Grid computing community has concentrated on the creation of advanced services that allow access to high-end remote resources such as batch systems at supercomputing

centers, large-scale storage systems, large scale instruments, and remote applications. 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). In order to provide interoperability between diverse heterogeneous implementations, CORBA [8] defines interoperability mechanisms. A high-level, distributed computing model, vendor independence, and a strong interoperability thrust all combined to make CORBA an attractive and popular distributed computing standard [9]. 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.

1.3.3 Comparison with P2P

Peer-to-Peer (P2P) networks and Grids share the same focus on harnessing resources across multiple administrative domains. Grid support for a variety of applications, and focus on providing infrastructure with quality of service to moderate sized, homogeneous, and partially trusted communities whereas P2P support for intermittent participation in vertically integrated applications for significantly larger communities of untrusted, anonymous individual Grids usually have some form of centralized management and security which P2P systems lack, making them ideal for providing anonymity [10]. Grids used for complex applications increase from tens to thousands of nodes, their functionalities should be decentralized to avoid bottlenecks. The P2P model could help to ensure Grid scalability. Designers could use the P2P philosophy and techniques to implement nonhierarchical decentralized Grid systems. A P2P approach is needed both to implement Grid tools and services, and design and develop Grid applications that must access and coordinate remote resources and services. P2P systems are generally far more scalable than Grid Computing systems. P2P systems are generally more tolerant of single-point failures than Grids.

1.4 Major Components of Grid Computing

Grid Computing has following major components [11]:

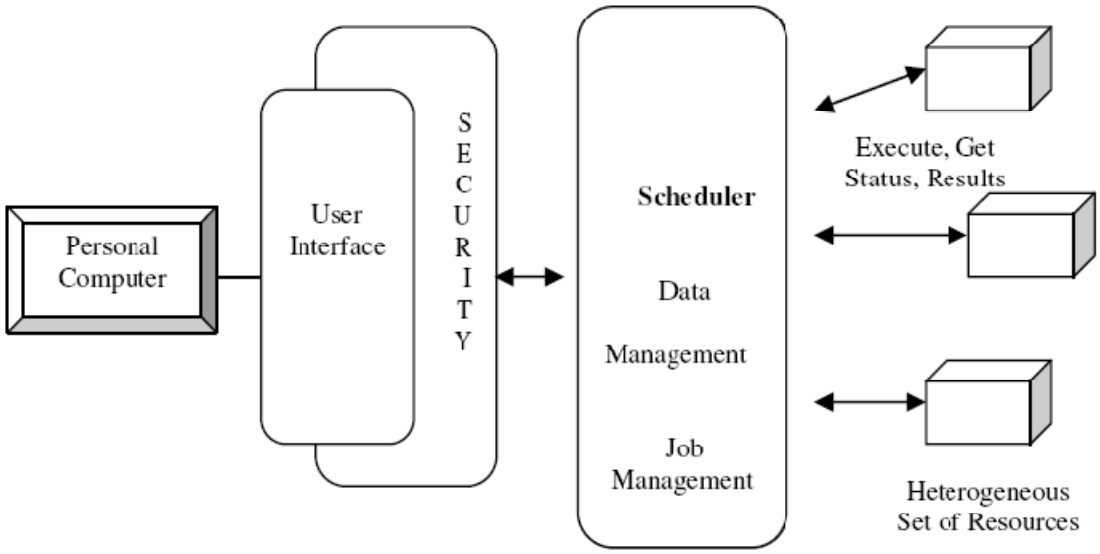


Figure 1.3: Components of Grid [11]

1.4.1 User Interface

Accessing information on the Grid is fairly important, and the user interface component handles this task for the user. User interface sits between the user and the Grid and user can only interact to Grid using the user interface. .

1.4.2 Security

Computers on a Grid are networked. They can 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.

1.4.3 Workload management

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

1.4.4 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.

1.4.5 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. 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.

1.5 Grid Scheduling

Grid is a collaborative problem-solving environment in which one or more user jobs can be submitted without knowing where the resources are or even who own the resources. Grid scheduling [12] is the process of scheduling applications over Grid resources. A Grid scheduler is different from local scheduler in that a local scheduler only manages a single site or cluster and usually owns the resource.

1.6 Objective of Thesis

Existing approaches of Grid scheduling doesn't give much emphasis on the performance of a Grid scheduler. Grid schedulers allocate resources to the jobs to be executed on First come First serve basis. If we provide an already optimize queue to the scheduler using various scheduling methods e.g. Shortest Job First, Priority based scheduling etc, the performance of a scheduler can be improved. The objective of this thesis work is to design, develop and implement a scheduling algorithm for Grid on top of existing scheduler in order to improve its performance by providing optimized queues to the scheduler. Queues can be optimized by using various scheduling algorithms depending upon the performance criteria to be improved e.g. response time, throughput. Main tasks undertaken and methodology followed in the thesis are:

- A study of the current Grid schedulers, their architecture, Grid scheduling approaches and challenges involved in various types of existing Grid scheduling algorithms and how they evolved.
- A survey of Grid Simulation Tools and Techniques e.g. GridSim and Netbeans which are the open source tools.
- Design, Implementation and Simulation of the proposed Grid scheduling algorithm.

Chapter 2

Review of Literature

2.1 A Generic Grid Scheduler Architecture

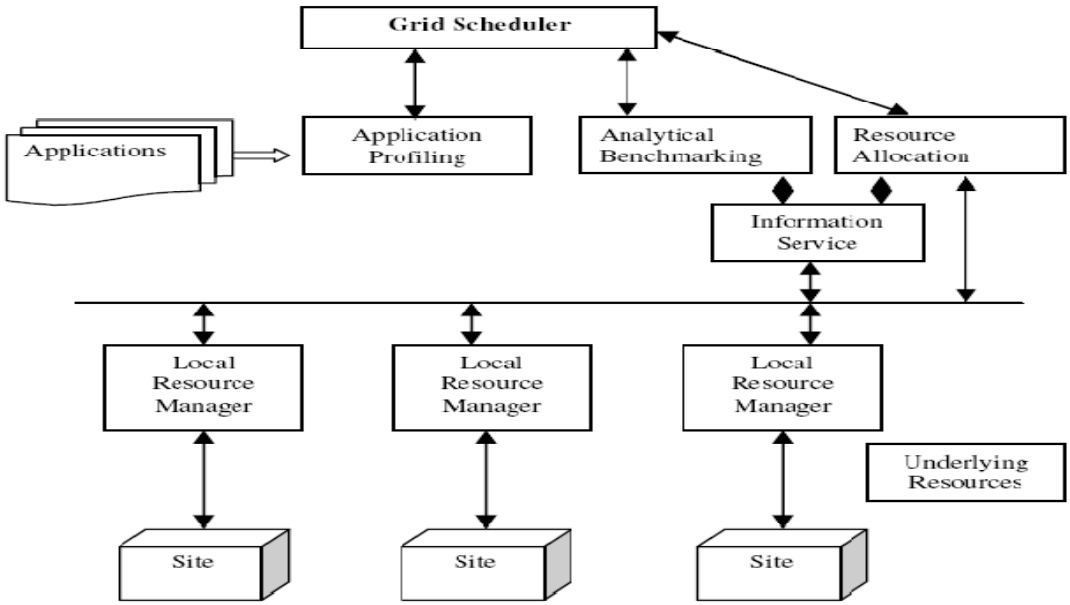


Figure 2.1: General Grid Scheduler Architecture [13]

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 [13].

Real resources lay 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 [6] 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) and Network Weather Service (NWS) are two best examples that provide information service. [13].

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 [13].

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 unauthorized 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. [6]

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 Process

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 [3] 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:

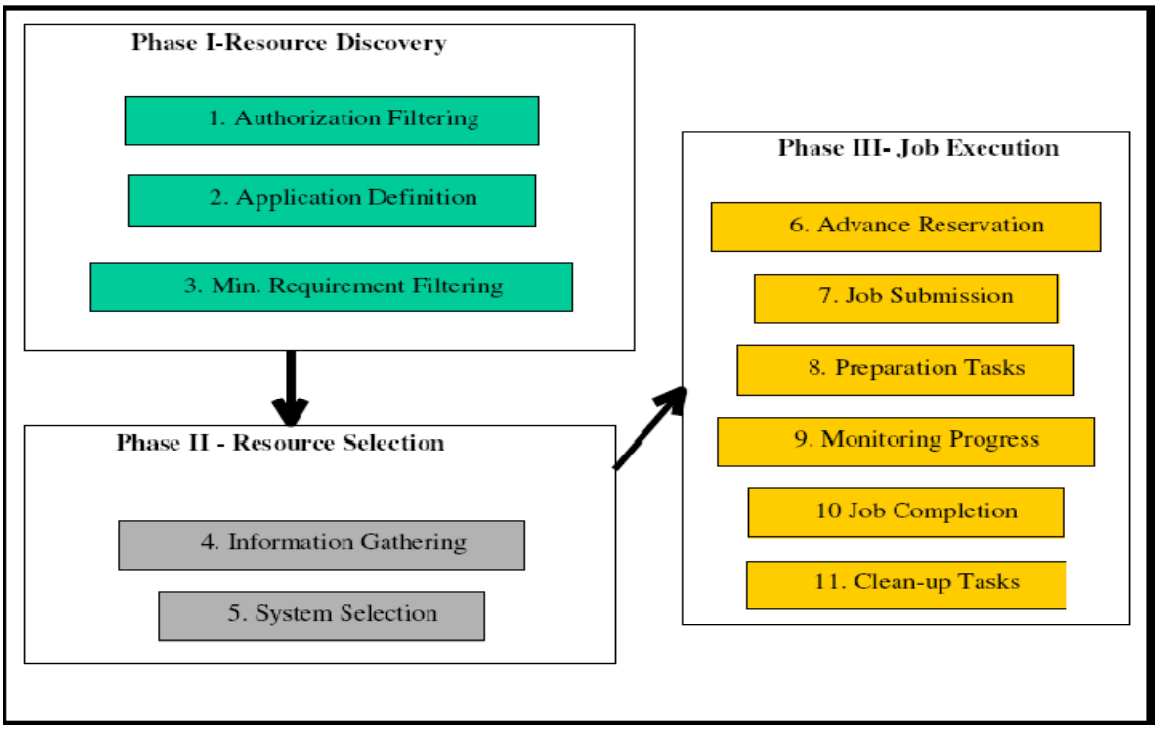


Figure 2.2: Grid Scheduling Process [3]

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.

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.

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 [3]:

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.

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 [3]:

Make an Advance Reservation (Optional): 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.

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.

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) was considered unsuccessful because it did not address the need to stage files automatically.

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

Find out if Job is done: When the job is finished, the user needs to be notified.

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 Types of Grid Scheduling

There have been a number of efforts attempting to design scheduling systems for Grids, each having its unique features. A comprehensive set of different types of scheduling [14] is discussed in this section.

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* 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 [15] 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.

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.

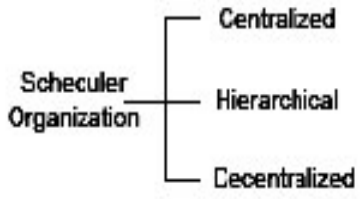


Figure 2.3 Scheduler Organization Types [16]

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 [17] is a scheduling system which exploits idea of economy mechanism.

2.6 Parallel and Distributed Vs Grid Scheduling Algorithms

In [18], Casavant et al propose a hierarchical taxonomy for scheduling algorithms in general-purpose parallel and distributed computing systems. Since Grid is a special kind of such systems, scheduling algorithms in Grid can be treated as a subset of scheduling algorithms for parallel and distributed computing. From the top to the bottom, this subset can be identified as follows [19].

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 to the processor. While 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. These variances introduce forms of non-determinism into the running program [20]. 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 make span 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, the responsibility for making global scheduling decisions may lie with one centralized scheduler, or can 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 [21]:

Table2.1 Summary of Current Approaches in Grid Scheduling [21]

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.

2.9 Grid Simulation

Simulation is the imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviors of a selected physical or abstract system. Grid environment also can be simulated using several Grid simulators e.g. GridSim [22], Eclipse [23] etc. Grid simulators enable Grid users to work on Grid like environment without having to worry about the other external factors that may influence the Grid environment.

More so, 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 [22]:

- 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.

2.9.1 GridSim

GridSim[22] is a software platform that enables users to model and simulate the characteristics of Grid resources and networks with different configurations. Study Grids, or test new algorithms and strategies in a controlled environment. By using GridSim, they are able to perform repeatable experiments and studies that are not possible in a real dynamic Grid environment. Some of the GridSim features are outlined below:

- It allows modeling of different resource characteristics and their failure properties;
- It enables simulation of workload traces taken from real supercomputers;
- It allocates incoming jobs based on space-or time-shared mode;
- It supports reservation-based or auction mechanisms for resource allocation;
- It has the ability to schedule compute- and/or data-intensive jobs;
- It provides clear and well-defined interfaces for implementing different resource allocation algorithms;
- It allows modeling of several regional Grid Information Service (GIS) components.

In this thesis, the Grid simulator used is GridSim.

Chapter 3

Proposed Scheduling Algorithm

3.1 Problem Formulation

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 Quality of Service goals.

At the crux of the Grid scheduling system lays 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. 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 Proposed Approach

In this thesis, the approach followed consists of a use of multiple priority based scheduling with different algorithm used in each queue. The scheduler allocates the resource to the rightmost job of the topmost queue. The multiple queue scheduling algorithms, allows a process to move between queues. Every queue has its own scheduling algorithm. There are three scheduling algorithm (First Come First Serve, Shortest Job First, Round Robin) used for different queues. The queues are arranged in a

priority order with topmost queue having the highest priority. The idea is to separate processes according to the priority associated with it. If a process does not have any priority, it will be moved to a lowest-priority queue. This scheme leaves I/O-bound and interactive processes in the higher-priority queues. In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation [16]. Here, the three algorithms are used in all possible combinations for each queue in order to find out a best combination for a particular type of job and according to certain performance criteria.

For example, consider a multiple queue scheduler with three queues, numbered from 0 to 2 (figure-3.2). The scheduler will only execute first queue, job will jump to the next higher priority queue after some time span. When job will come then it placed into the queue according to the priority. [16]

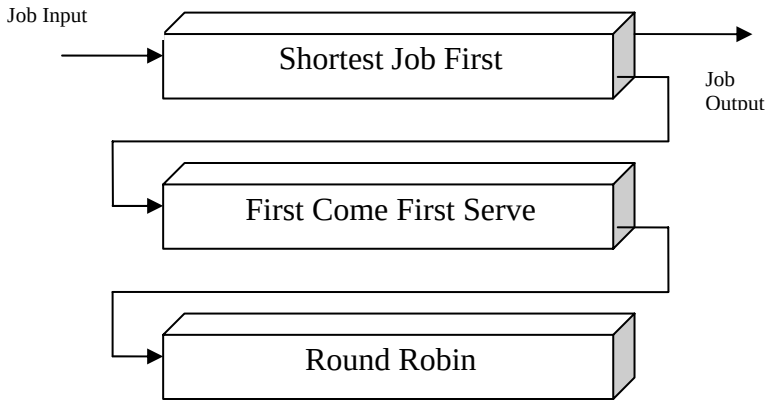


Figure-3.1 Priority Based Multiple Queues

In general, a multiple queue scheduler is defined by following parameter:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine which job assign to which priority queue.

- The method used to determine when to upgrade a process to higher-priority queue.
- The method used to determine which queue a process will enter that process needs service.

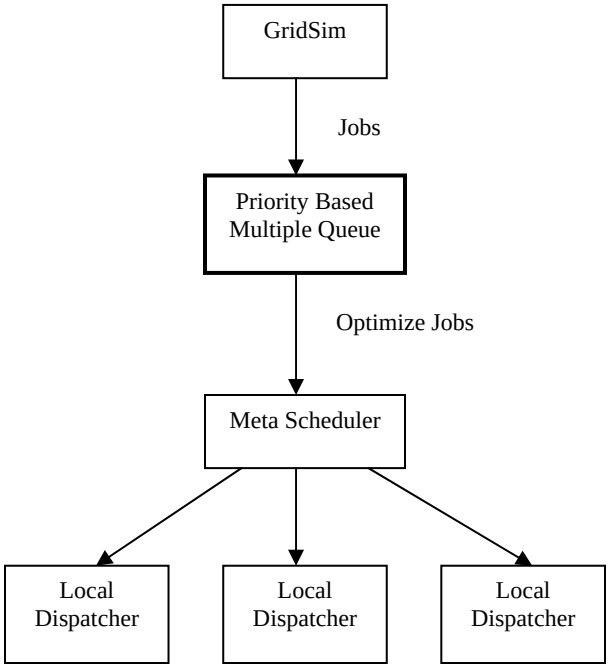


Figure-3.2 Block diagram for Implementation of Priority Based Multiple Queue

User send the job for execute in distributed environment, using GridSim. User can set the priority of the jobs, like which job is time effective, which one is cost effective, which one is economical etc. If user does not set priority of the job then it will be submit to default queue. GridSim will submit to the Priority based multiple queue according to the priority. This priority queue will optimize the job sequence and submit to the Grid Meta scheduler. That will be responsible for the job execution and completions.

3.3 Priority based Multiple Queues Scheduling Algorithm

Multiple Queues scheduling (Number of jobs N_j)

Step 1:- A consumer submits the jobs (j_i).

Step 2:- According to the priority (P) of the job (j), job will placed into the respective queue (Q) in each combination of multiple queues. Different combinations are shown below:

Combinations 1:- First Come First Serve -> Shortest Job First -> Round Robin

**Algorithm for job scheduling*

Combinations 2:- First Come First Serve -> Round Robin -> Shortest Job First

**Algorithm for job scheduling*

Combinations 3:- Shortest Job First -> First Come First Serve -> Round Robin

**Algorithm for job scheduling*

Combinations 4:- Shortest Job First -> Round Robin -> First Come First Serve

**Algorithm for job scheduling*

Combinations 5:- Round Robin -> First Come First Serve -> Shortest Job First

**Algorithm for job scheduling*

Combinations 6:- Round Robin -> Shortest Job First -> First Come First Serve

**Algorithm for job scheduling*

Step 3:- Finally analyze the different combination result and find the best combination for the required parameter. e.g best according to waiting time, response time etc.

Step 4:- Best combination's sequence of job will be submitted to the actual middleware scheduler, which are responsible for actual job execution and completion.

Step 5:- End

*****Algorithm for job scheduling

Job Scheduling (jobs, priority)

Step 1:- Jobs (j_i) will be placed into their respective queue according to the priority.

Step 2:- Calculation

->Average Waiting Time (W_t) for jobs

$$Wt = (W_{j1} + W_{j2} + \ldots.. + W_{ji}) / N_j \ ;$$

->Average Response Time (R_t) for jobs

$$Rt = (R_{j1} + R_{j2} + \ldots.. + R_{ji}) / N_j \ ;$$

->Average Turnaround Time (T_t) for jobs

$$Tt = (T_{j1} + T_{j2} + \ldots.. + T_{ji}) / N_j \ ;$$

Step 3:- For avoiding the starvation, job will jump to the next higher priority queue after some fixed time interval.

Step 4:- End

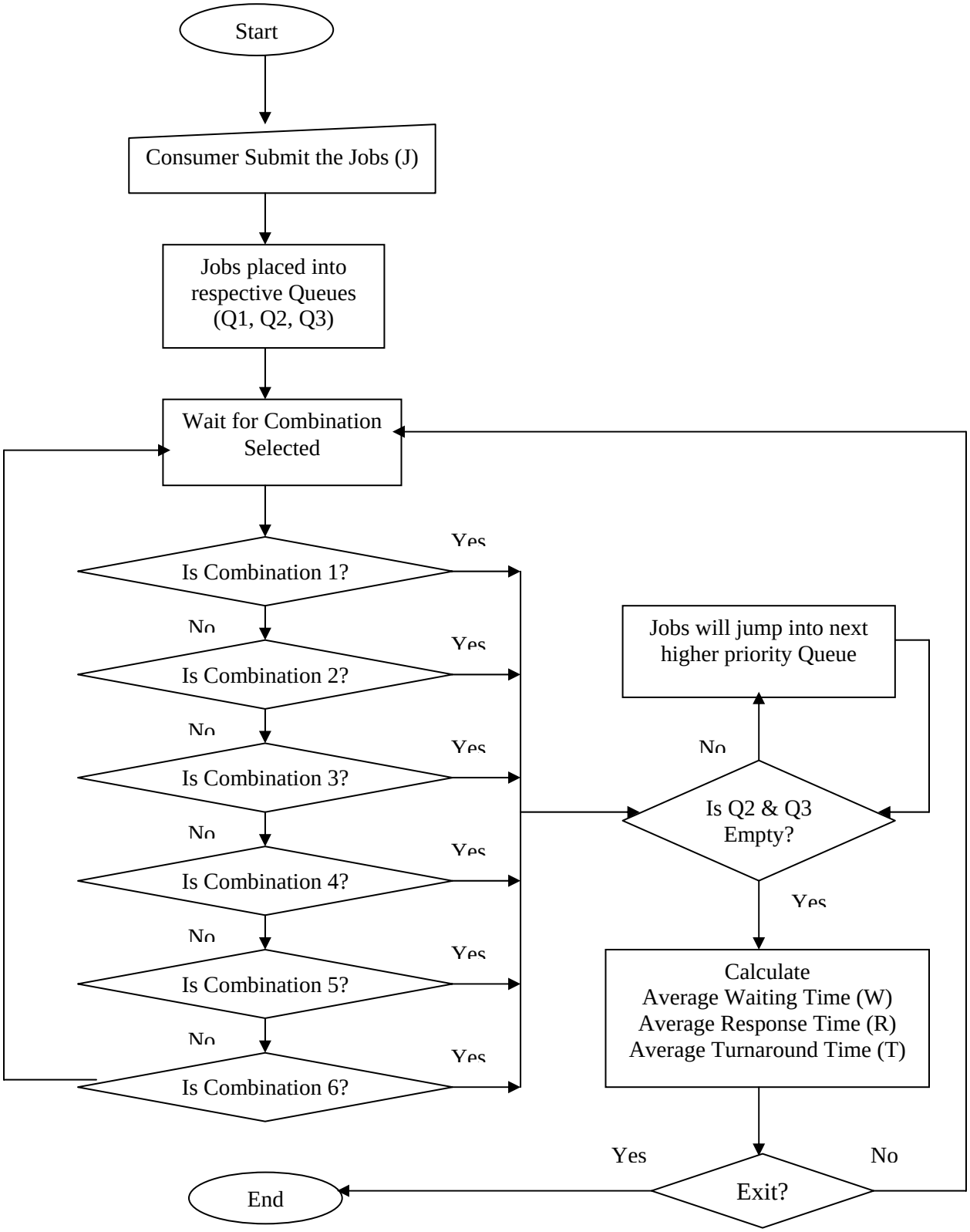


Figure-3.3 Flow Chart for Multiple Queues Scheduling Algorithm

Chapter 4

Implementation and Experimental Details

Throughout the implementation of the Priority Based Multiple Queue Algorithm emphasis has been on the use of Open Source Technologies and Toolkits. The Priority Based Multiple Queue Algorithm has been implemented in JAVA programming language on top of GridSim [24] Toolkit 4.0. The steps followed for the implementation are described below in detail with appropriate screenshots.

4.1 Installation of Pre-requisites and Necessary Components

4.1.1 Installation of GridSim Toolkit

In the current implementation of the Priority Based Multiple Queue 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 [16].

The GridSim [24] 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.

4.1.2 Salient Features of GridSim [24]

- 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.

4.1.3 GridSim Architecture

GridSim has employed a layered and modular architecture for Grid simulation to leverage existing technologies and manage them as separate components. A multilayer

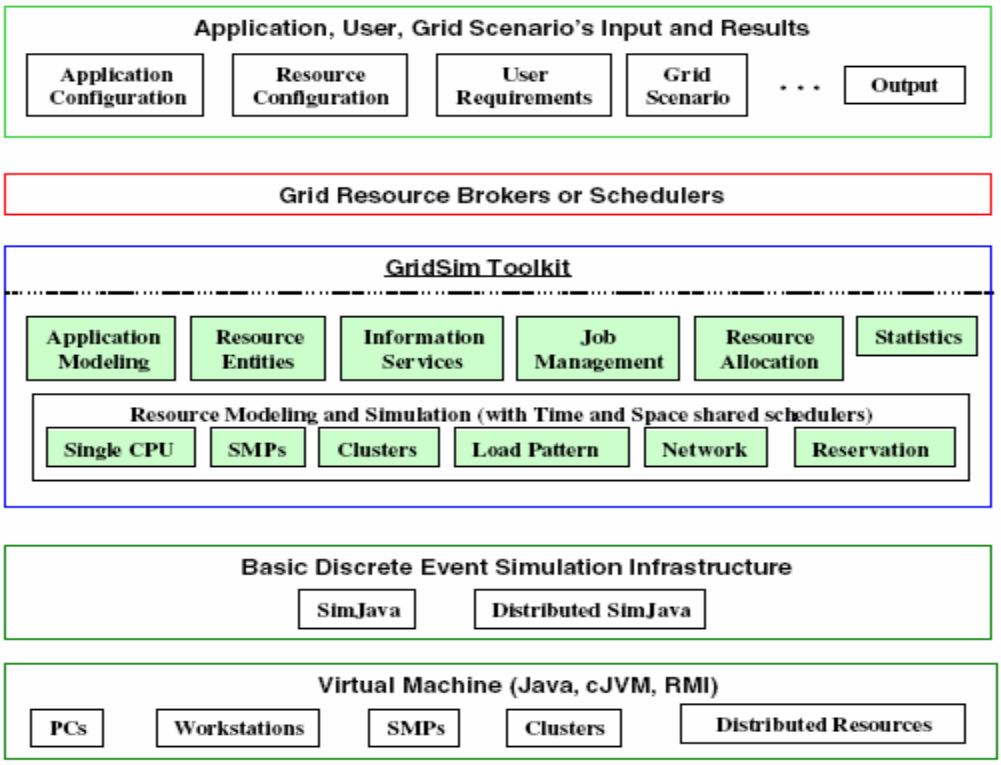


Figure 4.1 Architecture for GridSim platform and components [24]

architecture and abstraction for the development of GridSim platform and its applications is shown in Figure 4.1.

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 [25]. 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.

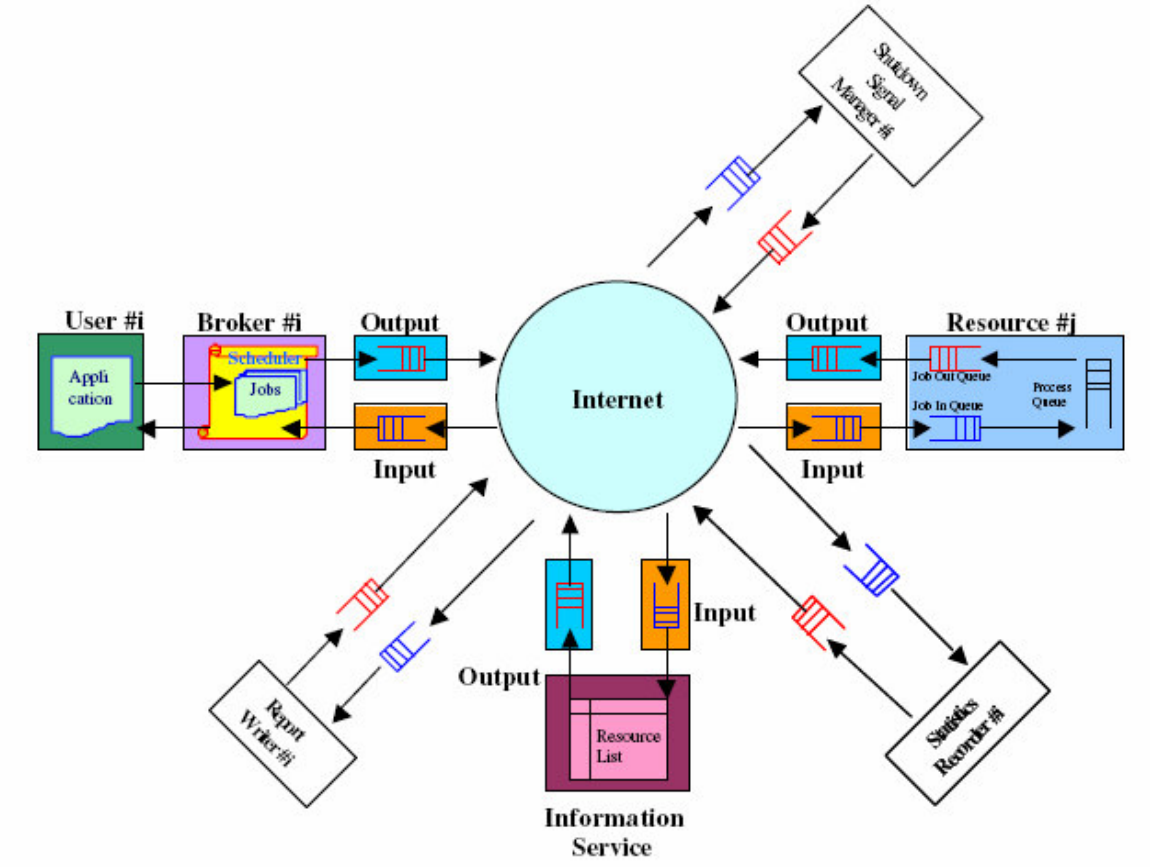


Figure 4.2 A Flow Diagram for GridSim based Simulations [24]

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.

4.2 Implementation of Priority Based Multiple Queue Scheduling Algorithm in GridSim

To use the GridSim for the implementation and simulation of the Priority Based Multiple Queue algorithm we need to follow the steps mentioned in Appendix-A; 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.

The complete Priority Based Multiple Queue algorithm is implemented in JAVA and run on top of GridSim version 4.0 Toolkit. This chapter explains and discusses the results obtained in the statistical light.

In the starting window the user is prompted to enter the values for the number of Jobs needed to be simulated.

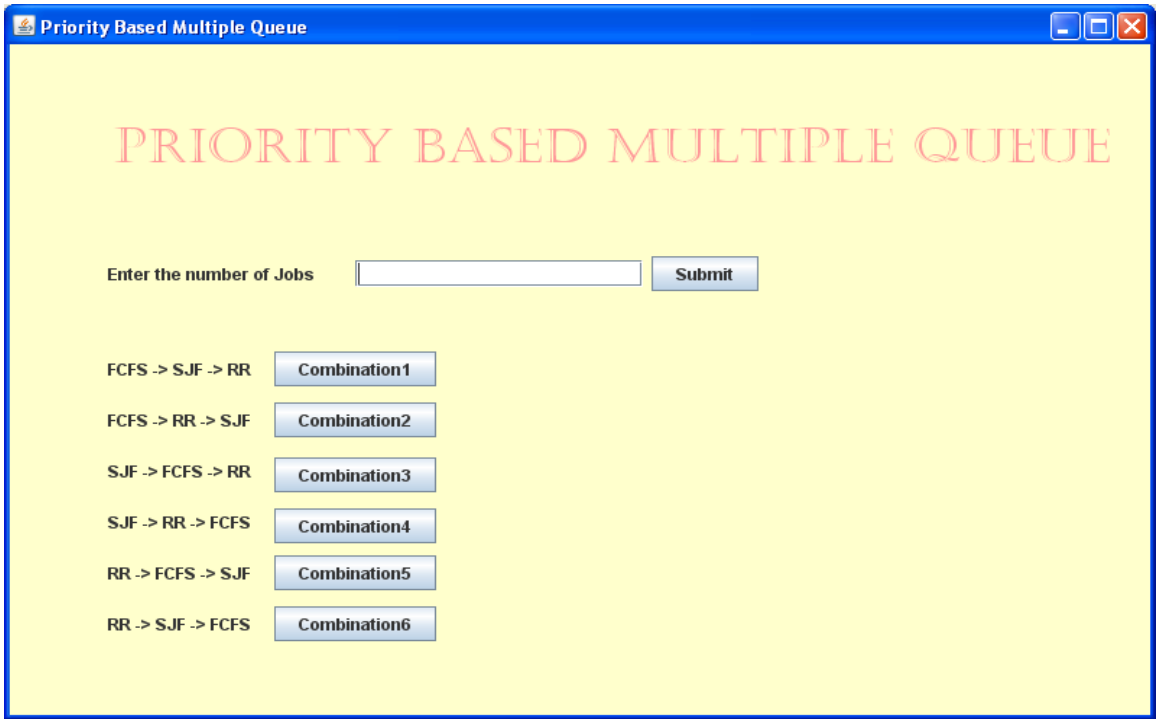


Figure 4.3 User Input Environment

Due to the GridSim, jobs will be prepared in simulated environment which shows in following figure. After that jobs will assign to the respective queue.

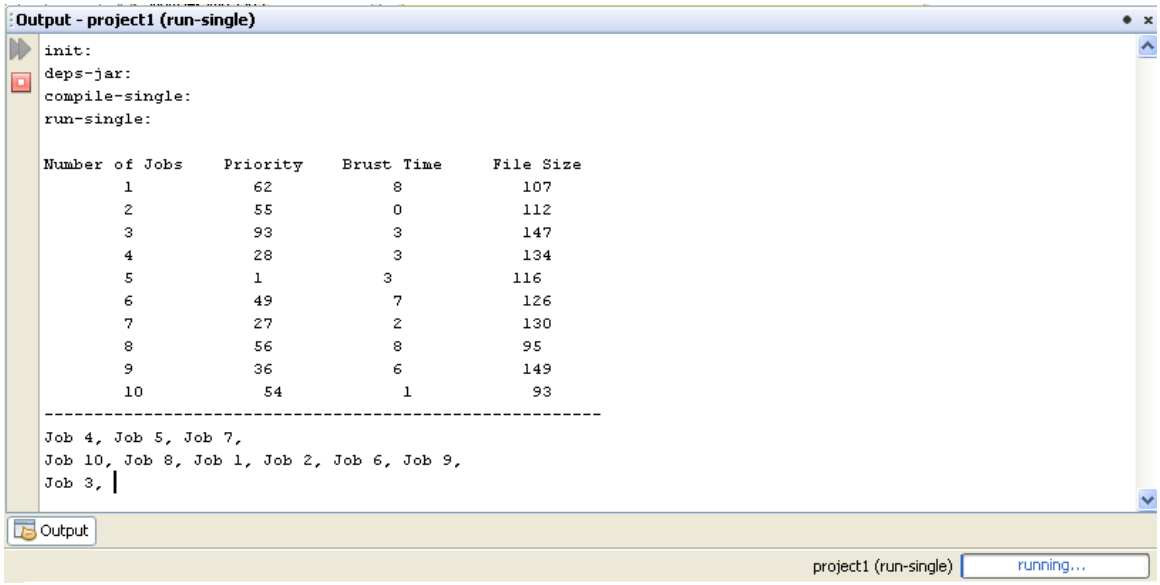


Figure 4.4 Job Created by GridSim

The GridSim toolkit supports modeling and simulation of a wide range of heterogeneous resources, such as single or multiprocessors, shared and distributed memory machines such as PCs, workstations, SMPs, and clusters with different capabilities and configurations. It can be used for modeling and simulation of application scheduling on various classes of parallel and distributed computing systems such as clusters [26], Grids, and P2P networks [27]. The resources in clusters are located in a single administrative domain and managed by a single entity, whereas in Grid and P2P systems, resources are geographically distributed across multiple administrative domains with their own management policies and goals. Another key difference between cluster and Grid/P2P systems arises from the way application scheduling is performed. The GridSim toolkit provides facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. These features can be used to simulate resource brokers or Grid schedulers for evaluating performance of scheduling algorithms or heuristics.

GridSim create the jobs and assigned to the respective priority queue. There are three queues with three different scheduling algorithms. There are six combinations of the different queues. Same input (job) to every combination of queues and determine which one the best combination for execution of jobs. Figure-2 is self-explanatory, that what the role of the Priority based multiple queue is. It will take the job form the user in simulated environment. Optimize the sequence of the job, and submitted to Meta Grid scheduler, which is responsible for the job execution and job completions. In this way priority based multiple queue can increase the performance of the Meta scheduler.

For example, consider the scenario of the jobs in which jobs comes after regular intervals having burst time and priority. If user does not specify any priority then that job will be assign to the last queue of the multiple queue. The job scenario table below:

Table 4.1: Jobs Scenario Table for the Priority Based Multiple Queues.

Jobs	**Arrival Time (millisecond)**	**Burst Time (milliseconds)**	**Priority**
Job 1	0	8	62
Job 2	1	0	55
Job 3	2	3	93
Job 4	3	3	31
Job 5	4	3	1
Job 6	5	7	49
Job 7	6	2	30
Job 8	7	8	56
Job 9	8	6	36
Job 10	9	1	54

In the priority based multiple queues, have three queues, each having its own algorithm for job arrangement in their respective queue. So there are six different combinations of these three queues and try to identify which combination is best for the implementation. First combination (Figure 4.5) is First Come First Serve in first queue, Shortest Job First in second queue and Round Robin in the last queue (FCFS -> SJF -> RR). At last jobs will arrange in the sequence according to the first combination. Calculate the average

waiting time, response time, and turn around time for first combination. That jobs sequence will send to the Grid Meta scheduler which is in optimize manner, which shows in following figure.

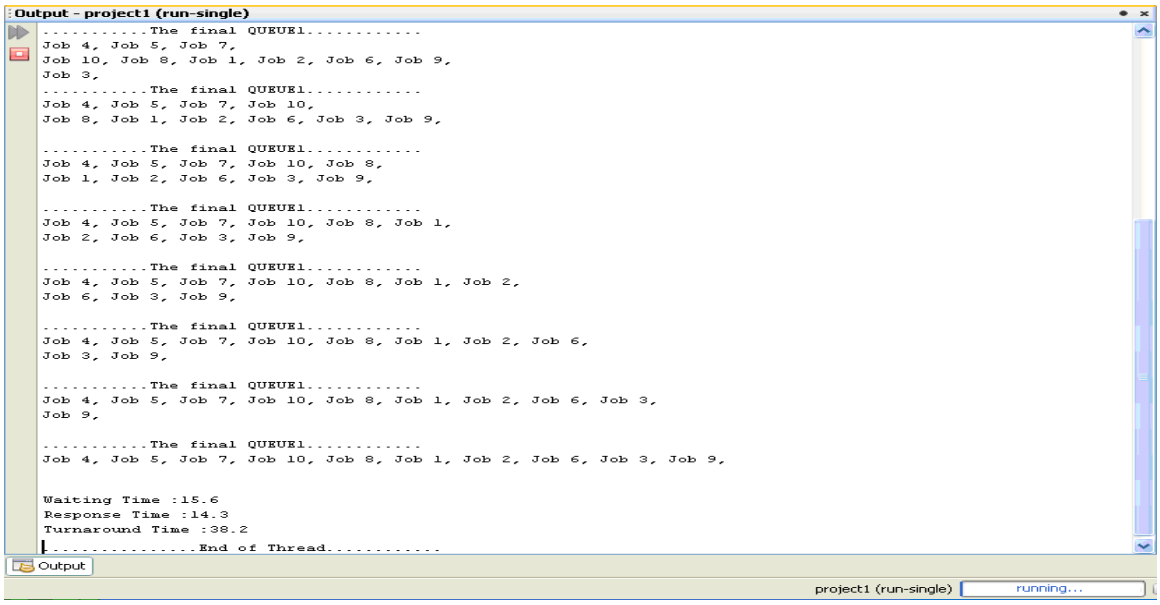


Figure 4.5 Final Queue Preparations in First Combination

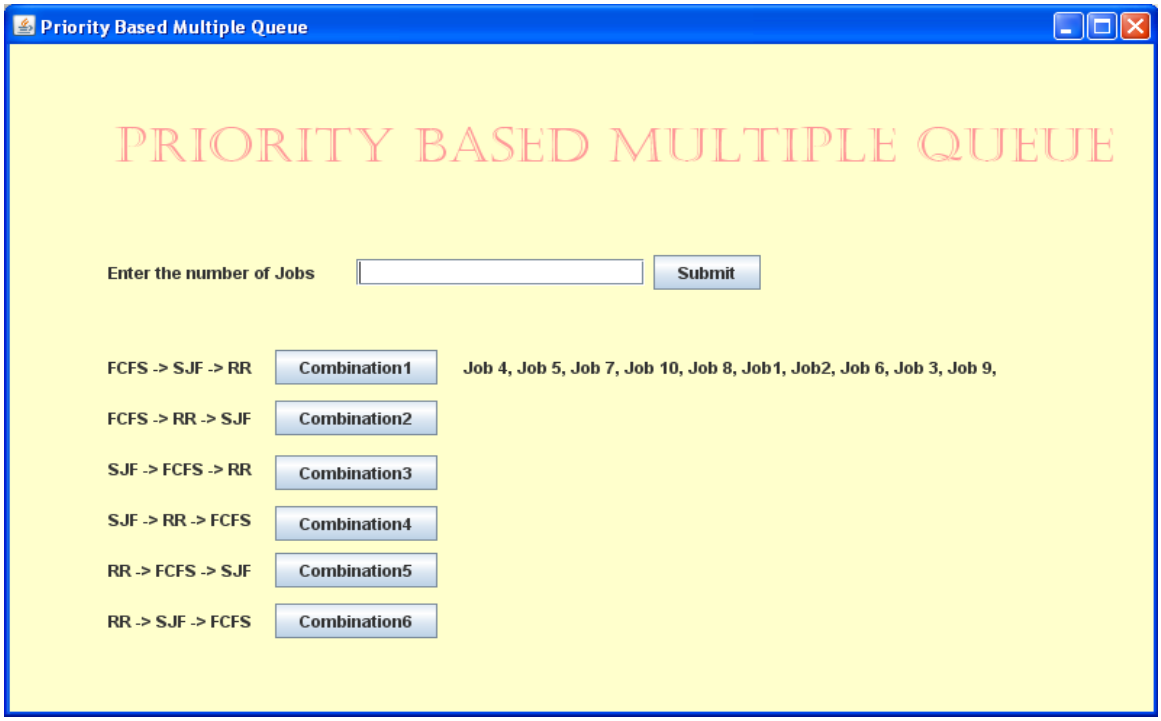


Figure 4.6 Waiting for Combination Selection

Second combination (Figure 4.7) is First Come First Serve in first queue, Round Robin in Second queue and Shortest Job First in the last queue (FCFS -> RR -> SJF). At last jobs will arrange in the sequence according to the second combination. Calculate the average waiting time, response time, and turn around time for second combination. That jobs sequence will send to the Grid Meta scheduler which is in optimize manner, which shows in following figure.

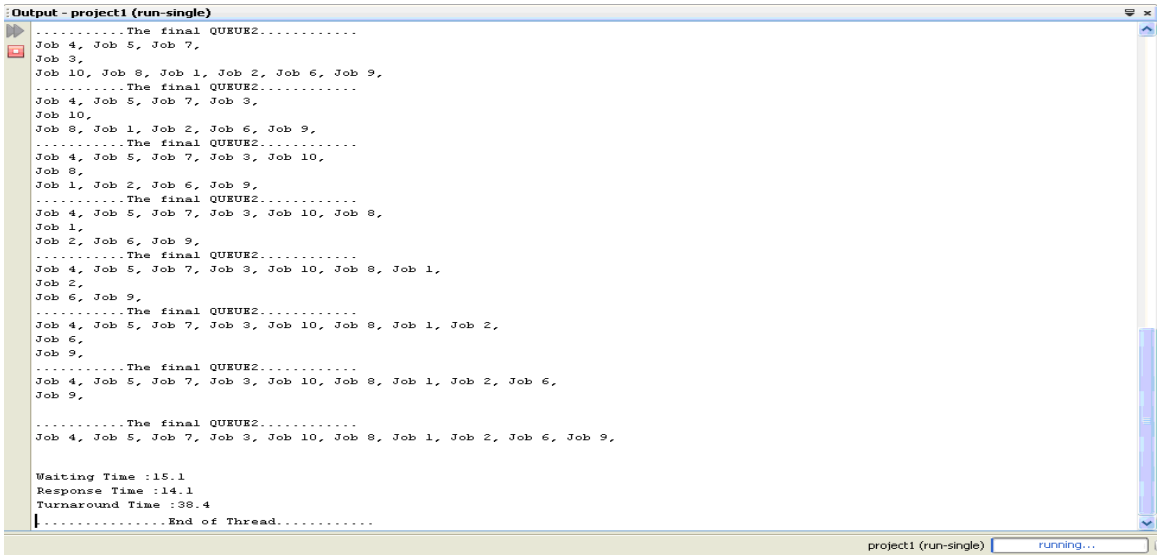


Figure 4.7 Final Queue Preparations in Second Combination

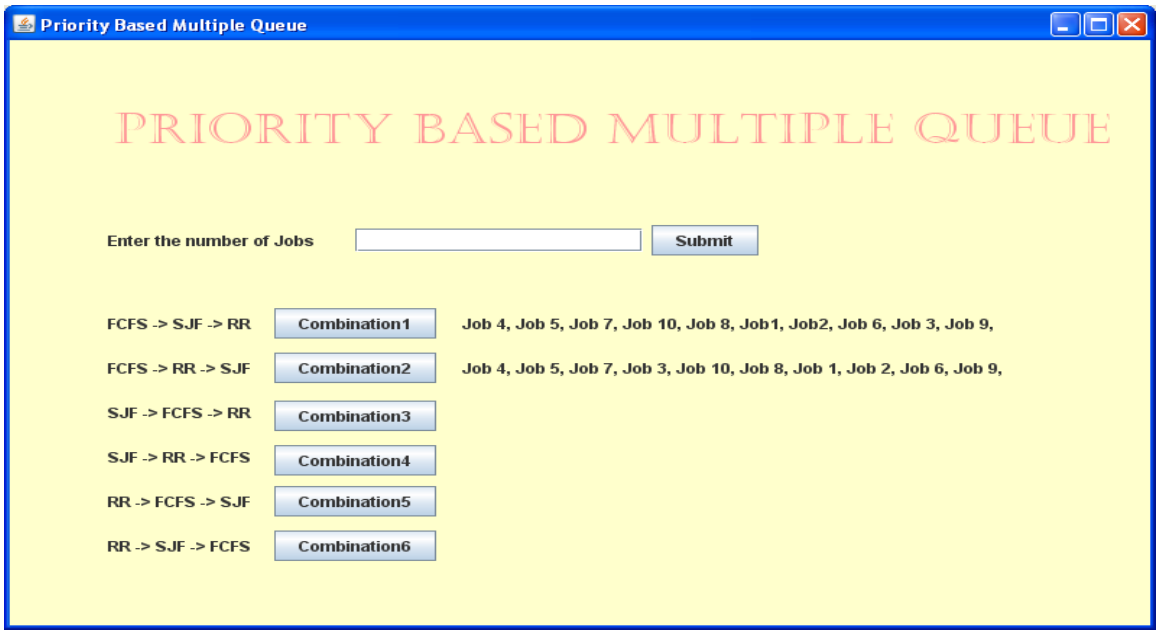


Figure 4.8 Waiting for Combination Selection

Similarly in third combination (Figure 4.9) is Shortest Job First in first queue, First Come First Serve in second queue and Round Robin in the last queue (SJF -> FCFS -> RR). At last jobs will arrange in the sequence according to the third combination. Calculate the average waiting time, response time, and turn around time for second combination. In forth combination (Figure 4.10) is Shortest Job First in first queue, Round Robin in second queue and First Come First Serve in the last queue (SJF -> RR -> FCFS). At last jobs will arrange in the sequence according to the forth combination. Calculate the average waiting time, response time, and turn around time for second combination. In fifth combination (Figure 4.11) is Round Robin in first queue, First Come First Serve in second queue and Shortest Job First in the last queue (RR -> FCFS ->SJF). At last jobs will arrange in the sequence according to the fifth combination. Calculate the average waiting time, response time, and turn around time for second combination. In sixth combination (Figure 4.12) is Round Robin in first queue, Shortest Job First in second queue and First Come First Serve in the last queue (RR -> SJF -> FCFS). At last jobs will arrange in the sequence according to the sixth combination. Calculate the average waiting time, response time, and turn around time for second combination. That jobs sequence will send to the Grid Meta scheduler which is in optimize manner, which shows in following figures.

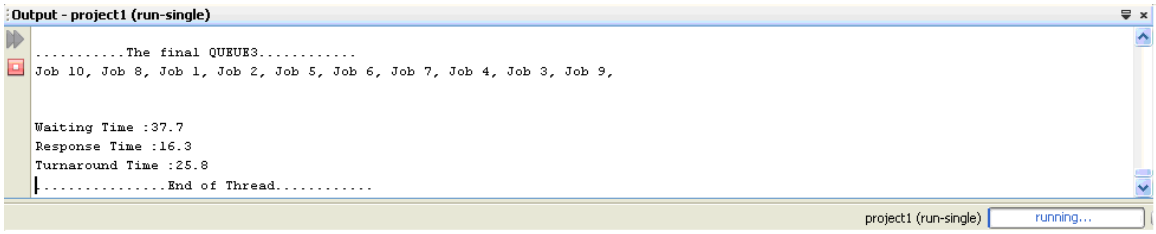


Figure 4.9 Final Queue Preparations in Third Combination

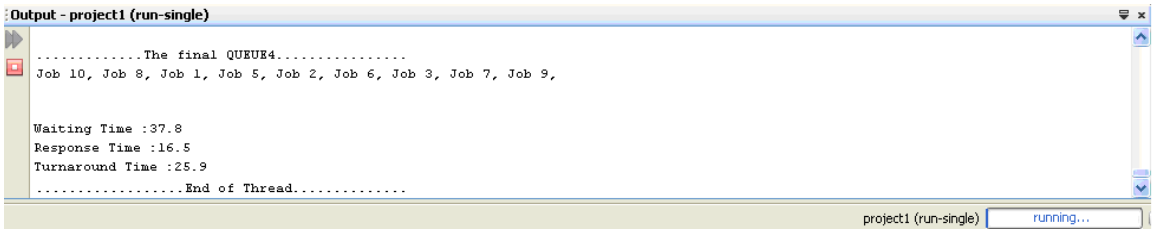


Figure 4.10 Final Queue Preparations in Fourth Combination

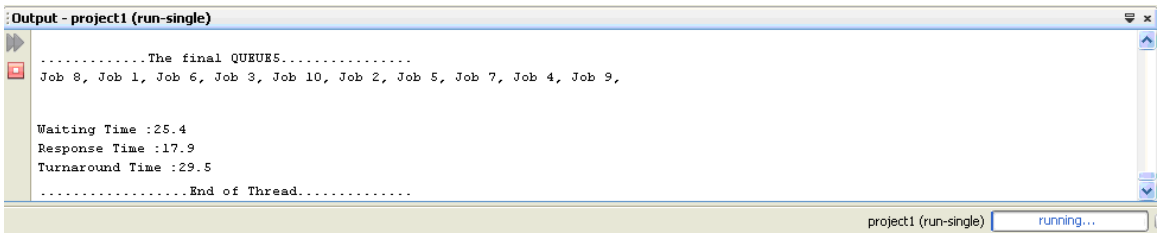


Figure 4.11 Final Queue Preparations in Fifth Combination

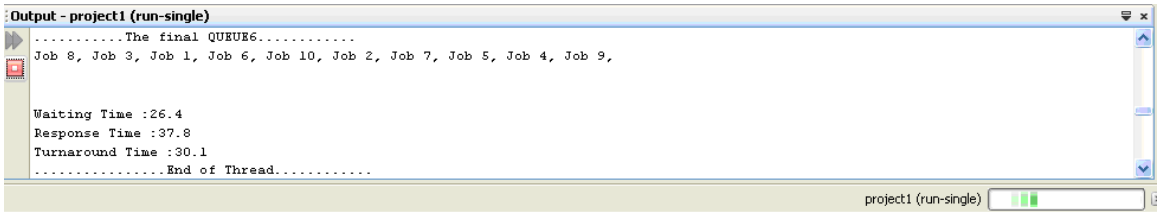


Figure 4.12 Final Queue Preparations in Sixth Combination

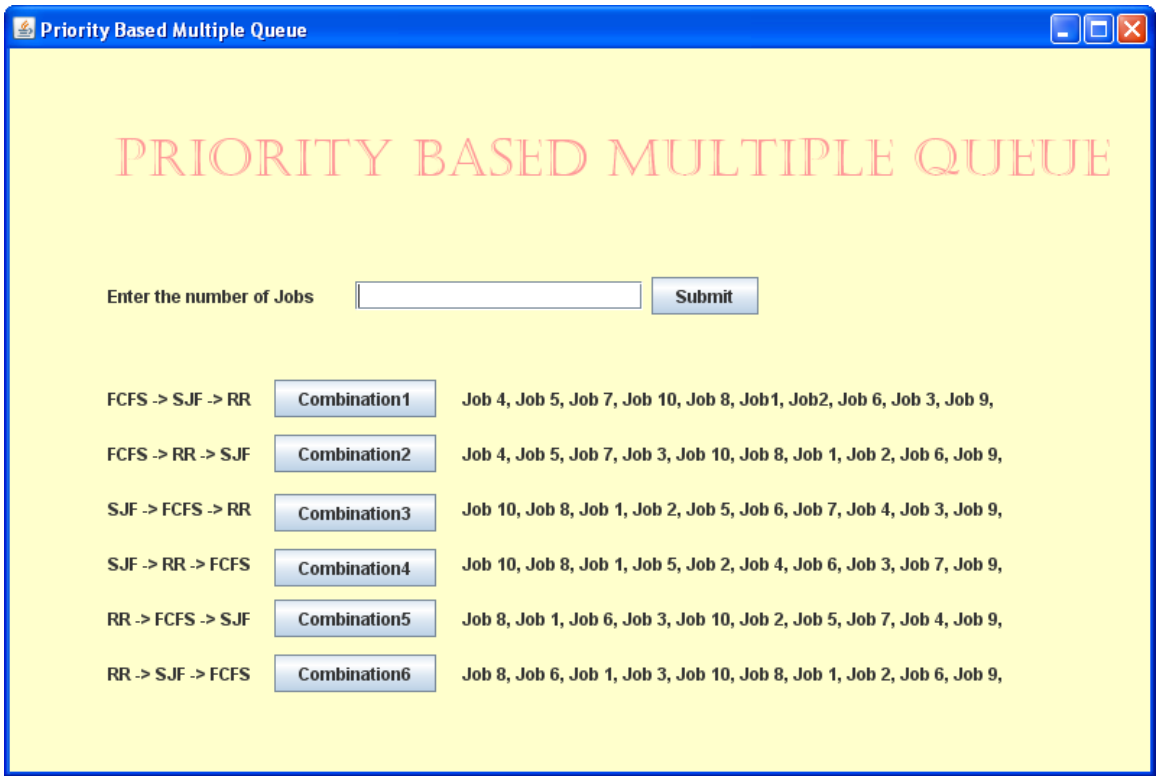


Figure 4.13 All Combination Prepared

Finally the entire job's sequence is shown in the figure-4.13 at each combination. The best combination is in this scenario is Combination 2 (FCFS -> RR -> SJF) for waiting

time and response time, and for the turnaround time Combination 1(FCFS -> SJF -> RR) is the best combination for this scenario. The summery of this case study is shown in following table.

Table 4.2: comparison of different combination of algorithms based on performance metric

Combination	Average waiting Time(millisecond)	Average turnaround time(millisecond)	Average response time(millisecond)
FCFS -> SJF -> RR	15.6	38.2	14.3
FCFS -> RR -> SJF	**15.1**	38.4	**14.1**
SJF -> FCFS -> RR	37.7	**25.8**	16.3
SJF -> RR -> FCFS	37.8	25.9	16.5
RR -> FCFS -> SJF	25.4	29.5	17.9
RR -> SJF -> FCFS	26.4	30.1	37.8

The selection of the combination is depends upon the scenario of the jobs. It means that, anyone of the combination can give the optimize result which depends upon the scenario of the jobs. The result may vary scenario to scenario. Finally best job sequence is submitted to the middleware for the purpose of computation of jobs.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This Thesis work primarily focuses on scheduling the jobs in an optimize manner. The user submits the numbers of jobs, which in this thesis, are created in a simulated environment. These jobs are arranged in different combinations of scheduling algorithms. Once the combinations are applied, the best combination for the given job is determined and is submitted to the middleware for computation.

The following graphs will help to conclude the thesis work very clearly. There are two type of graphs mentioned here. One is for comparison among different scheduling algorithms (First Come First Serve, Shortest Job First, and Round Robin) and another is for combination of these scheduling algorithms.

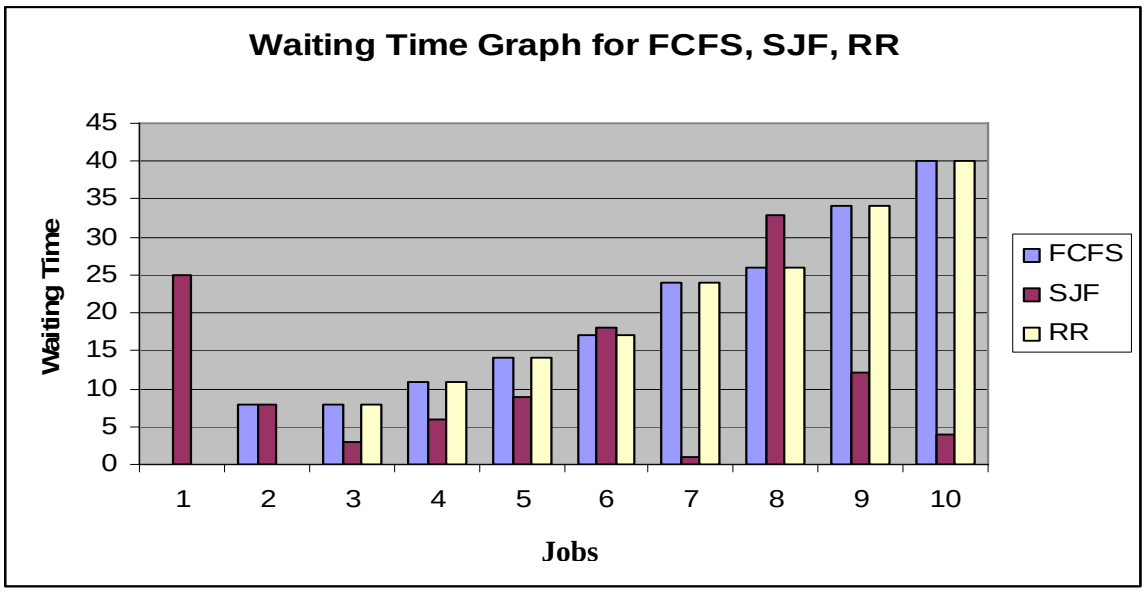


Figure 5.1 Waiting Time Graph for FCFS, SJF, and RR

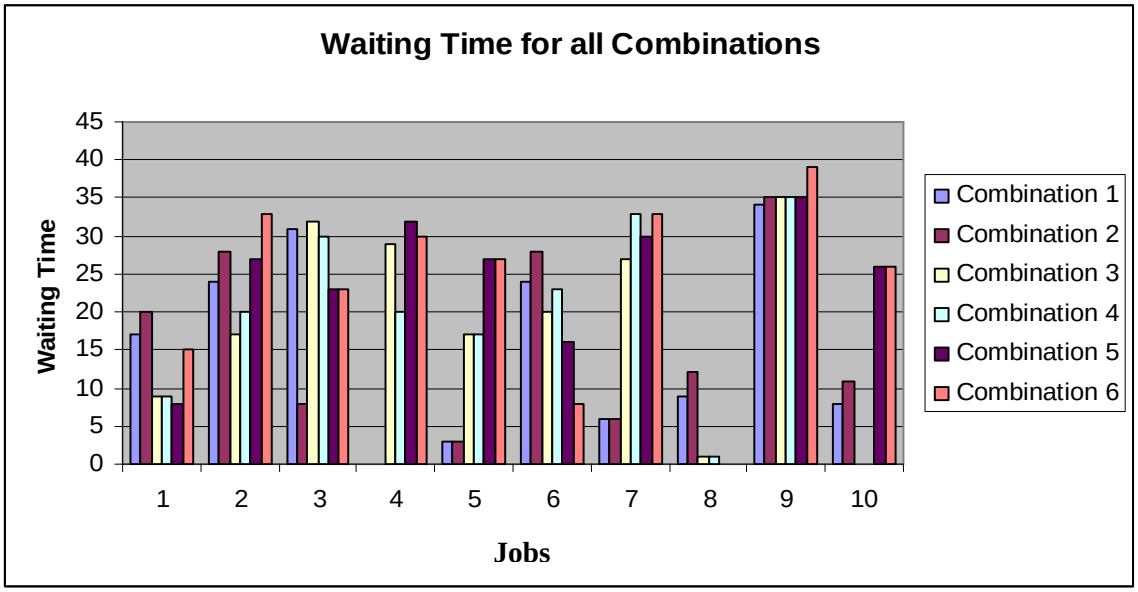


Figure 5.2 Waiting Time Graph for six Combinations

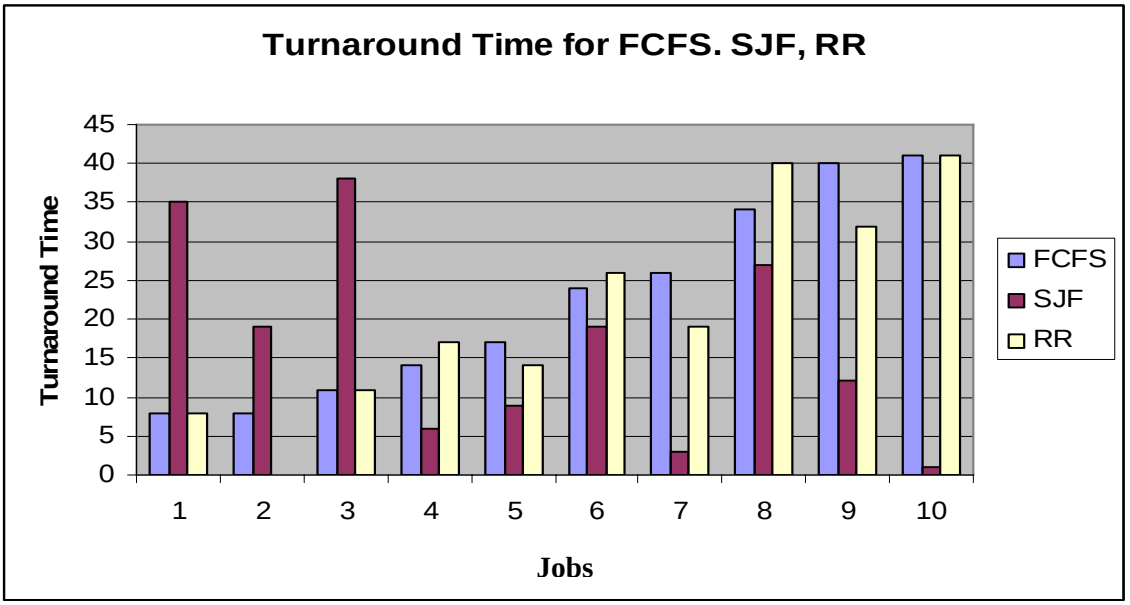


Figure 5.3 Turnaround Time Graph for FCFS, SJF, and RR

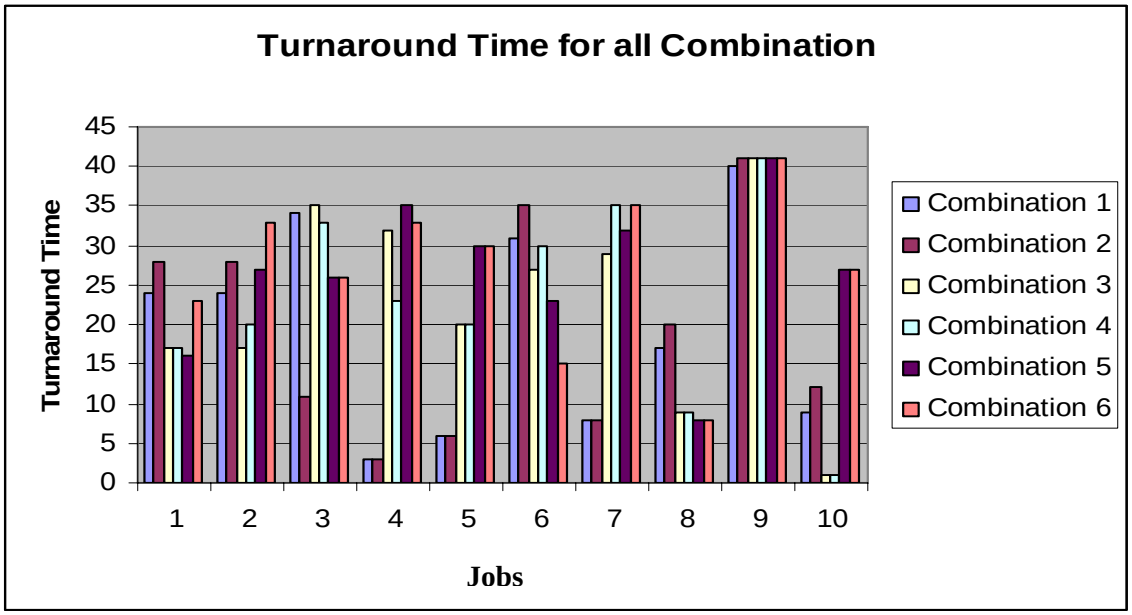


Figure 5.4 Turnaround Time Graph for six Combinations

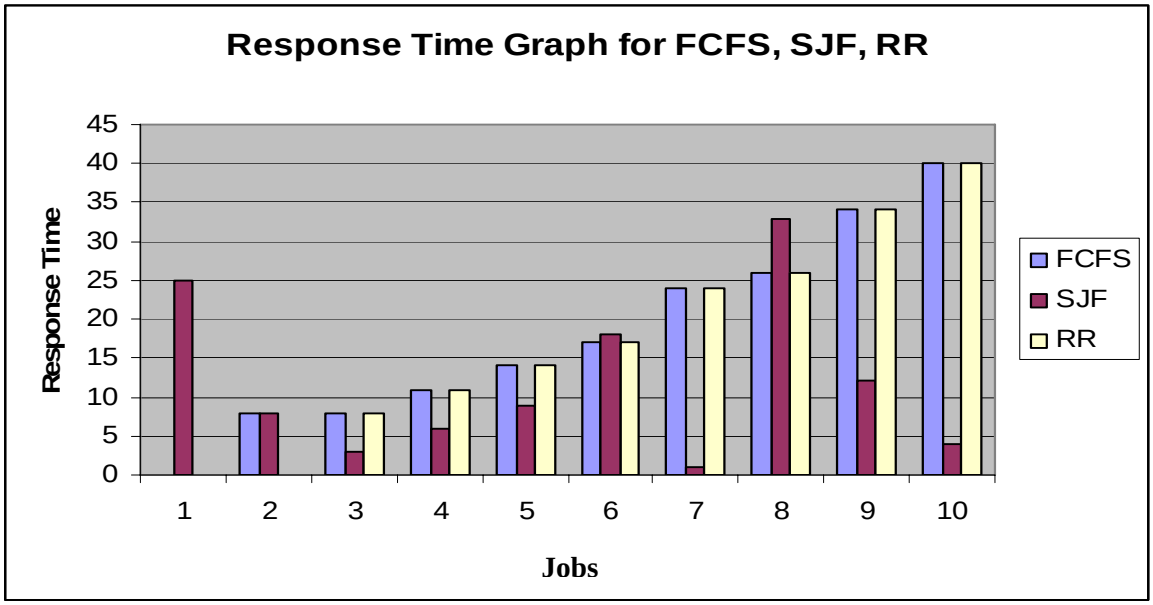


Figure 5.5 Response Time Graph for FCFS, SJF, and RR

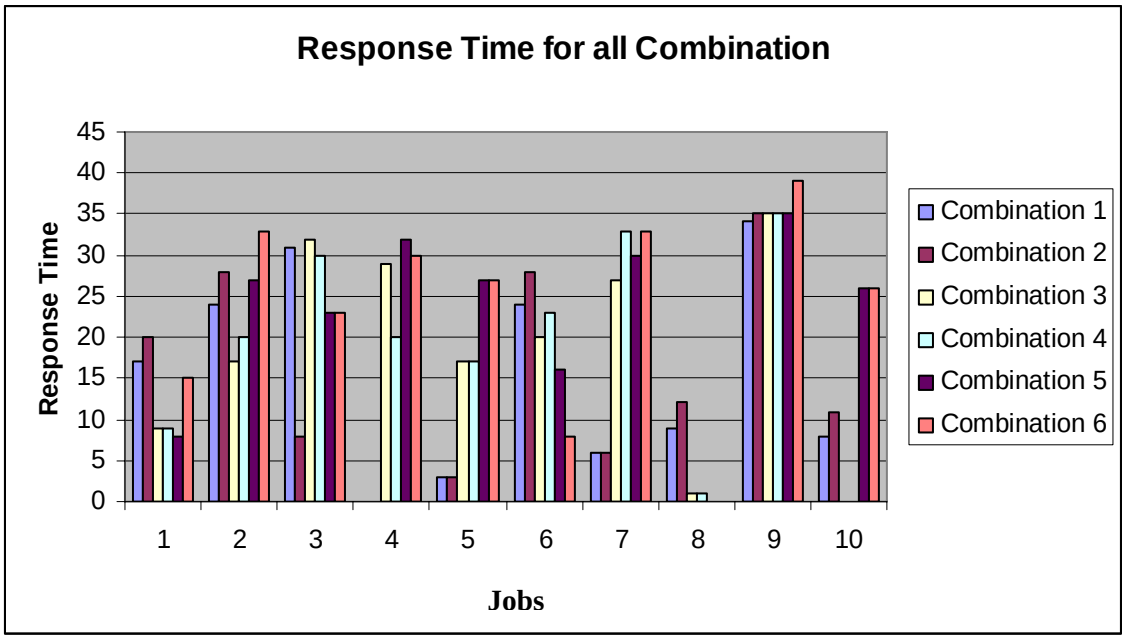


Figure 5.6 Response Time Graph for six Combinations

5.2 Summary of contributions

- This thesis demonstrates a new technique to select the best combination of job sequence from all the possible combinations using different combination of scheduling algorithms.
- Priority based multiple queue approach is used to solve the problem of selecting the best combination of job sequence. This increases the performance of a scheduler and in turn the Grid environment.
- Facilitates user to enter jobs according to his/her choice using a simulated environment. In the actual Grid environment, user may specify the priority of the job or scenario of the job.
- Priority based multiple queue scheduling algorithm uses first come first serve, shortest job first, round robin scheduling in order to find the best combination for a job sequence.

5.3 Future Scope

In future, the improvements can be done by also taking into account the dynamic behavior of the grid resources. There might be cases when resource required for a job, which is ready to be executed, is busy and job then has to be put on a waiting queue. In future waiting queue can also be implemented to take the above mentioned problem into account.

Secondly, the algorithm can be made more optimized to include more comparison parameters for comparing different queues. In this thesis only thee parameters are considered i.e. waiting time response time and throughput. Other performance parameters e.g. turnaround time, CPU utilization etc can also be added in future.

Thirdly, this thesis is carried out in a simulated environment. In future the work done here can be used in an actual Grid environment.

The work performed in this thesis can be used to optimize the scheduling algorithm for the Grid. This is because it provides the user friendly environment and choice to user to select his combination according to his choice.

A further extension to this work would be use to enhance the different scheduling algorithm to improve the performance, quality of service etc.

In the above scheduling algorithm, only CPU bound jobs are considered, in future the algorithm can be enhanced to include both CPU bound and Input bound jobs. Here we assume that, task scheduling is handling by local scheduler or resource management service, in future this part can be included in the algorithm to make it more efficient. In future, similar types of jobs group together and find the best combination of that group can be used to enhance the performance of the scheduling algorithm.

References

[1] Kleinrock, L. *MEMO on* Grid Computing, University of California, Los Angeles, 1969.

[2] 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.

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

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

[5] Foster, I. and Kesselman, C. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, San Francisco, CA, 677 pp, 1999.

[6] Ferreira, L., Bieberstein, N., Berstis, V., Armstrong, J., *Introduction to Grid Computing with Globus,* Redbook IBM Corp.

[7] Comparison of Grid Computing vs. Cluster Computing at, "http://www.jatit.org/research/introduction_grid_computing.htm".

[8] "CORBA: Common Object Request Broker Architecture, http://www.omg.org/gettingstarted/corbafaq.htm.

[9] Gregor von Laszewski, Manish Parashar, Snigdha Verma, Jarek Gawor, Kate Keahey, Nell Rehn**,** "A CORBA Commodity Grid Kit", The Applied Software Systems Laboratory, Department of Electrical and Computer Engineering.

[10] Domenico Talia, Paolo Trunfio, "A P2P Grid Services-Based Protocol: Design and Evaluation", University of Calabria.

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

[12] Yan Liu, "Grid Scheduling", *Department of Computer Science, University of Iowa.*

[13] N. Tonellotto, Information Science and Technologies Institute Italian National Research Council Italy, R. Yahyapour Institute for Robotics Research University of Dortmund Germany, "A Proposal for a Generic Grid Scheduling Architecture".

[14] 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.

[15] <http://www.cs.wisc.edu/condor/>

[16] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, "Operating System Principles Seventh Edition".

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

[18] T. Casavant, and J. Kuhl, *A Taxonomy of Scheduling in General-purpose Distributed Computing Systems*, in IEEE Trans. on Software Engineering Vol. 14, No.2, pp. 141--154, February 1988.

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

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

[21] 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.

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

[23] Eclipse Project, http://www.eclipse.org/eclipse/

[24] 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.

[25] 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.

[26] Buyya R (Ed.). High Performance Cluster Computing: Architectures and Systems, vol. 1. Prentice-Hall: Englewood Cliffs, NJ, 1999.

[27] Oram A (Ed.). Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly, 2001.

Papers Communicated/Published

1. Satyarth Kumar Singh, Damandeep Kaur, **"Priority Based Multiple Queue Scheduling Algorithm for Grid"**, 2nd IEEE Asia-Pacific, Service Computing Conference, on Dec 9-12, 2008, Jiaosi, Yilan Taiwan [Communicated]

Appendix –A

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 this 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));

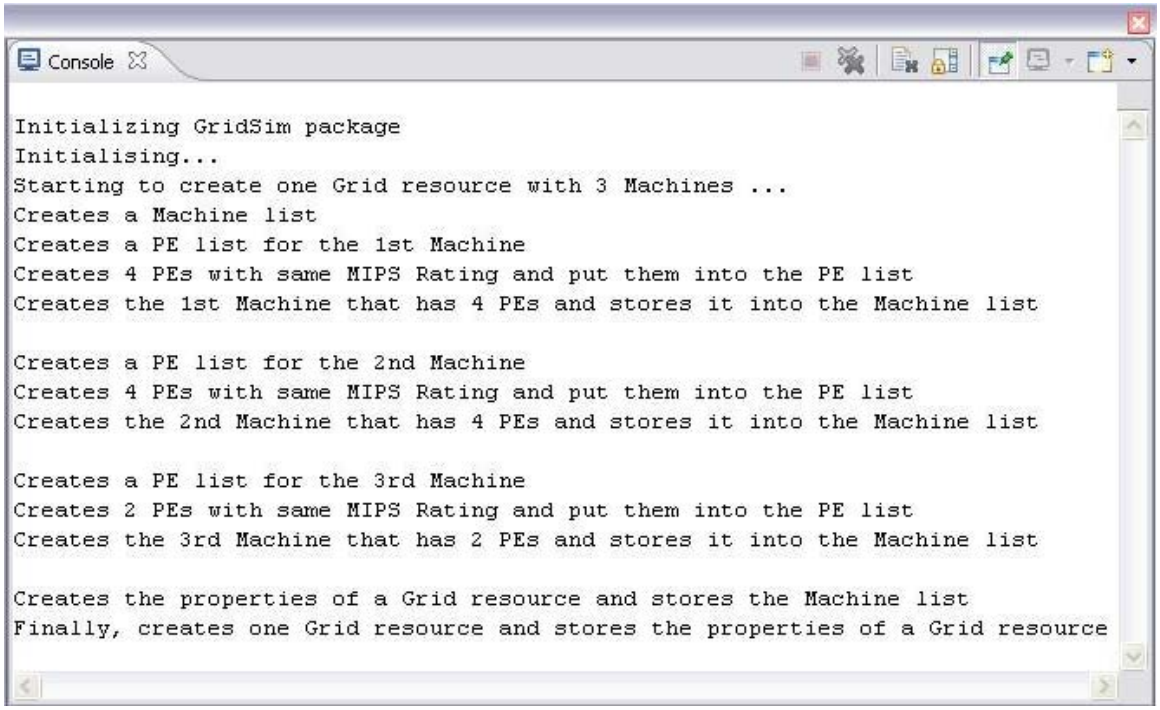
5. Repeat the steps from step 2 to step 4 to create additional number of machines.

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: Grid Resource creation in GridSim

Creating Grid User(s) in GridSim

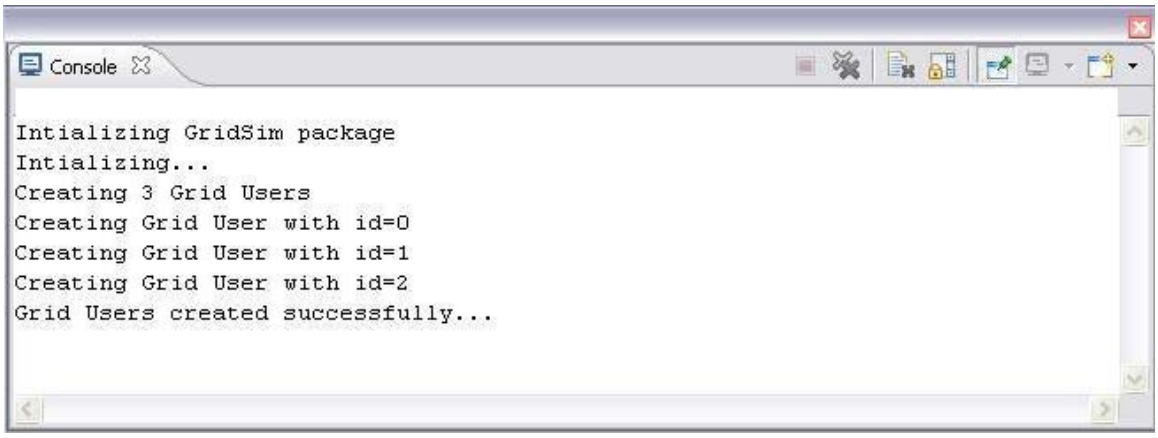
The steps to create Grid User(s) in GridSim are straightforward, but each User must have a unique ID. In the next step when the Gridlets (jobs) are created at that time we need to specify the User ID to which the Gridlet (job) belongs, therefore Grid user creation precedes Gridlet (job) creation. The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

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());

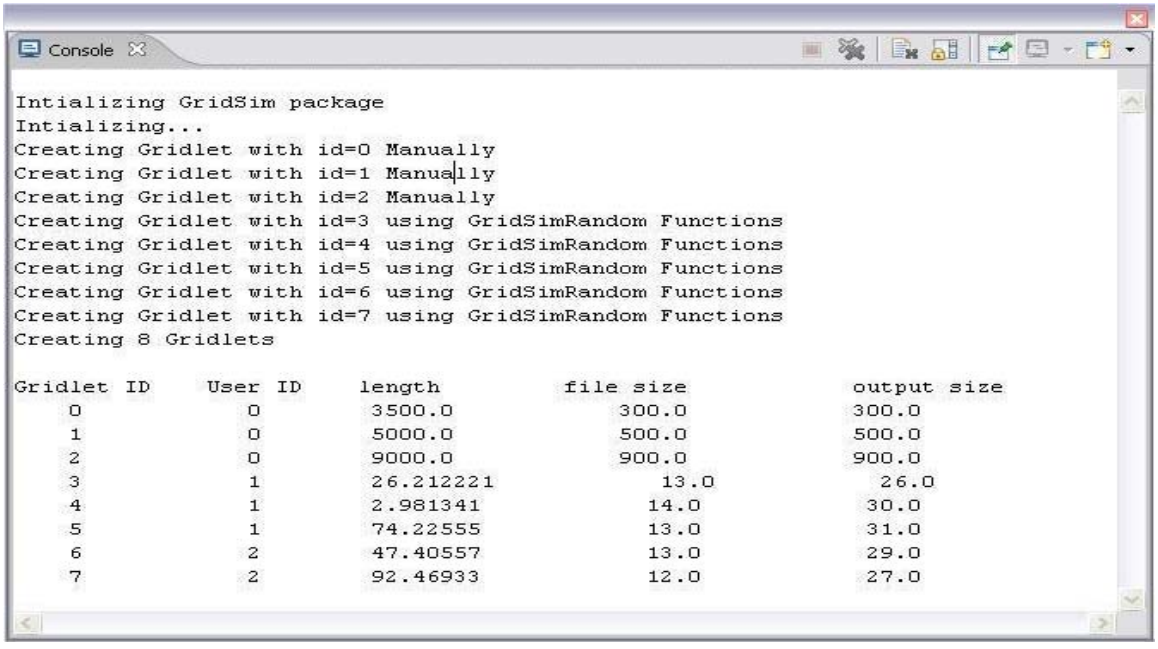
output_size = (long) GridSimRandom.real(250, min_range, max_range, random.nextDouble());

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: Gridlet creation in GridSim

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: Grid Entity creation in GridSim

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();

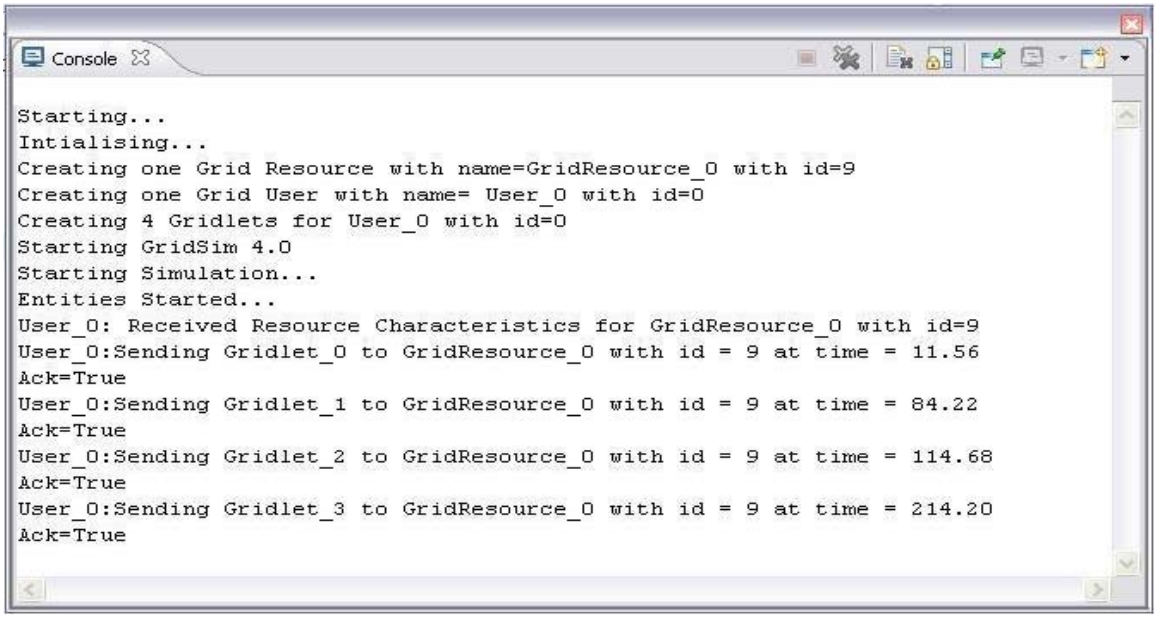
4. From the Gridlet list take a Gridlet at a time and then submit to the Grid Resource Entity with the Grid Resource ID to which we intend to submit. The last parameter in the method call below signifies that we need an acknowledgement for Gridlet successful submission.

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: Gridlet Submission in GridSim

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

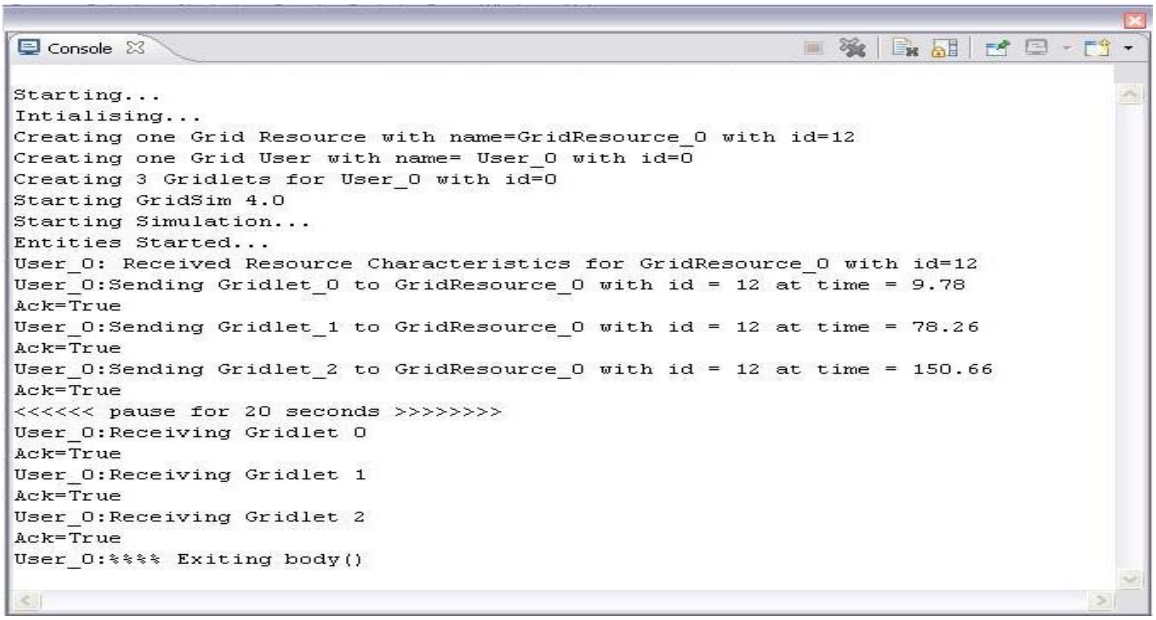
2. For each Gridlet submitted call the method

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.

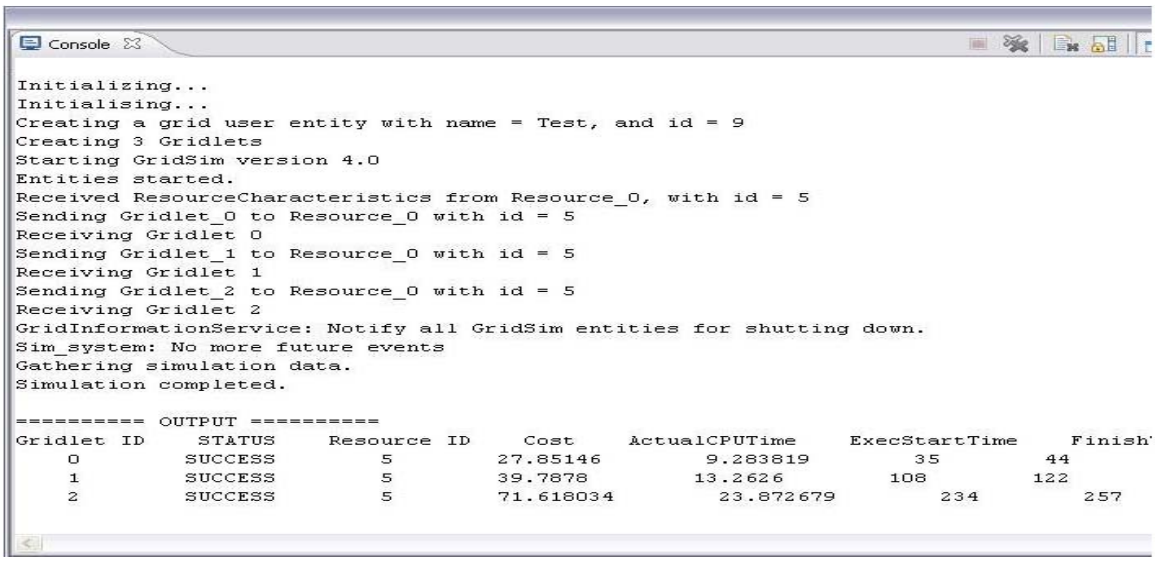


Screenshot: Gridlet Retrieval in GridSim

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()

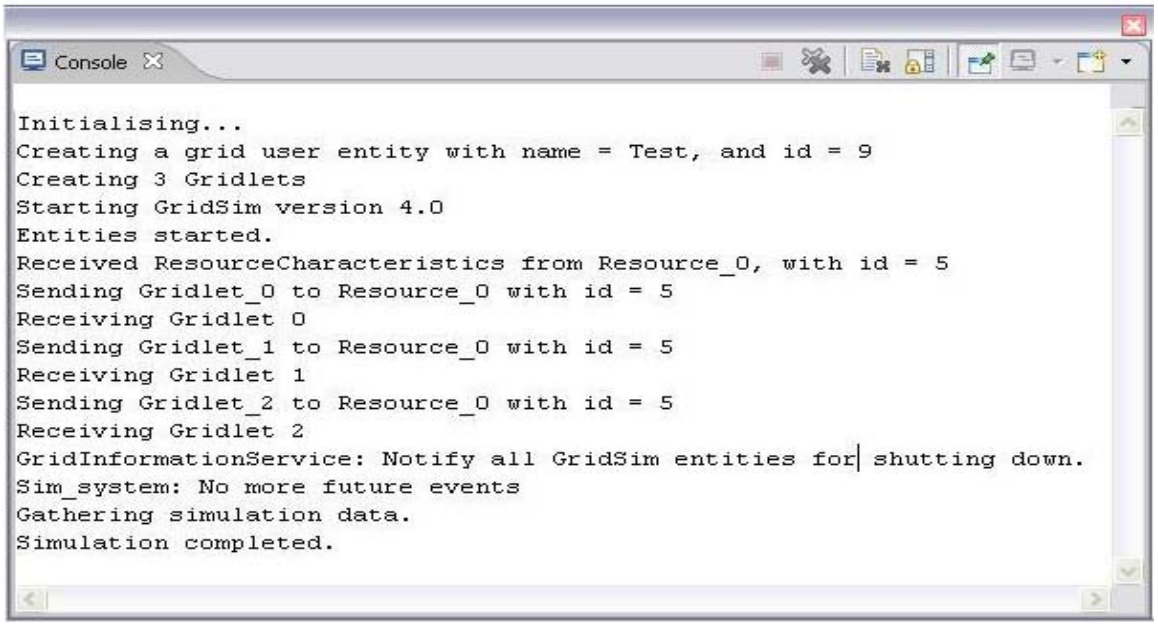


Screenshot: Gridlet Info Retrieval in GridSim

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: Shutting Down GridSim