

M-ant: MODIFIED ANT ALGORITHM FOR TASK SCHEDULING IN GRID COMPUTING

A Dissertation Submitted in partial fulfillment of the requirements
for the award of the degree of

**MASTER OF ENGINEERING
(Computer Technology & Applications)**

By

Saurabh Agrawal
College Roll No. 10/CTA/04
Delhi University Roll No. 8514

Under the guidance of

Dr. Goldie Gabrani



Department Of Computer Engineering
Delhi College of Engineering
Bawana Road, Delhi-110042
(University of Delhi)

June-2006

CERTIFICATE

This is to certify that dissertation entitled “**M-ant: Modified Ant Algorithm for Task Scheduling in Grid Computing**” which is submitted by **Saurabh Agrawal** in partial fulfillment of the requirement for the award of degree M.E. in Computer Technology & Applications to Delhi College of Engineering, Delhi is a record of the candidate work carried out by him.

(Project Guide)

Dr. Goldie Gabrani

Head of the Department

Department of Computer Engineering

Delhi College of Engineering

Bawana Road, Delhi

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extend my heartiest felt gratitude to everybody who helped me throughout the course of this project.

I would like to express my heartiest felt regards to **Dr. Goldie Gabrani**, Head of the Department, Department of Computer Engineering for the constant motivation and support during the duration of this project. It is my privilege and honor to have worked under the supervision. Her invaluable guidance and helpful discussions in every stage of this project really helped me in materializing this project. It is indeed difficult to put her contribution in few words.

I would also like to take this opportunity to present my sincere regards to my teachers viz. **Professor D. Roy Choudhury, Dr S. K. Saxena, Mr. Rajeev Kumar and Mrs. Rajni Jindal** for their support and encouragement.

I am thankful to my friends and classmates for their unconditional support and motivation during this project.

Saurabh Agrawal

M.E. (Computer Technology & Applications)

College Roll No. 10/CTA/04

Delhi University Roll No. 8514

Abstract

Load balanced task scheduling is very important problem in complex grid environment. Finding optimal schedules for such an environment is an NP-hard problem, and so heuristic approaches must be used. Ant-algorithm is a heuristic task scheduling algorithm which is distributable, scalable and fault tolerant. It uses the state prediction of the resources for scheduling which is necessary for effective utilization of resources.

The current ant-algorithm schedules the task based on possibilities of resources. The problem with this algorithm is that it schedules the task to low possibility resource even if high possibility resource is idle but also, If the tasks are always scheduled to the resource with high possibility, then the burden on the resource may be increased and the jobs may be kept waiting in the queue waiting for the resource to be free. In this project, I propose an enhanced ant-algorithm for task scheduling in grid which gives better throughput with a controlled cost. The simulation results of various scheduling algorithms are also compared. The results also show that the enhanced version works better than the ant-algorithm. The inclusion of price factor into the ant-algorithm makes this new scheduling algorithm more suitable for wide use.

Table of Contents

1. Introduction	8
1.1 Grid Computing.....	8
1.2 Introduction to GridSim.....	9
1.2.1 Features.....	10
1.3 Motivation.....	11
1.4 Objective.....	11
2. Task Scheduling	12
2.1 Grid Computing and Task Scheduling.....	12
2.1.1 Characteristics of Grid Task Scheduling.....	12
2.2 State Estimation.....	13
3. GridSim: Grid Modeling and Simulation Toolkit	15
3.1 System Architecture.....	15
3.1.1 SimJava Discrete Event Model.....	17
3.1.2 GridSim Entities.....	17
3.2 Models.....	20
3.2.1 Application Model.....	20
3.2.2 Interaction Protocols Model.....	20
3.2.3 Resource Model.....	22
3.3 GridSim Java Package Design.....	27
3.4 Building Simulations with GridSim.....	31
3.4.1 A Method for Simulating Application Scheduling.....	31
3.4.2 Economic Grid Resource Broker Simulation.....	34
3.5 Visual Modeler for Grid Modeling.....	36
3.5.1 Architecture of Visual Modeler.....	36
3.5.2 Design of Visual Modeler.....	37
3.5.3 Grid Computing Environment Simulation.....	40
4. Algorithm Design	43
4.1 Simulation Architecture.....	43
4.1.1 Grid Simulation Architecture.....	43
4.1.2 The Work Mode of Schedule Center.....	44
4.2 Scheduling Algorithm.....	45
4.2.1 Ant Algorithm.....	45
4.2.2 Ant Algorithm for Task Scheduling in Grid Computing.....	46
4.2.3 Problem with Ant Algorithm.....	47

4.3 Enhancement to the Algorithm.....	48
5. Implementation Details.....	49
5.1 Implementation.....	49
5.1.1 Input to the Simulation.....	50
6. Results and Discussion.....	56
6.1 Simulation Results.....	56
7. Conclusion and Future Work.....	63
References.....	64
Appendix: Source Code.....	66

List of Figures and Tables

Figures

Fig 1.1 A generic view of the World-Wide Grid computing environment.....	9
Fig 2.1 State estimation taxonomy.....	13
Fig 3.1 A Modular architecture for GridSim platform and components.....	16
Fig 3.2 Process flow of GridSim Simulation.....	33
Fig 3.3 The basic Model View Controller architecture.....	37
Fig 3.4 Class diagram that contains the model classes of VM and their relationships.....	38
Fig 3.5 Wizard dialog to create Grid users and resources.....	40
Fig 3.6 Resource dialog to view Grid resource properties.....	41
Fig 3.7 User dialog to view Grid user properties.....	42
Fig 4.1 Grid Simulation architecture.....	43
Fig 5.1 Number of users and Resources.....	52
Fig 5.2 Resource Information.....	52
Fig 5.3 Processor information of resource.....	53
Fig 5.4 Adding another resource.....	53
Fig 5.5 Editing User Information.....	54
Fig 5.6 User Information.....	54
Fig 5.7 Adding another User.....	55
Fig 5.8 Starting the Simulation.....	55
Fig 6.1 Results of different algorithms.....	56
Fig 6.2 Threshold Vs Execution time in Modified ant algorithm.....	57
Fig 6.3 Threshold Vs Total Cost in Modified ant algorithm.....	58
Fig 6.4(i) Graph showing scalability of Ant Algorithm(Execution Time).....	59
Fig 6.4(ii) Graph showing scalability of Ant Algorithm(Total Cost).....	60
Fig 6.5(i) Comparison Graphs of various algorithm results(Execution Time).....	61
Fig 6.5(ii) Comparison Graphs of various algorithm results(Total Cost).....	62

Tables

Table 5.1 Parameters of resources.....	50
Table 5.2 Task parameters of 20 users.....	51

Chapter 1

Introduction

1.1 Grid Computing

In the last decade, even though the computational capability and network performance have gone to a great extent, there are still problems in the fields of science, engineering, and business, which cannot be effectively dealt with using the current generation of supercomputers. In fact, due to their size and complexity, these problems are often resource (computational and data) intensive and consequently entail the use of a variety of heterogeneous resources that are not available in a single organization. The emergence of the Internet as well as the availability of powerful computers and high-speed network technologies as low-cost commodity components is rapidly changing the computing landscape and society. These technology opportunities have led to the possibility of using wide-area distributed computers for solving large-scale problems, leading to what is popularly known as Grid computing.

Grids enable the sharing, selection, and aggregation of a wide variety of resources including super computers, storage systems, data sources, and specialized devices that are geographically distributed and owned by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce.

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing [4].

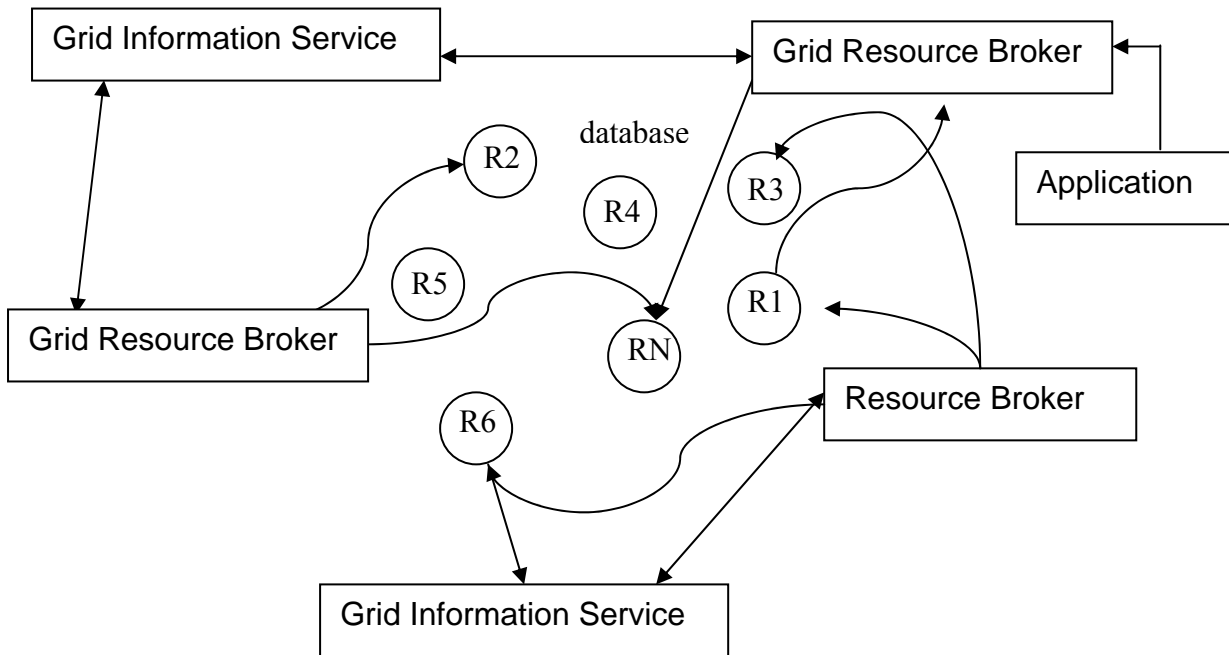


Fig. 1.1 A generic view of the World-Wide Grid computing environment.

1.2 Introduction to 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, Grids, and P2P networks. 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 *schedulers* in cluster systems focus on enhancing overall system performance and utility, as they are responsible for the whole system. In contrast, schedulers in Grid/P2P systems called *resource brokers*, focus on enhancing performance of a specific application in such a way that its end-users' requirements are met.

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.

The GridSim toolkit resource modeling facilities are used to simulate the World-Wide Grid resources managed as time or space-shared scheduling policies. The broker and user entities extend the GridSim class to inherit ability for communication with other entities. In GridSim, application tasks/jobs are modeled as Gridlet objects that contain all the information related to the job and its execution management details such as job length in MI (Million Instructions), disk I/O operations, input and output file sizes, and the job originator. The broker uses GridSim's job management protocols and services to map a Gridlet to a resource and manage it throughout its lifecycle.

1.2.1 Features

Salient features of the GridSim toolkit include the following[1].

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space-or time-shared mode.
- Resource capability can be defined (in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark).
- Resources can be located in any time zone.
- Weekends and holidays can be mapped depending on resource's local time to model non-Grid (local) workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.
- Network speed between resources can be specified.

1.3 Motivation

Resource management and task scheduling are very important and complex problems in grid computing. In grid environment, autonomy resources are linked through internet to form a huge virtual seamless environment. The resources in the grid are heterogeneous and the structure of the grid is changing almost all the time. In a grid, some resources may fail, some new resources enroll the grid, and some resources resume to work. So, it is necessary to do resource state prediction to get proper task scheduling.

Although various classification of task scheduling strategies exist, depending on the type of the grid, the scheduler organization and task scheduling strategy has to be chosen such that the resources in the grid are effectively utilized. Along with the scheduling organization, state estimation is also necessary for the dynamic grid. To achieve maximum resource utilization and high job throughput, re-scheduling of the jobs on different resources is also some times required.

Since the static algorithms like round robin algorithm, fastest processor-largest task first algorithm etc, are no longer suitable for the effective utilization of the grid, a good task scheduling method should be used which is distributable, scalable and fault tolerant

1.4 Objective

The aim of this project is to simulate and analyze a task scheduling algorithm which is based on well known ant-algorithm. The task scheduling algorithm should work better in the grid environment, in which the structure of the grid changes dynamically almost all the time. It should reduce the total execution time of jobs submitted to the grid, by effectively scheduling the jobs on to the appropriate resources in the grid. It should also reduce the cost of using the resources to execute the jobs. So, both the total execution time of the jobs and the cost of executing the jobs should be taken into consideration in scheduling the jobs on to the resources. If the cost of resources is not a factor (situations like, all the resources of our interest belong to the same organization), a modified form of ant-algorithm which can further reduce the total execution of the jobs.

Chapter 2

Task Scheduling on Grid

2.1 Grid Computing and Task Scheduling

Grid is an aggregation of a wide variety of resources that may include super computers, storage systems, data sources, and specialized devices that are geographically distributed and owned by different organizations. The term Grid is chosen as an analogy to the electric power Grid that provides consistent, pervasive, dependable, transparent access to electricity, irrespective of its source.

The technology used for solving large-scale computational and data intensive problems in science, engineering, and commerce using the Grid is Grid Computing.

The grid system is responsible for the execution of the job submitted to it. The advanced grid systems would include a “task scheduler” which automatically finds the most appropriate machine on which a given job is to run. This resource selection is very important in reducing the total execution time and cost of executing the tasks which depends on the task scheduling algorithm.

2.1.1 Characteristics of Grid Task Scheduling

In Grid environment autonomy resources are linked through Internet to form a huge virtual seamless environment. The transfer costs are very high at most time. So the tasks in Grid are usually coarse granularity tasks, for example, parameter sweeps application [4].

The resources in one Grid are heterogeneous, and the structure of the Grid is changing almost all the time: some resource gets failed, some new resource enrolls the Grid, and some resource resumes to work. So the dynamic of Grid is notable, good task scheduling method in Grid computing should be distributable, scalable, and fault tolerance.

It's very important to make state predictions for all resources. The predictive approaches use heuristic, pricing model or machine learning approaches. For example, Apples uses Network Weather Service heuristic model, punch system uses machine learning based prediction, Ninf system and NimrodG use pricing model.

The Grid computing environment can be viewed as a M/M/m queue system. Grid resources provide computing services, information services, and storage services, etc. The Grid clients ask for services. Everyone hopes good services. Many persons are studying the method of resource management and task scheduling.

2.2 State Estimation

The resources in Grid are heterogeneous, and the structure of the Grid is dynamic: some resource gets failed, some new resource enrolls the Grid, and some resource resumes to work. So, state estimation is also very important in scheduling the jobs in the grid.

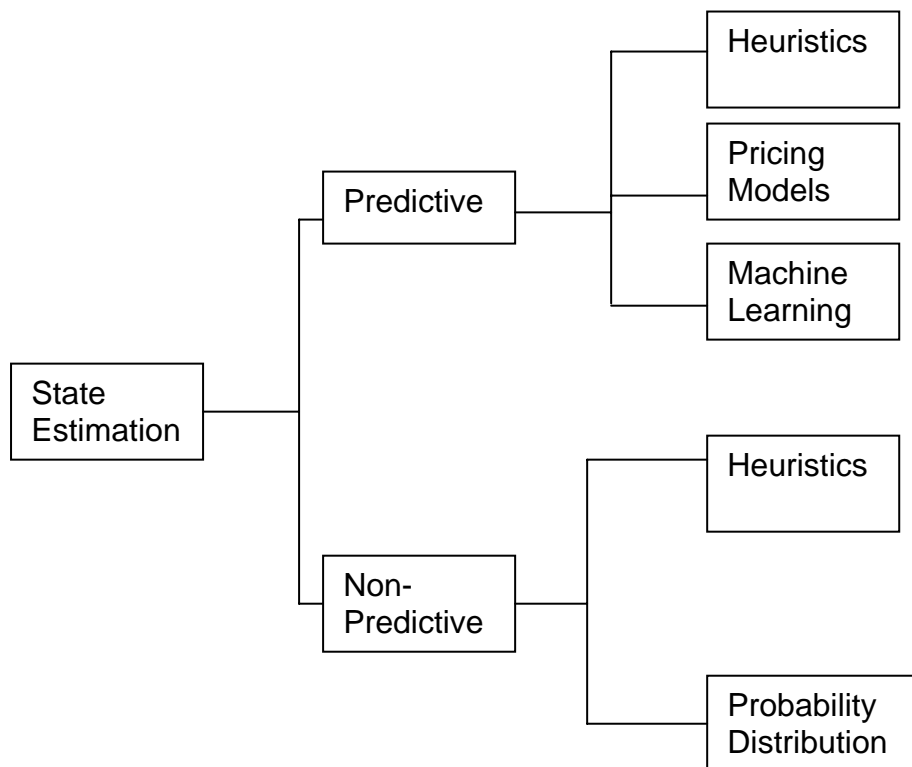


Fig. 2.1 State estimation taxonomy.

Non-predictive state estimation uses only the current job and resource status information since there is no need to take into account historical information. Non-predictive approaches use either heuristics based on job and resource characteristics or a probability distribution model based on an offline statistical analysis of expected job characteristics.

A predictive approach takes current and historical information such as previous runs of an application into account in order to estimate state. Predictive models use either heuristic, pricing model or machine learning approaches. In a heuristic approach, predefined rules are used to guide state estimation based on some expected behavior for Grid applications. In a pricing model approach, resources are bought and sold using market dynamics that take into account resource availability and resource demand. In machine learning, online or offline learning schemes are used to estimate the state.

Chapter 3

GridSim: Grid Modeling and Simulation Toolkit

The GridSim 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 domain distributed computing systems such as clusters and Grids. Application schedulers in the Grid environment, called resource brokers, perform resource discovery, selection, and aggregation of a diverse set of distributed resources for an individual user [6]. This means that each user has his or her own private resource broker and hence it can be targeted to optimize for the requirements and objectives of its owner. In contrast, schedulers, managing resources such as clusters in a single administrative domain, have complete control over the policy used for allocation of resources. This means that all users need to submit their jobs to the central scheduler, which can be targeted to perform global optimization such as higher system utilization and overall user satisfaction depending on resource allocation policy or optimize for high priority users.

3.1 System Architecture

We 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 3.1 [1]. The first layer is concerned with the scalable Java interface and the runtime machinery, called JVM (Java Virtual Machine), whose implementation is available for single and multiprocessor systems including clusters. 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. Recently, a distributed implementation of SimJava was also made available. 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.

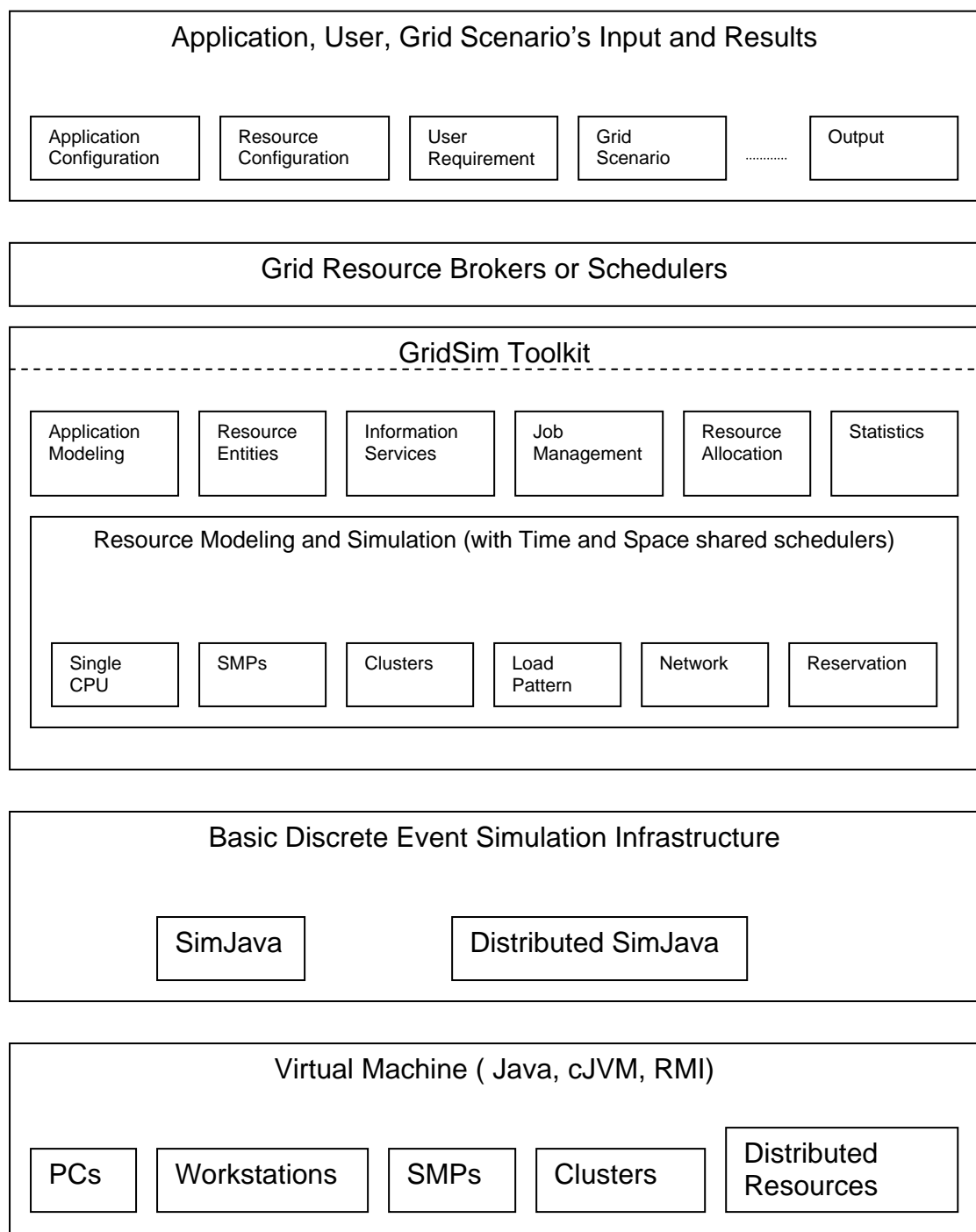


Figure 3.1 A Modular architecture for GridSim platform and components

3.1.1. SimJava Discrete Event Model

SimJava is a general purpose discrete event simulation package implemented in Java [13]. Simulations in SimJava contain a number of entities, each of which runs in parallel in its own thread. An entity's behaviour is encoded in Java using its `body()` method. Entities have access to a small number of simulation primitives:

- `sim schedule()` sends event objects to other entities via ports;
- `sim hold()` holds for some simulation time;
- `sim wait()` waits for an event object to arrive.

These features help in constructing a network of active entities that communicate by sending and receiving passive event objects efficiently.

The sequential discrete event simulation algorithm, in SimJava, is as follows. A central object Sim system maintains a timestamp ordered queue of future events. Initially all entities are created and their `body()` methods are put in run state. When an entity calls a simulation function, the Sim system object halts that entity's thread and places an event on the future queue to signify processing the function. When all entities have halted, Sim system pops the next event off the queue, advances the simulation time accordingly, and restarts entities as appropriate. This continues until no more events are generated. If the JVM supports native threads, then all entities starting at exactly the same simulation time may run concurrently.

3.1.2. GridSim Entities

GridSim supports entities for simulation of single processor and multiprocessor, heterogeneous resources that can be configured as time- or space-shared systems. It allows setting of the clock to different time zones to simulate geographic distribution of resources. It supports entities that simulate networks used for communication among resources. During simulation, GridSim creates a number of multi-threaded entities, each of which runs in parallel in its own thread. An entity's behavior needs to be simulated within its `body()` method, as dictated by SimJava.

A simulation environment needs to abstract all the entities and their time-dependent interactions in the real system. It needs to support the creation of user-defined time-dependent response functions

for the interacting entities. The response function can be a function of the past, current, or both states of entities. GridSim based simulations contain entities for the users, brokers, resources, information service, statistics, and network based I/O [1]. The design and implementation issues of these GridSim entities are discussed below.

3.1.2.1. User

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

- types of job created, e.g. job execution time, number of parametric replications, etc.;
- scheduling optimization strategy, e.g. minimization of cost, time, or both;
- activity rate, e.g. how often it creates new job;
- time zone; and
- absolute deadline and budget; or
- D- and B-factors, deadline and budget relaxation parameters, measured in the range [0, 1] express deadline and budget affordability of the user relative to the application processing requirements and available resources.

3.1.2.2. Broker

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

3.1.2.3. Resource

Each instance of the Resource entity represents a Grid resource. Each resource may differ from the rest of the resources with respect to the following characteristics:

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

The resource speed and the job execution time can be defined in terms of the ratings of standard benchmarks such as MIPS and SPEC. They can also be defined with respect to the standard machine. Upon obtaining the resource contact details from the Grid information service, brokers can query resources directly for their static and dynamic properties.

3.1.2.4. Grid Information Service

Providing resource registration services and keeping track of a list of resources available in the Grid. The brokers can query this for resource contact, configuration, and status information.

3.1.2.5. Input and output

The flow of information among the GridSim entities happens via their Input and Output entities. Every networked GridSim entity has I/O channels or ports, which are used for establishing a link between the entity and its own Input and Output entities. Note that the GridSim entity and its Input and Output entities are threaded entities, i.e. they have their own execution thread with `body()` method that handles events. The use of separate entities for input and output enables a networked entity to model full duplex and multi-user parallel communications. The support for buffered input and output channels associated with every GridSim entity provides a simple mechanism for an entity to communicate with other entities and at the same time enables modeling of the necessary communications delay transparently.

3.2 Models

3.2.1 Application Model

GridSim does not explicitly define any specific application model. It is up to the developers (of schedulers and resource brokers) to define them. Parallel application models such as process parallelism, Directed Acyclic Graphs (DAGs), divide and conquer etc., can also be modeled and simulated using GridSim [6].

In GridSim, each independent task may require varying processing time and input files size. Such tasks can be created and their requirements are defined through Gridlet objects. A Gridlet is a package that contains all the information related to the job and its execution management details such as job length expressed in MIPS, disk I/O operations, the size of input and output files, and the job originator. These basic parameters help in determining execution time, the time required to transport input and output files between users and remote resources, and returning the processed Gridlets back to the originator along with the results. The GridSim toolkit supports a wide range of Gridlet management protocols and services that allow schedulers to map a Gridlet to a resource and manage it throughout the life cycle.

3.2.2 Interaction Protocols Model

The protocols for interaction between GridSim entities are implemented using events. In GridSim, entities use events for both service request and service delivery. The events can be raised by any entity to be delivered immediately or with specified delay to other entities or itself. The events that are originated from the same entity are called internal events and those originated from the external entities are called external events. Entities can distinguish these events based on the source identification associated with them. The GridSim protocols are used for defining entity services. Depending on the service protocols, the GridSim events can be further classified into synchronous and asynchronous events. An event is called synchronous when the event source entity waits until the event destination entity performs all the actions associated with the event (i.e. the delivery of full service). An event is called asynchronous when the event source entity raises an event and continues with other activities without waiting for its completion. When the destination entity receives such events or service requests, it responds back with results by sending one or more

events, which can then take appropriate actions. It should be noted that external events could be synchronous or asynchronous, but internal events need to be raised as asynchronous events only to avoid deadlocks.

The GridSim entities (user, broker, resource, information service, statistics, shutdown, and report writer) send events to other entities to signify the request for service, to deliver results, or to raise internal actions. Note that GridSim implements core entities that simulate resource, information service, statistics, and shutdown services. These services are used to simulate a user with application, a broker for scheduling, and an optional report writer for creating statistical reports at the end of a simulation. The event source and destination entities must agree upon the protocols for service request and delivery. The protocols for interaction between the user-defined and core entities are pre-defined.

When GridSim starts, the resource entities register themselves with the Grid Information Service (GIS) entity, by sending events. This resource registration process is similar to GRIS (Grid Resource Information Server) registering with GIIS (Grid Index Information Server) in the Globus system. Depending on the user entity's request, the broker entity sends an event to the GIS entity, to signify a query for resource discovery. The GIS entity returns a list of registered resources and their contact details. The broker entity sends events to resources with a request for resource configuration and properties. They respond with dynamic information such as resources cost, capability, availability, load, and other configuration parameters. These events involving the GIS entity are synchronous in nature.

Depending on the resource selection and scheduling strategy, the broker entity places asynchronous events for resource entities in order to dispatch Gridlets for execution—the broker need not wait for a resource to complete the assigned work. When the Gridlet processing is finished, the resource entity updates the Gridlet status and processing time and sends it back to the broker by raising an event to signify its completion.

The GridSim resources use internal events to simulate resource behavior and resource allocation. The entity needs to be modeled in such a way that it is able to receive all events meant for it. However, it is up to the entity to decide on the associated actions. For example, in time-shared resource simulations internal events are scheduled to signify the completion time of a Gridlet, which has the smallest remaining processing time requirement. Meanwhile, if an external event arrives, it changes the share resource availability for each Gridlet, which means the most recently

scheduled event may not necessarily signify the completion of a Gridlet. The resource entity can discard such internal events without processing.

3.2.3 Resource Model

In the GridSim toolkit, we can create Processing Elements (PEs) with different speeds (measured in either MIPS or SPEC-like ratings). Then, one or more PEs can be put together to create a machine. Similarly, one or more machines can be put together to create a Grid resource. Thus, the resulting Grid resource can be a single processor, shared memory multiprocessors (SMP), or a distributed memory cluster of computers. These Grid resources can simulate time- or space-shared scheduling depending on the allocation policy. A single PE or SMP-type Grid resource is typically managed by time-shared operating systems that use a round-robin scheduling policy for multitasking. The distributed memory multiprocessing systems (such as clusters) are managed by queuing systems, called space-shared schedulers, that execute a Gridlet by running it on a dedicated PE when allocated. The space-shared systems use resource allocation policies such as first-come-first-served (FCFS), back filling, shortest-job-first-served (SJFS), and so on. It should also be noted that resource allocation within high-end SMPs could also be performed using the space-shared schedulers.

Multitasking and multiprocessing systems allow concurrently running tasks to share system resources such as processors, memory, storage, I/O, and network by scheduling their use for very short time intervals. A detailed simulation of scheduling tasks in the real systems would be complex and time consuming. Hence, in GridSim, we abstract these physical entities and simulate their behavior using process oriented, discrete event ‘interrupts’ with a time interval as large as the time required for the completion of the smallest remaining-time job. The GridSim resources can send, receive, or schedule events to simulate the execution of jobs. It schedules self-events for simulating resource allocation depending on the scheduling policy and the number of jobs in the queue or in execution.

Let us consider the following scenario to illustrate the simulation of Gridlets execution and scheduling within a GridSim resource. A resource consists of two shared or distributed memory PEs each with a MIPS rating of 1, for simplicity. Three Gridlets that represent jobs with processing requirements equivalent to 10, 8.5, and 9.5 MI (million instructions) arrive in simulation times 0, 4, and 7, respectively.

3.2.3.1 Simulation of Scheduling in Time-shared resources

The GridSim resource simulator uses internal events to simulate the execution and allocation of PEs' share to Gridlet jobs. When jobs arrive, time-shared systems start their execution immediately and share resources among all jobs. Whenever a new Gridlet job arrives, we update the processing time of existing Gridlets and then add this newly arrived job to the execution set. We schedule an internal event to be delivered at the earliest completion time of the smallest job in the execution set. It then waits for the arrival of events.

A complete algorithm for simulation of time-share scheduling and execution is shown below. If a newly arrived event happens to be an internal event whose tag number is the same as the most recently scheduled event, then it is recognized as a job completion event. Depending on the number of Gridlets in execution and the number of PEs in a resource, GridSim allocates the appropriate PE share to all Gridlets for the event duration using the algorithm shown below [1]. It should be noted that Gridlets sharing the same PE would get an equal amount of PE share. The completed Gridlet is sent back to its originator (broker or user) and removed from the execution set. GridSim schedules a new internal event to be delivered at the forecasted earliest completion time of the remaining Gridlets.

Algorithm: Time-Shared Grid Resource Event Handler()

1. Wait for an event

2. If the external and Gridlet arrival event, then:

BEGIN /*a new job arrived*/

- a. Allocate PE Share for Gridlets Processed so far
- b. Add arrived Gridlet to Execution Set
- c. Forecast completion time of all Gridlets in Execution Set
- d. Schedule an event to be delivered at the smallest completion time

END

3. If event is internal and its tag value is the same as the recently scheduled internal event tag,

BEGIN /*a job finish event*/

- a. Allocate PE Share for Gridlets Processed so far
- b. Update finished Gridlet's PE and Wall clock time parameters and send it back to the broker
- c. Remove finished Gridlet from the Execution Set and add to Finished Set
- d. Forecast completion time of all Gridlets in Execution Set

- e. Schedule an event to be delivered at the smallest completion time

END

4. Repeat the above steps until the end of simulation event is received

Algorithm: PE Share Allocation(Duration)

BEGIN

1. Identify total MI per PE for the duration and the number of PEs that process one extra Gridlet

$TotalMIperPE = MIPSratingOfOnePE() * Duration$

$MinNoOfGridletsPerPE = NoOfGridletsInExec / NoOfPEs$

$NoOfPEsRunningOneExtraGridlet = NoOfGridletsInExec \% NoOfPEs$

2. Identify maximum and minimum MI share that Gridlet get in the Duration

If($NoOfGridletsInExec \leq NoOfPEs$), then:

$MaxSharePerGridlet = MinSharePerGridlet = TotalMIperPE$

$MaxShareNoOfGridlets = NoOfGridletsInExec$

Else /* $NoOfGridletsInExec > NoOfPEs$ */

$MaxSharePerGridlet = TotalMIperPE / MinNoOfGridletsPerPE$

$MinSharePerGridlet = TotalMIperPE / (MinNoOfGridletsPerPE + 1)$

$MaxShareNoOfGridlets = (NoOfPEs - NoOfPEsRunningOneExtraGridlet) * MinNoOfGridletsPerPE$

END

When Gridlet1 arrives at time 0, it is mapped to PE1 and an internal event to be delivered at time 10 is scheduled since the predicted completion time is still 10. At time 4, Gridlet2 arrives and it is mapped to PE2. The completion time of Gridlet2 is predicted as 12.5 and the completion time of Gridlet1 is still 10 since both of them are executing on different PEs. A new internal event is scheduled, which will still be delivered at time 10. At time 7, Gridlet3 arrives, which is mapped to PE2. It shares the PE time with Gridlet2. At time 10, an internal event is delivered to the resource to signify the completion of Gridlet1, which is then sent back to the broker. At this moment, as the number of Gridlets is equal the number of PEs, they are mapped to different PEs. An internal event to be delivered at time 14 is scheduled to indicate the predicted completion time of Gridlet2. As simulation proceeds, an internal event is delivered at time 14 and Gridlet2 is sent back to the broker. An internal event to be delivered at time 18 is scheduled to indicate the predicted completion time of Gridlet3. Since there were no other Gridlets submitted before this time, the resource receives an internal interrupt at time 18, which signifies the completion of Gridlet3.

3.2.3.2 Simulation of Scheduling in Space-shared resources

The GridSim resource simulator uses internal events to simulate the execution and allocation of PEs to Gridlet jobs. When a job arrives, space-shared systems start its execution immediately if there is a free PE available, otherwise, it is queued. During the Gridlet assignment, job-processing time is delivered to signify the completion of the scheduled Gridlet job. The resource simulator then frees the PE allocated determined and the event is scheduled for delivery at the end of the execution time. Whenever a Gridlet job finishes, an internal event is to it and checks if there are any other jobs waiting in the queue. If there are jobs waiting in the queue, then it selects a suitable job depending on the policy and assigns it to the PE which is free.

A complete algorithm for simulation of space-share scheduling and execution is shown below.

Algorithm: Space-Shared Grid Resource Event Handler()

1. Wait for an event and Identify Type of Event received

2. If it external and Gridlet arrival event, then:

BEGIN /* a new job arrived */

- • If the number of Gridlets in execution are less than the number of PEs in the resource, then Allocate PE to the Gridlet() /* It should schedule an Gridlet completion event */
- • If not, Add Gridlet to the Gridlet Submitted Queue

END

3. If event is internal and its tag value is the same recently scheduled internal event tag,

BEGIN /* a job finish event */

- Update finished Gridlet's PE and Wall clock time parameters and send it back to the broker.
- Set the status of PE to FREE.
- Remove finished Gridlet from the Execution Set and add to the Finished Set.
- If Gridlet Submitted Queue has Gridlets in waiting, then

Choose the Gridlet to be Processed() /* e.g., first one in Q if FCFS policy is used */

Allocate PE to the Gridlet() /* It should schedule a Gridlet completion event */

END

4. Repeat above steps until end of simulation event is received

If a newly arrived event happens to be an internal event whose tag number is the same as the most recently scheduled event, then it is recognized as a Gridlet completion event. If there are Gridlets in the submission queue, then depending on the allocation policy (e.g. the first Gridlet in the queue if

FCFS policy is used), GridSim selects a suitable Gridlet from the queue and assigns it to the PE or a suitable PE if more than one PE is free. Illustration of the allocation of PEs to Gridlets is as follows.

Algorithm: Allocate PE to the Gridlet(Gridlet gl)

BEGIN

1. Identify a suitable Machine with Free PE
2. Identify a suitable PE in the machine and Assign to the Gridlet
3. Set Status of the Allocated PE to BUSY
4. Determine the Completion Time of Gridlet and Set and internal event to be delivered at the completion time

END

The completed Gridlet is sent back to its originator (broker or user) and removed from the execution set. GridSim schedules a new internal event to be delivered at the completion time of the scheduled Gridlet job.

When Gridlet1 arrives at time 0, it is mapped to PE1 and an internal event to be delivered at time 10 is scheduled since the predicted completion time is still 10. At time 4, Gridlet2 arrives and it is mapped to PE2. The completion time of Gridlet2 is predicted as 12.5 and the completion time of Gridlet1 is still

10 since both of them are executing on different PEs. A new internal event to be delivered at time 12.5 is scheduled to signify the completion of Gridlet2. At time 7, Gridlet3 arrives. Since there is no free PE available on the resource, it is put into the queue. The simulation continues, i.e. the GridSim resource waits for the arrival of a new event. At time 10 a new event is delivered which happens to signify the completion of Gridlet1, which is then sent back to the broker. It then checks to see if there are any Gridlets waiting in the queue and chooses a suitable Gridlet (in this case Gridlet2, based on FCFS policy) and assigns the available PE to it. An internal event to be delivered at time 19.5 is scheduled to indicate the completion time of Gridlet3 and then waits for the arrival of new events. A new event is delivered at the simulation time 12.5, which signifies the completion of Gridlet2, which is then sent back to the broker. There is no Gridlet waiting in the queue, so it proceeds without scheduling any events and waits for the arrival of the next event. A new internal event arrives at the simulation time 19.5, which signifies the completion of Gridlet3.

This process continues until resources receive an external event indicating the termination of simulation.

For every Grid resource, the non-Grid (local) workload is estimated based on typically observed load conditions depending on the time zone of the resource. The network communication speed between a user and the resources is defined in terms of a data transfer speed (baud rate).

3.3 GridSim Java package design

The GridSim package implements the following classes [14].

class gridsim.Input: this class extends the `eduni.simjava.SimEntity` class. It defines a port through which a simulation entity receives data from the simulated network. It maintains an event queue to serialize the data-in-flow and delivers to its parent entity. Simultaneous inputs can be modeled using multiple instances of this class.

class gridsim.Output: this class is very similar to the `gridsim.Input` class and it defines a port through which a simulation entity sends data to the simulated network. It maintains an event queue to serialize the data-out-flow and delivers to the destination entity. Simultaneous outputs can be modeled by using multiple instances of this class.

class gridsim.GridSim: this is the main class of the GridSim package that must be extended by GridSim entities. It inherits event management and threaded entity features from the `eduni.simjava.SimEntity` class. The GridSim class adds networking and event delivery features, which allow synchronous or asynchronous communication for service access or delivery. All classes that extend the GridSim class must implement a method called `body()`, which is automatically invoked since it is expected to be responsible for simulating entity behavior.

The entities that extend the GridSim class can be instantiated with or without networked I/O ports. A networked GridSim entity gains communication capability via the objects of GridSim's I/O entity classes, `gridsim.Input` and `gridsim.Output` classes. Each I/O entity will have a unique name assuming each GridSim entity that the user creates has a unique name. For example, a resource entity with the name 'Resource2' will have an input entity whose name is prefixed with 'Input_', making the input entity's full name 'Input_Resource2', which is expected to be unique. The I/O

entities are concurrent entities, but they are visible within the GridSim entity and are able to communicate with other GridSim entities by sending messages.

The GridSim class supports methods for simulation initialization, management, and flow control. The GridSim environment must be initialized to set-up the simulation environment before creating any other GridSim entities at the user level. This method also prepares the system for simulation by creating three GridSim internal entities—GridInformationService, GridSimShutdown, and GridStatistics. The GridInformationService entity simulates the directory that dynamically keeps a list of resources available in the Grid. The GridSimShutdown entity helps in wrapping up a simulation by systematically closing all the opened GridSim entities. The GridStatistics entity provides standard services during the simulation to accumulate statistical data. Invoking the GridSim.Start () method starts the Grid simulation. All the resource and user entities must be instantiated in between invoking the above two methods.

The GridSim class supports static methods for sending and receiving messages between entities directly or via network entities, managing and accessing handles to various GridSim core entities, and recording statistics.

classgridsim.PE: this is used to represent CPU/PE, the capability of which is defined in terms of MIPS rating.

classgridsim.PEList: maintains a list of PEs that make up a machine.

classgridsim.Machine: represents a uniprocessor or shared memory multiprocessor machine.

classgridsim.MachineList: an instance of this class simulates a collection of machines. It is up to the GridSim users to define the connectivity among the machines in a collection. Therefore, this class can be instantiated to model simple LAN to cluster to WAN.

classgridsim.ResourceCharacteristics: this represents static properties of a resource such as resource architecture, OS, management policy (time-or space-shared), cost, and time zone at which the resource is located along resource configuration.

classgridsim.GridResource: extends the GridSim class and gains communication and concurrent entity capability. An instance of this class simulates a resource with properties defined in an object of the gridsim.ResourceCharacteristicsclass. The process of creating a Grid resource is as follows:

first create PE objects with a suitable MIPS/SPEC rating, second assemble them together to create a machine. Finally, group one or more objects of the machine to form a resource. A resource having a single machine with one or more PEs is managed as a time-shared system using a round-robin scheduling algorithm. A resource with multiple machines is treated as a distributed memory cluster and is managed as a space-shared system using FCFS scheduling policy or its variants.

classgridsim.GridSimStandardPE: defines MIPS rating for a standard PE or enables the users to define their own MIPS/SPEC rating for a standard PE. This value can be used for creating PEs with relative MIPS/SPEC rating for GridSim resources and creating Gridlets with relative processing requirements.

classgridsim.ResourceCalendar: this class implements a mechanism to support modeling a local load on Grid resources that may vary according to the time zone, time, weekends, and holidays.

classgridsim.GridInformationService: a GridSim entity that provides Grid resource registration, indexing and discovery services. The Grid resources register their readiness to process Gridlets by registering themselves with this entity. GridSim entities such as the resource broker can contact this entity for resource discovery service, which returns a list of registered resource entities and their contact address. For example, scheduling entities use this service for resource discovery.

classgridsim.Gridlet: this class acts as job package that contains job length in MI, the length of input and output data in bytes, execution start and end time, and the originator of the job. Individual users model their application by creating Gridlets for processing them on Grid resources assigned by scheduling entities (resource brokers).

classgridsim.GridletList: can be used to maintain a list of Gridlets and support methods for organizing them.

classgridsim.GridSimTags: contains various static command tags that indicate a type of action that needs to be undertaken by GridSim entities when they receive events. The different types of tags supported in GridSim along with comments indicating possible purpose are shown in Figure below.

classgridsim.ResGridlet: represents a Gridlet submitted to the resource for processing. It contains a Gridlet object along with its arrival time and the ID of the machine and the PE allocated to it. It acts

as a placeholder for maintaining the amount of resource share allocated at various times for simulating time-shared scheduling using internal events.

classgridsim.GridStatistics: this is a GridSim entity that records statistical data reported by other entities. It stores data objects with their label and timestamp. At the end of simulation, the user-defined report-writer entity can query recorded statistics of interest for report generation.

classgridsim.Accumulator: the objects of this class provide a placeholder for maintaining statistical values of a series of data added to it. It can be queried for mean, sum, standard deviation, and the largest and smallest values in the data series.

classgridsim.GridSimShutdown: this is a GridSim entity that waits for termination of all user entities to determine the end of simulation. It then signals the user-defined report-writer entity to interact with the GridStatistics entity to generate a report. Finally, it signals the end of simulation to other GridSim core entities.

classgridsim.GridSimRandom: this class provides static methods for incorporating randomness in data used for any simulation. Any predicted/estimated data, e.g. number of Gridlets used by an experiment, execution time and output size of a Gridlet etc., need to be mapped to real-world data by introducing randomness to reflect the uncertainty that is present in the prediction/estimation process and the randomness that exists in the nature itself. The execution time of a Gridlet on a particular resource, for example, can vary depending on the local load, which is not covered by the scope of the GridSim to simulate.

Global tags in the GridSim package:

```
public class GridSimTags {
public static final double SCHEDULE_NOW = 0.0; // 0.0 indicates NO delay
public static final int END_OF_SIMULATION = -1;
public static final int INSIGNIFICANT = 0; // ignore tag
public static final int EXPERIMENT = 1; // User <-> Broker
public static final int REGISTER_RESOURCE = 2; // GIS -> ResourceEntity
public static final int RESOURCE_LIST = 3; // GIS <-> Broker
public static final int RESOURCE_CHARACTERISTICS = 4; // Broker <-> ResourceEntity
public static final int RESOURCE_DYNAMICS = 5; // Broker <-> ResourceEntity
public static final int GRIDLET_SUBMIT = 6; // Broker -> ResourceEntity
public static final int GRIDLET_RETURN = 7; // Broker <- ResourceEntity
public static final int GRIDLET_STATUS = 8; // Broker <-> ResourceEntity
public static final int RECORD_STATISTICS = 9; // Entity -> GridStatistics
public static final int RETURN_STAT_LIST = 10; // Entity <- GridStatistics
public static final int RETURN_ACC_STATISTICS_BY_CATEGORY = 11;
public static final int DEFAULT_BAUD_RATE = 9600; // Default Baud Rate for entities
}
```

3.4 Building Simulations with GridSim

To simulate Grid resource brokers using the GridSim toolkit, the developers need to create new entities that exhibit the behavior of Grid users and scheduling systems. The user-defined entities extend the GridSim base class to inherit the properties of concurrent entities capable of communicating with other entities using events. The detailed steps involved in modeling resources and applications, and simulating brokers using the GridSim toolkit are discussed below.

3.4.1 A method for Simulating Application Scheduling

In this section we present high-level steps, to demonstrate how GridSim can be used to simulate a Grid environment to analyze scheduling algorithms.

- First, we need to create Grid resources of different capabilities and configurations (a single or multiprocessor with time/space-shared resource manager). We also need to create users with different requirements (application and quality of service requirements).
- Second, we need to model applications by creating a number of Gridlets and define all parameters associated with jobs. The Gridlets need to be grouped together depending on the application model.
- Then, we need to create a GridSim user entity that creates and interacts with the resource broker scheduling entity to coordinate execution experiment. It can also directly interact with GIS and resource entities for Grid information and submitting or receiving processed Gridlets. However, for modularity sake, we encourage the implementation of a separate resource broker entity by extending the GridSim class.
- Finally, we need to implement a resource broker entity that performs application scheduling on Grid resources. First, it accesses the GIS, and then inquires for resource capability including cost. Depending on the processing requirements, it develops a schedule for assigning Gridlets to resources and coordinates the execution. The scheduling policies can be systems-centric or user-centric.

Figure 3.2 depicts the flowchart of GridSim Simulation used the project.

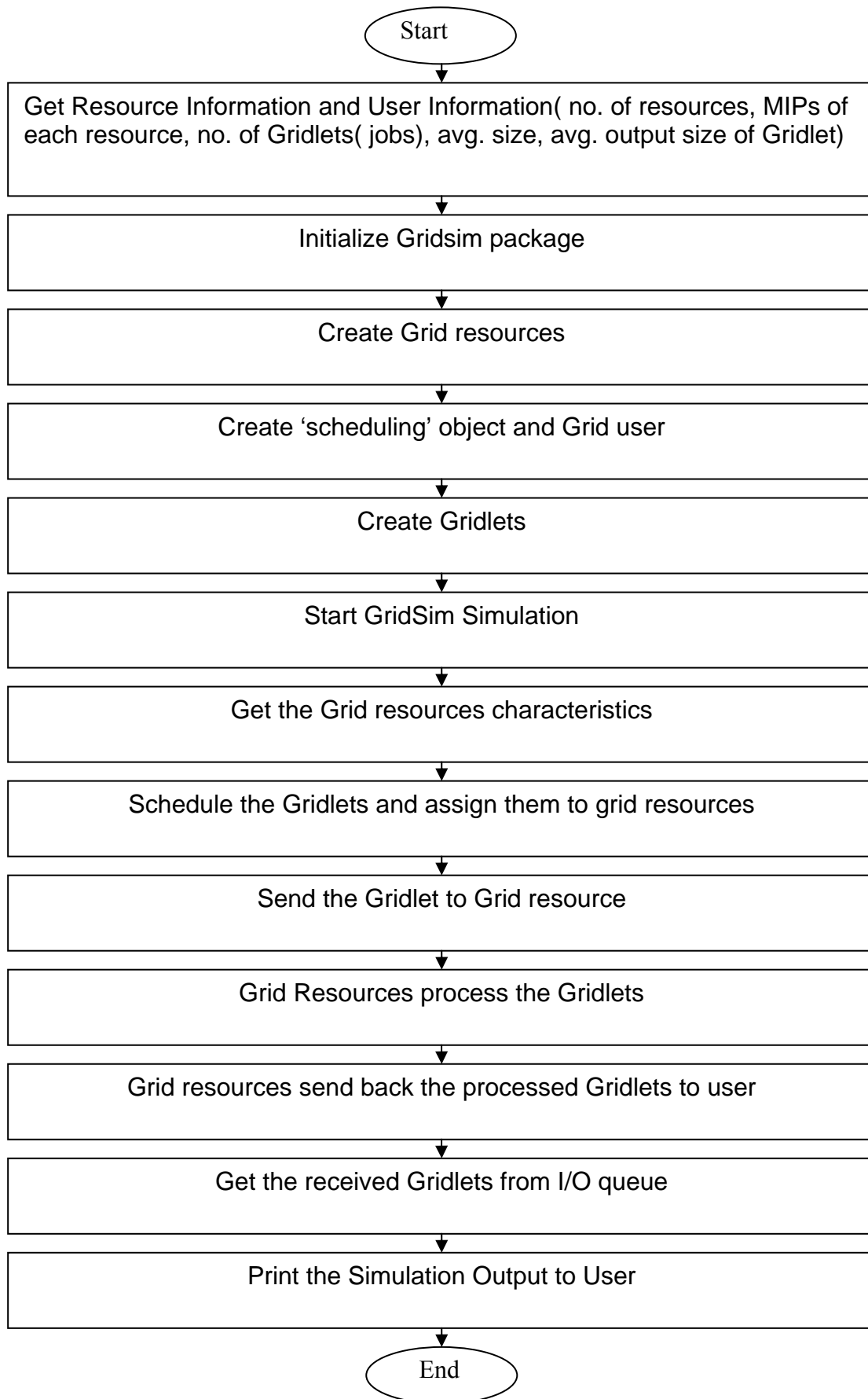


Figure 3.2 Process flow of GridSim Simulation

3.4.2 Economic Grid Resource Broker Simulation

We used the GridSim toolkit to simulate the Grid environment and a Nimrod-G like deadline and budget constrained scheduling system called the economic Grid resource broker. The simulated Grid environment contains multiple resources and user entities with different requirements [1]. The users create an experiment that contains application specification (a set of Gridlets that represent application jobs with different processing) and quality of service requirements (deadline and budget constraints with optimization strategy). We created two entities that simulate the users and the brokers by extending the GridSim class. When simulated, each user entity having its own application and quality of service requirements creates its own instance of the broker entity for scheduling Gridlets on resources.

3.4.2.1 Broker Architecture

The key components of the broker are the experiment interface, resource discovery and trading, the scheduling flow manager backed with scheduling heuristics and algorithms, the Gridlets dispatcher, and the Gridlets receptor. The following high-level steps describe the functionality of the broker components and their interaction.

1. The user entity creates an experiment that contains an application description (a list of Gridlets to be processed) and user requirements to the broker via the experiment interface.
2. The broker resource discovery and trading module interacts with the GridSim GIS entity to identify contact information of resources and then interacts with resources to establish their configuration and access cost. It creates a broker resource list that acts as a placeholder for maintaining resource properties, a list of Gridlets committed for execution on the resource, and the resource performance data as predicted through the measure and extrapolation methodology.
3. The scheduling flow manager selects an appropriate scheduling algorithm for mapping Gridlets to resources depending on the user requirements. Gridlets that are mapped to specific resource are added to the Gridlets list in the broker resource.
4. For each of the resources, the dispatcher selects the number of Gridlets that can be staged for execution according to the usage policy to avoid overloading resources with single user jobs.
5. The dispatcher then submits Gridlets to resources using the GridSim asynchronous service.

6. When the Gridlet processing completes, the resource returns it to the broker's Gridlet receptor module, which then measures and updates the runtime parameter, resource or MI share available to the user. It aids in predicting the job consumption rate for making scheduling decisions.
7. Steps 3–6 continue until all the Gridlets are processed or the broker exceeds deadline or budget limits. The broker then returns updated experimental data along with processed Gridlets back to the user entity.

The Grid broker package implements the following key classes.

class Experiment: acts as a placeholder for representing simulation experiment configuration that includes synthesized application (a set of Gridlets stored in GridletList) and user requirements such as D and B factors or deadline and budget constraints, and optimization strategy. It provides methods for updating and querying the experiment parameters and status. The user entity invokes the broker entity and passes its requirements via the experiment object. On receiving an experiment from its user, the broker schedules Gridlets according to the optimization policy set for the experiment.

class UserEntity: a GridSim entity that simulates the user. It invokes the broker and passes the user requirements. When it receives the results of application processing, it records parameters of interest with the gridsim.Statistics entity. When it has no more processing requirements, it sends the END OF SIMULATION event to the gridsim.GridSimShutdown entity.

class Broker: a GridSim entity that simulates the Grid resource broker. On receiving an experiment from the user entity, it carries out resource discovery, and determines deadline and budget values based on D and B factors, and then proceeds with scheduling. It schedules Gridlets on resources depending on user constraints, optimization strategy, and cost of resources and their availability. When it receives the results of application processing, it records parameters of interest with the gridsim.Statistics entity. When it has no more processing requirements, it sends the END OF SIMULATION event to the gridsim.GridSimShutdown entity.

class BrokerResource: acts as a placeholder for the broker to maintain a detailed record on the resources it uses for processing user applications. It maintains resource characteristics, a list of Gridlets assigned to the resource, the actual amount of MIPS available to the user, and a report on the Gridlets processed. These measurements help in extrapolating and predicating the resource performance from the user point of view and aid in scheduling jobs dynamically at runtime.

class ReportWriter: a user-defined, optional GridSim entity which is meant for creating a report at the end of each simulation by interacting with the gridsim.Statistics entity. If the user does not want to create a report, then he or she can pass 'null' as the name of the ReportWriter entity. Note that users can choose any name for the ReportWriter entity and for the class name since all entities are identified by their name defined at the runtime.

3.5 Visual Modeler for Grid Modeling

The GridSim 1.0 toolkit does not have any additional GUI tools that enable easier and faster modeling and simulation of Grid environments. It only comprises the Java-based packages that are to be called by users' self-written Java simulation programs. VM has now been included in GridSim 2.0 toolkit [2]. There are many similar tools like VM that generate source code, but are not related to Grid simulation in specific, such as SansGUITM and SimCreator.

Other than the GridSim toolkit, there are several other tools that support application scheduling simulation in Grid computing environments. The notable ones include Bricks, MicroGrid and SimGrid.

3.5.1 Architecture of Visual Modeler

VM adopts Model View Controller (MVC) architecture as illustrated in Figure below. The MVC architecture is designed to separate the user display from the control of user input and the underlying information model. The following are the reasons for using MVC architecture in VM:

- opportunity to represent the same domain information in different ways. Designers can therefore refine the user interfaces to satisfy certain tasks and
- user characteristics.
- ability to develop the application and user interface separately.
- ability to inherit from different parts of the class hierarchy.
- ability to define control style classes which provide common features separately from how these features may be displayed.

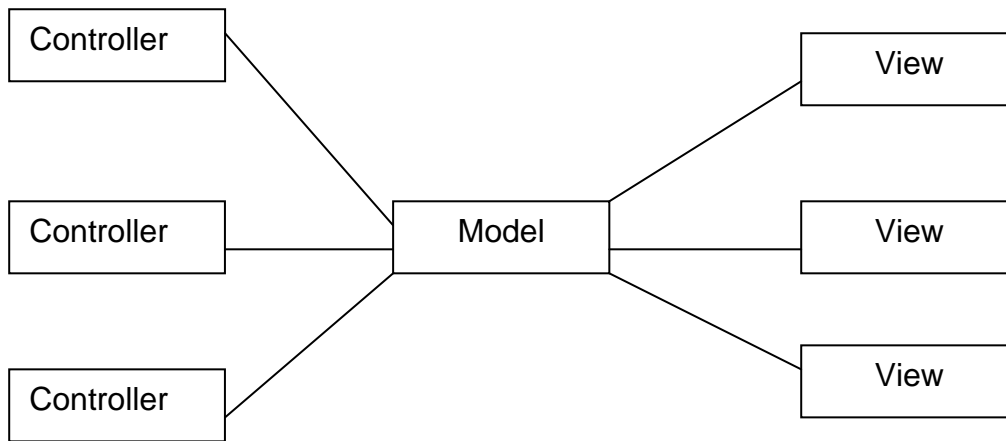


Fig 3.3 The basic Model View Controller architecture

VM is programmed using Java as Java supports powerful Swing libraries that facilitate easy GUI programming [2]. In addition, Java works well with the MVC architecture. VM support the following main features:

- enables the creation of many Grid testbed users and resources.
- generates the simulation scenario into Java code that the users can compile and run the simulation with the GridSim toolkit.
- saves and retrieves a VM project file that contains an experiment scenario in eXtensible Markup Language (XML) format, and
- works on different operating system platforms (as it is implemented in Java).

3.5.2 Design of Visual Modeler

The MVC paradigm proposes three class categories:

1. Models – provide the core functionality of the system.
2. Views – present models to the user. There can be more than one view of the same model.
3. Controllers – control how the user interacts with the view objects and manipulates models or views.

Java supports MVC architecture with two classes:

- Observer – any object that wishes to be notified when the state of another Object changes.
- Observable – any object whose state may be of interest to another object.

3.5.2.1 Model

VM consists of three model classes, whose relationships are shown in Figure below.

The models are:

- FileModel – deals with UserModel and ResourceModel for saving and retrieving an XML file format for the VM project file.
- UserModel – stores, creates and deletes one or more user objects.
- ResourceModel – stores, creates and deletes one or more resource objects.

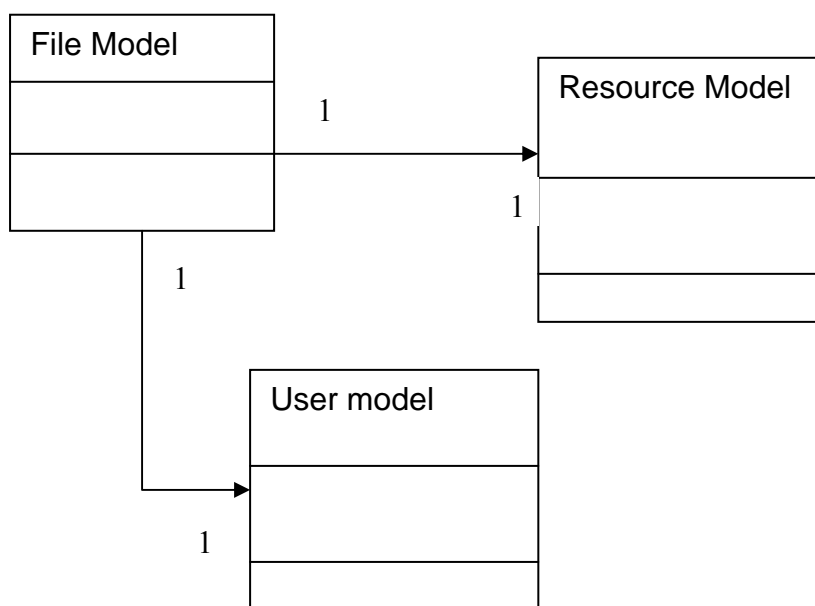


Fig 3.4 Class diagram that contains the model classes of VM and their relationships

Both UserModel and ResourceModel contain objects that are hybrids of both model and view, i.e. they have their own widgets to display the stored values. This technique reduces the number of classes needed and the overall design complexity, thus saving significant development time. All

models extend Java's Observable class, which provides the behaviours required to maintain a list of observers and notify them when there are changes to the model. The initialization for UserModel and ResourceModel is straightforward with both classes not requiring any parameters. However, FileModel needs to request a reference to each of the UserModel and ResourceModel object. It thus becomes an observer of UserModel and ResourceModel when dealing with the saving and retrieving project file.

3.5.2.2 View

Views are registered as dependents of a model. The model broadcasts a message to all dependents when it changes. The main views of VM are:

- MenuView – creates GUI components for menu bar.
- IconView – creates GUI components for icon toolbar.
- DisplayView – creates the main window to display the lists of Grid user and resource.

These three main views are created first, by constructing their widgets independently and then adding their sub-views. This design ensures that the subviews are separate from their parents' view. Each main view contains one or more references to the model since the model contains user/resource objects that also display widgets. This design reduces the number of method calls and any overheads associated with the interaction between the object and its view. There are two possible approaches for creating user/resource objects: with or without their widgets. Creating objects without their widgets requires much less time. However, Lines of Codes (LOC) inside the object class is slightly larger than that of the approach with widgets. The approach without widgets requires less passing of reference objects and messages in comparison to the other method. In addition, the time taken and the amount of memory needed to create components and to register their event listeners are minimal. VM can thus support fast creation of a large number user/resource objects at any one time. This is helpful since a Grid simulation model does not have a limited number of Grid users and resources. Therefore, VM adopts the without-widgets approach by displaying its GUI upon the user's request. When the user closes its GUI window, the GUI components are retained so that they need not be created again for future access (similar to the cache concept in web browsers). This reduces the utilization of memory and enables fast display of GUI components repeatedly.

3.5.2.3 Controller

The controller needs to be informed of changes to the model as any modifications of the model will also affect how the controller processes its input. The controller may also alter the view when there is no change to the model. The primary function of the controller is to manage mouse/keyboard event bindings (e.g. to create pop-up menus). If an input event requires modification of applicationspecific data, the controller notifies the model accordingly. In this implementation, controllers which relate to views have several responsibilities. Firstly, they implement Java's event-handler interfaces to listen for appropriate events, e.g. the icon toolbar controller detects the clicking of the save toolbar button and notifies FileModel to save a project file. Secondly, views delegate the construction and maintenance of button panels to controllers. Thirdly, controllers can broadcast semantic events (e.g. save file), to objects who have informed the controller that they are interested in these events.

3.5.3 Grid Computing Environment Simulation

This section describes how a simulated Grid computing environment is created using VM. First, the Grid users and resources for the simulated Grid environment have to be created. This can be done easily using the wizard dialog as shown in Figure below. The VM user only needs to specify the required number of users and resources to be created. Random properties can also be automatically generated for these users and resources. The VM user can then view and modify the properties of these Grid users and resources by activating their respective property dialog.

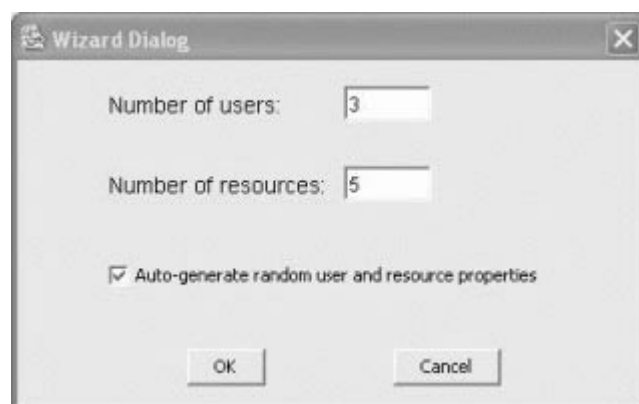


Fig. 3.5. Wizard dialog to create Grid users and resources

Figure below shows the property dialog of a sample Grid resource. Resources of different capabilities and configurations can be simulated, by setting properties such as cost of using this resource, allocation policy of resource managers (time/space-shared) and number of machines in the resource (with Processing Elements (PEs) in each machine and their Million Instructions Per Second (MIPS) rating).

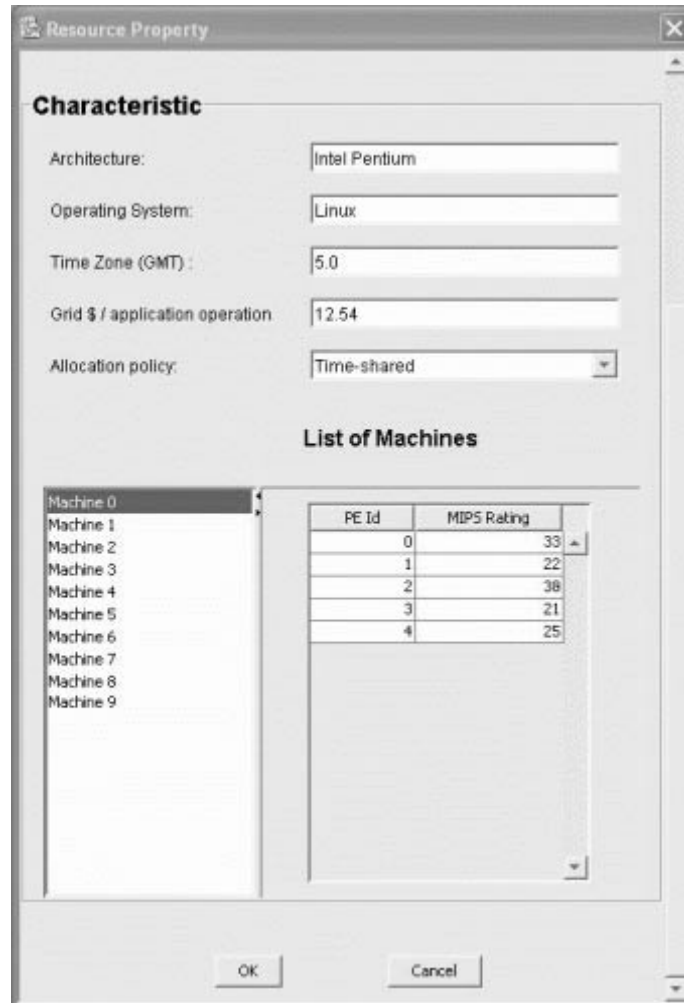


Fig. 3.6. Resource dialog to view Grid resource properties

Figure below shows the property dialog of a sample Grid user. Users can be created with different requirements (application and quality of service requirements). These requirements include the baud rate of the network (connection speed), maximum time to run the simulation, time delay between each simulation, and scheduling strategy such as cost and/or time optimization for running the application jobs. The application jobs are modelled as Gridlets. The parameters of Gridlets that can be defined includes number of Gridlets, job length of Gridlets (in Million Instructions (MI)), and length of input and output data (in bytes). VM provides a useful feature that supports random distribution of these parameter values within the specified derivation range. Each Grid user has its

own economic requirement (deadline and budget) that constrains the running of application jobs. VM supports the flexibility of defining deadline and budget based on factors or values. If it is factor-based (between 0.0 and 1.0), a budget factor close to 1.0 signifies the Grid user's willingness to spend as much money as required. The Grid user can have the exact cost amount that it is willing to spend for the value-based option.

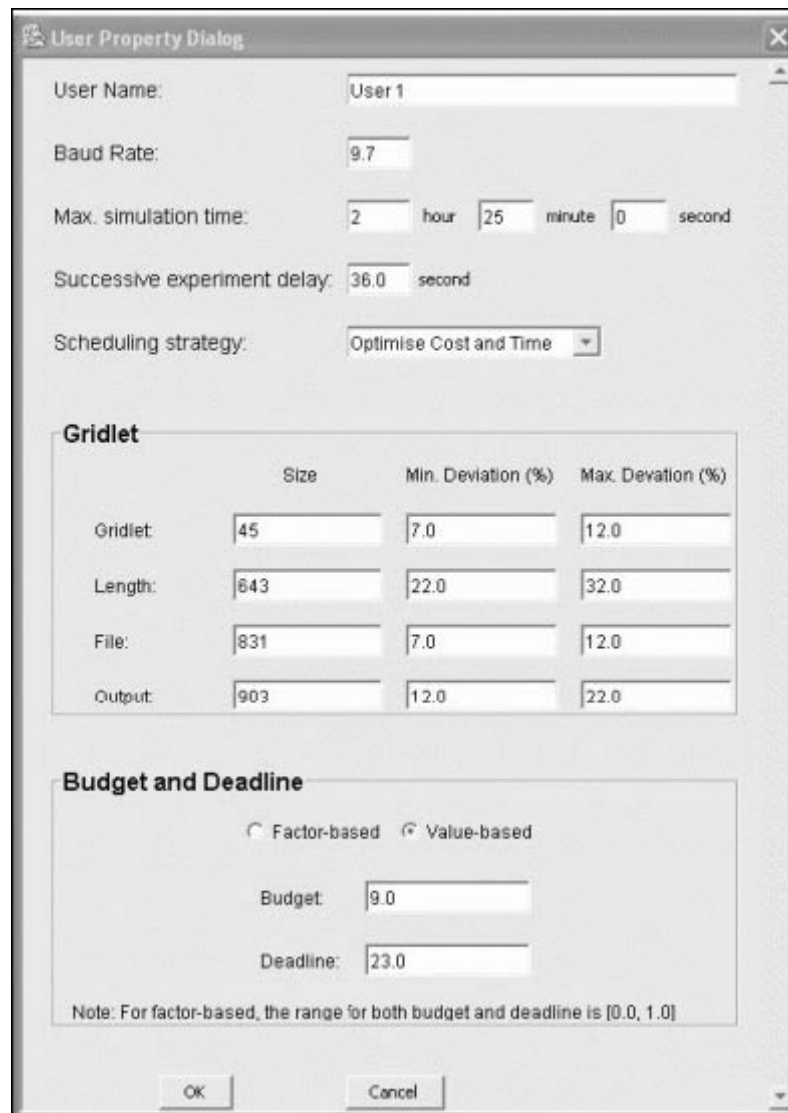


Fig. 3.7. User dialog to view Grid user properties

VM will automatically generate Java code for running the Grid simulation. This file can then be compiled and run with the GridSim toolkit packages to simulate the required Grid computing environment.

In Grid environment, the client nodes can enroll the Grid at any time, deliver requests to the schedule center, and monitor the implementation of themselves tasks. There are five important modules in the schedule center [4]; the architecture of the scheduler center can be expressed as in the figure 4.1. Each node denotes a client or a Grid resource, and each edge denotes a link between nodes.

4.1.2 The Work Mode of schedule Center

When a Grid client delivers a request, the Grid works as follows:

- The client delivers a request that contains an application description (workload of the application, communication load, and time limit, etc.) to the task receiver.
- The task receiver queues the tasks in priority, and delivers the first task in the queue to scheduling manager. The receiver maintains an unscheduled task queue; record the client name and user requirements, application workload, communication load, and time limit, etc.
- The scheduling manager selects a most appropriate scheduling scheme from all schemes according to the resource graph and user requirement, then deliver the scheme to task dispatcher and inform resource monitor. This is the most important and complex module in the schedule center, there are many scheduling strategies can be put in the scheduling manager.
- Task dispatcher delivers the task to the selected resource, and gives transfer delay and actual task assignment result to the resource monitor. The dispatcher maintains a scheduled task queue; record the assigned resource name, application workload, communication load, and time limit, etc. when some task is finished or failed, the dispatcher delete the task in the queue or put the task back to unscheduled task queue, and notifies the resource monitor.
- The resource monitor maintains the up to date state of every resource and revises the resource graph.
- We use a graphic to express the Grid environment, each node denotes a client or a Grid resource (number of processors, speed of each processor, I/O bandwidth, RAM capability,

and disk capacity, etc.), and each edge denotes a link between nodes (bandwidth, delay, quantity of pheromone. etc.).

4.2 Scheduling Algorithm

4.2.1 Ant Algorithm

Dorigo M. introduced the Ant algorithm in 1996, which is a new heuristic, predictive scheduling algorithm; it is based on the behavior of real ants. When the blind insects, such as ants look for food, every moving ant lays some pheromone on the path, thus making the path it followed by a trail of this substance. While an isolated ant moves essentially at random, an ant encountering a previously laid trail can detect it and decide with high probability to follow it, thus reinforcing the trail with its own pheromone. The collective behavior that emerges means where the more are the ants following the trail, the more that trail become attractive for being followed, then the pheromone on shorter path will be increased quickly, the quantity of pheromone on every path will affect the possibility of other ants to select path. At last all the ants will choose the shortest path [4].

Experiments show that ant algorithm have on optimum problem solving and exist extensible heuristics that means solving the optimum problem with m scale could extended to solve the similar problem with larger than m scale effectively.

The algorithm has been successfully used to solve many NP problems, such as TSP, assignment problem, job-shop scheduling and graph coloring. The algorithm has inherent parallelism, and we can validate its scalability. So, it's obvious that ant algorithm is suitable to be used in Grid computing task scheduling.

We utilize the characteristics of ant algorithms above mentioned to schedule jobs. We can carry on new task scheduling by experience depend on the result in the past task scheduling. It is very helpful for being within the grid environment.

We design an ant algorithm for task scheduling in Grid Computing, and put it in the scheduling manager of the simulation center.

4.2.2 Ant Algorithm for Task scheduling in Grid Computing

When a resource 'j' enrolls the Grid, it is asked to submit its performance parameters, Number of PE, processing capability MIPS of every PE and the communication ability etc. Resource monitor tests these parameters for validation, and initialize the links pheromone or trail intensity for the resource j in the ant algorithm [4].

Let $\tau_j(0)$ be the trail intensity on path j from the scheduling center to the corresponding resources j at time 0, it is given by the formula:

$$\tau_j(0) = m \times p + c / s_j \quad (4.1)$$

Where 'm' is the No. of PEs

'p' is the MIPS of one PE

'c' is the size of the parameters

's_j' is the parameter transfer time from resource 'j' to resource monitor.

c/s_j is related to the communication bandwidth ability of the resource j.

Every time a new resource joins the grid, a resource gets failed, a task is assigned, or there is some task returned, the pheromone on path from schedule centre to the corresponding resource will be changed. So the trail intensity at time t is given by

$$\tau_j^{new} = \rho \cdot \tau_j^{old} + \Delta \tau_j \quad (4.2)$$

$\Delta \tau_j$ is the change of pheromone on path from the schedule center to resource j between time t-1 and t

ρ is the permanence of pheromone ($0 < \rho < 1$)

$1 - \rho$ is evaporation coefficient of pheromone.

When task is assigned to resource j,

$$\Delta \tau_j = -k \quad (4.3)$$

k is the compute and transfer quality of the task which is relevant to computation workload and communication quantity of the job.

When task successfully returned from resource 'j', and resource j is released

$$\Delta\tau_j = C_e \times k, \quad C_e \text{ is the encourage factor.} \quad (4.4)$$

When task failed and returned from resource 'j',

$$\Delta\tau_j = C_p \times k, \quad C_p \text{ is the punish factor.} \quad (4.5)$$

Different choices about above coefficients arose different quantity of $\Delta\tau_j$ and when to update the $\tau_j(t)$ cause different instantiation of the ant algorithm.

The transition possibility of task assignment to every resource will be recomputed as:

$$\begin{aligned} \rho_j^k(t) &= \frac{[\tau_j(t)]^\alpha * [\eta_j(t)]^\beta}{\sum_u [\tau_u(t)]^\alpha * [\eta_u]^\beta} \quad j, u \in 1 \dots m \\ &= 0 \quad \text{others} \end{aligned} \quad (4.6)$$

$\tau_j(t)$ is the pheromone intensity on the path from schedule center to resource j at time t.

$\eta_j(t)$ is called visibility, namely the innate performance quantity of the resource j, that is $\tau_j(0)$.

α is the importance of the pheromone.

β is the importance of resource innate attributes.

This gives the probabilities of the resources in the grid which helps the task scheduler in scheduling. The scheduler assigns a new task 'k' on to a resource 'j' at time 't' with a probability equal to it's corresponding $\rho_j^k(t)$.

4.2.3 Problem with Ant Algorithm

The scheduler schedules a task based on the possibilities $[\rho_j^k(t)]$ of the resources. The problem with this algorithm is, it may schedule a task to a resource with low possibility even if the resources with high possibility are free. Also, If the tasks are always scheduled to the resource with high

possibility, then the burden on the resource may be increased and the jobs may be kept waiting in the queue waiting for the resource to be free.

4.3 Enhancement to the Algorithm

The algorithm can be modified in such a way that if the difference between the possibility of the resource selected using ant-algorithm for execution of a task and the possibility of the resource with highest probability is greater than certain threshold, then the task will be scheduled to the resource with highest possibility. Otherwise, the task will be assigned to the resource selected by the ant-algorithm only. The selection of the threshold plays an important role. Since this modified algorithm takes the resource with highest possibility into consideration, the cost of execution of the tasks may increase when compared to that of ant-algorithm.

So, enhancement to the algorithm is done as, if scheduler using ant algorithm finds a resource 'j' for which a new task 'k' can be scheduled at time 't' with a possibility $\rho_j^k(t)$. Then check for

$$(\rho_h^k(t) - \rho_j^k(t)) \leq T_h \quad (4.7)$$

Where 'h' is the resource with highest $\rho^k(t)$

T_h is the threshold which can be taken as $1/(\text{no. of resources})$

If the above equation is false then task is assigned to resource 'h'. Otherwise to resource 'j'.

Chapter 5

Implementation Details

5.1 Implementation

Grid resources information and Grid user's information will be given as input to start the simulation. Then the simulation starts step by step as shown in the flow diagram figure 3.2. First, GridSim initialization will be performed. Next, resource creation followed by the construction of 'Scheduling' class objects. Each Scheduling object corresponds to a grid user. The 'Scheduling' class implements the gridlet creation for the user and also implements the scheduling of those gridlets on to the appropriate resources that are registered to the grid. The gridlets are created based on user specified parameters (total number of gridlets, average MI of each gridlet, and the deviation percentage of the MI).

Then, the system starts the GridSim simulation. It first gathers the characteristics of the available Grid resources created in the resource creation section in the system. The scheduler maintains the pheromone value of each resource (from schedule center) and will compute the possibilities of the current resources available in the Grid every time when a gridlet is to be scheduled. Then, the scheduler selects a resource based on the ant-algorithm and schedules the gridlets. When a gridlet is submitted to a resource it's pheromone value will be decreased by the compute and transfer quality of the task. The Grid resources process the received gridlets and send back the processed gridlets to the grid user.

The system then gathers the processed gridlets sent back to the user. If the gridlet is successfully is executed then, the pheromone value of the resource on which this gridlet is submitted will be increased. If the gridlet execution fails, the pheromone of the resource will be decreased. The simulation will be completed when all the user gridlet get executed on the grid resources and get back the results.

5.1.1 Input to the Simulation

In simulation, we use 10 resources and 20 users, some resource randomly gets failed or enrolls the grid. The parameters of the resources are given in the Table 5.1.

Resource	No. of PE	MIPS of PE	Communication Rate (Mb/s)	Cost/sec
R0	1	50	10	1
R1	4	377	30	2
R2	2	380	10	1
R3	16	410	40	7
R4	4	410	20	2
R5	2	200	20	2
R6	6	410	20	3
R7	16	410	50	7
R8	4	377	20	2
R9	16	410	50	7

Table 5.1 Parameters of resources

The job characteristics of 20 Grid users are given in Table 5.2

User	Job length	Job Size	Output Size
U1	75000	100	50
U2	65000	80	40
U3	8000	15	10
U4	5000	15	10
U5	70000	80	35
U6	75000	85	50
U7	200	10	10
U8	45000	90	40
U9	3000	30	10
U10	100	10	10
U11	70000	95	50
U12	22000	50	20
U13	500	15	10
U14	75000	100	50
U15	32000	40	25
U16	68000	80	45
U17	7000	20	10
U18	300	10	10
U19	1000	10	10
U20	54000	60	40

Table 5.2 Task parameters of 20 users

The parameter values $\alpha = 0.5$, $\beta = 0.5$, $\rho = 0.8$, $C_e = 1.1$, $C_p = 0.8$ are taken in the algorithm simulation.

Inputs to the simulation can be given by GUI created. Different pictures are shown below of the GUI for different tasks:

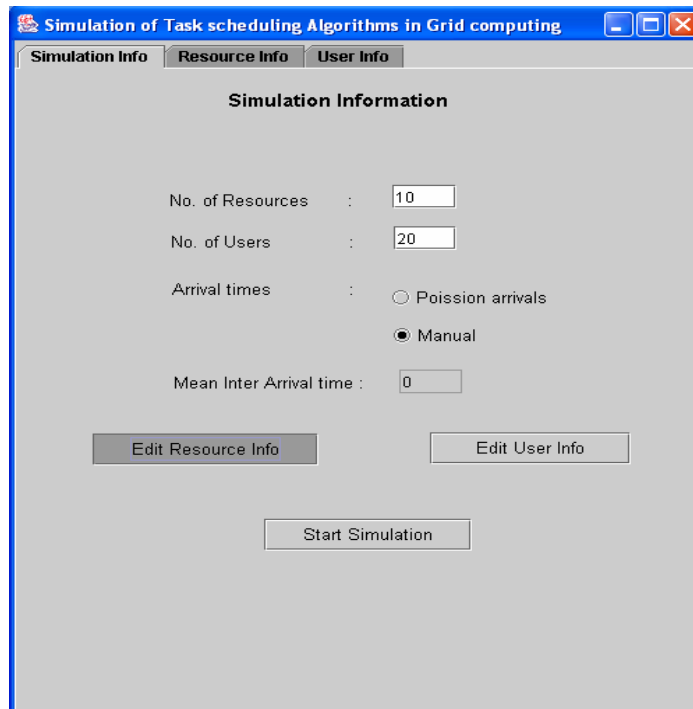


Fig 5.1 Number of users and Resources

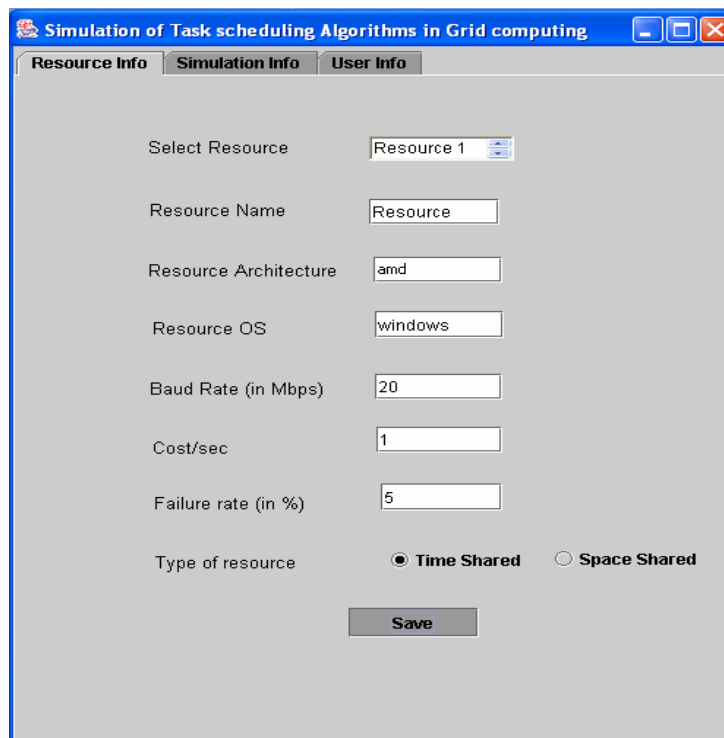


Fig 5.2 Resource Information

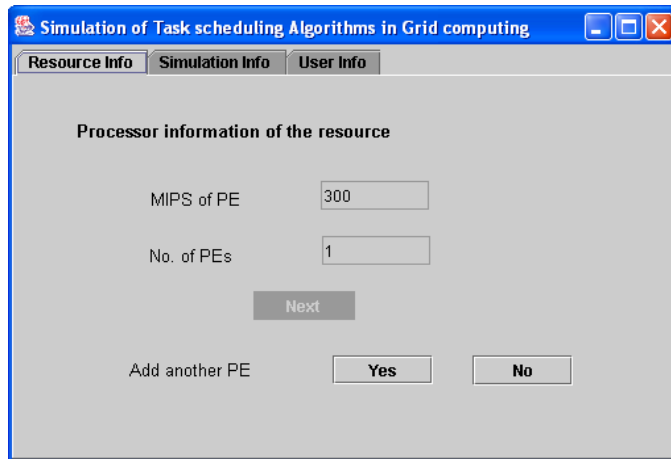


Fig 5.3 Processor information of resource

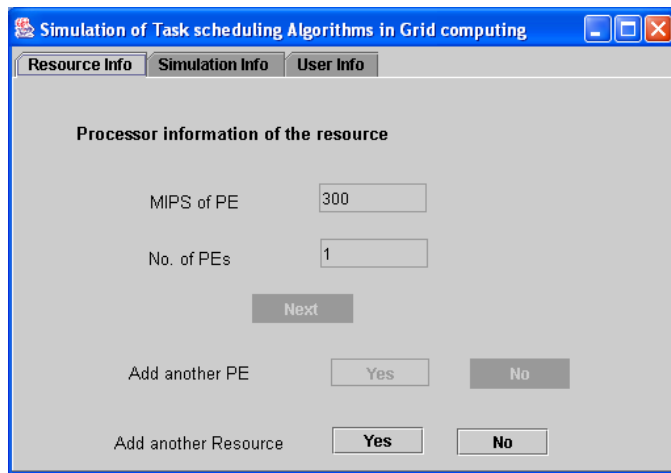


Fig 5.4 Adding another resource

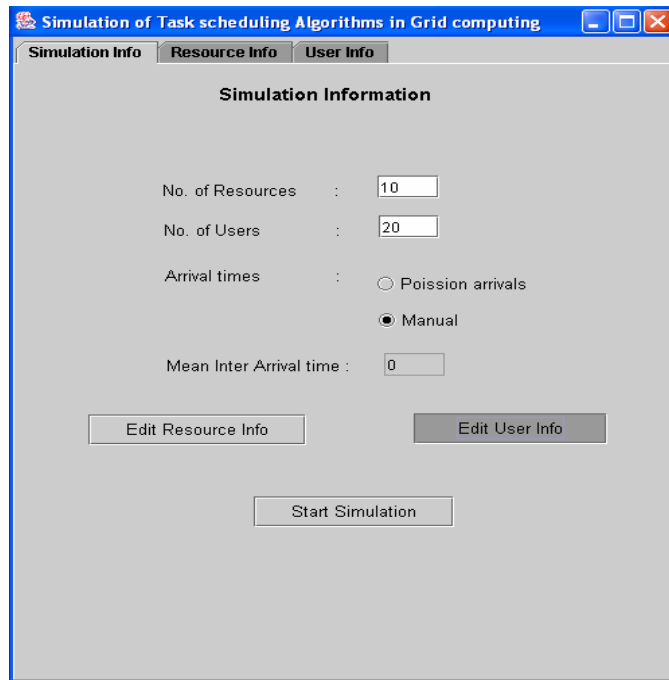


Fig 5.5 Editing User Information

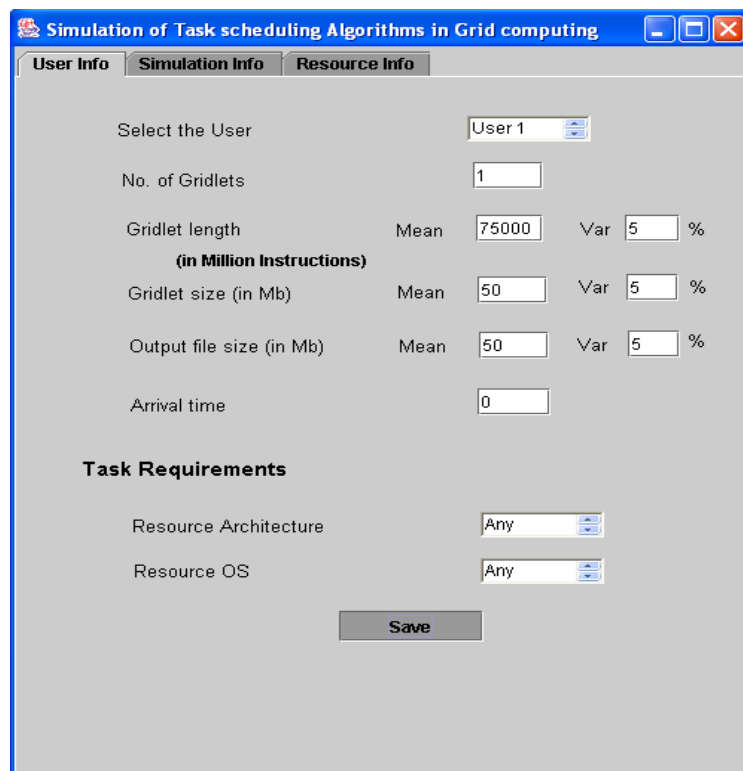


Fig 5.6 User Information

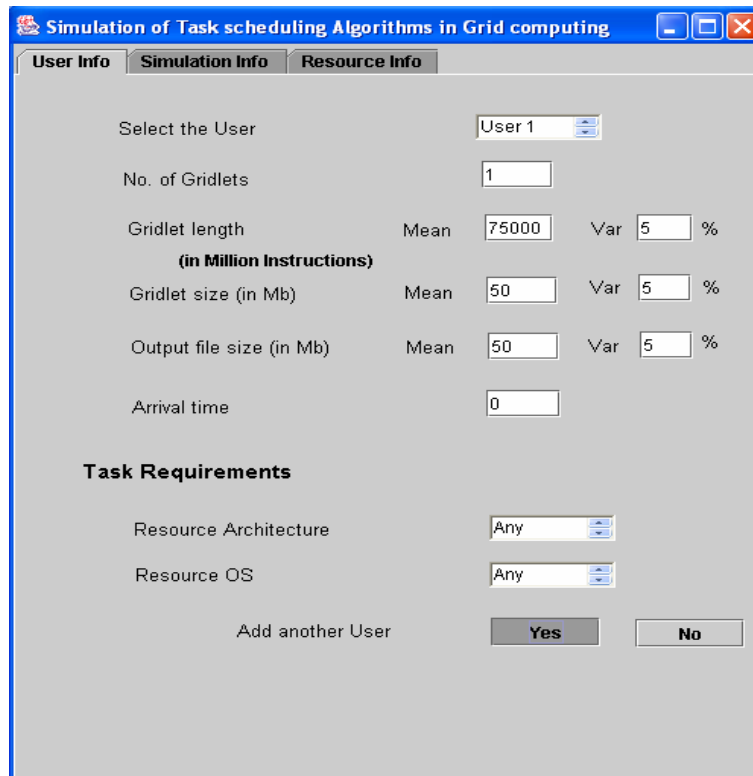


Fig 5.7 Adding another User

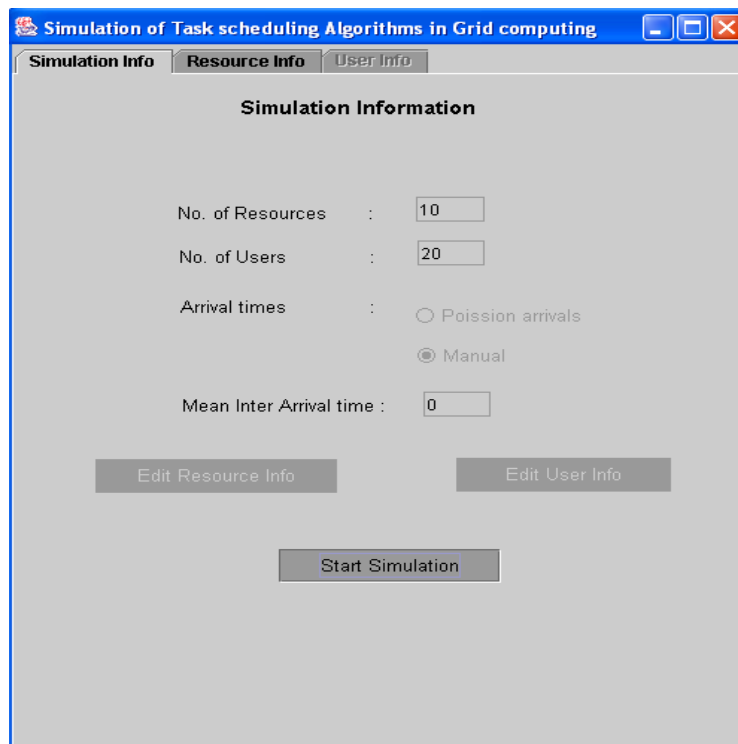


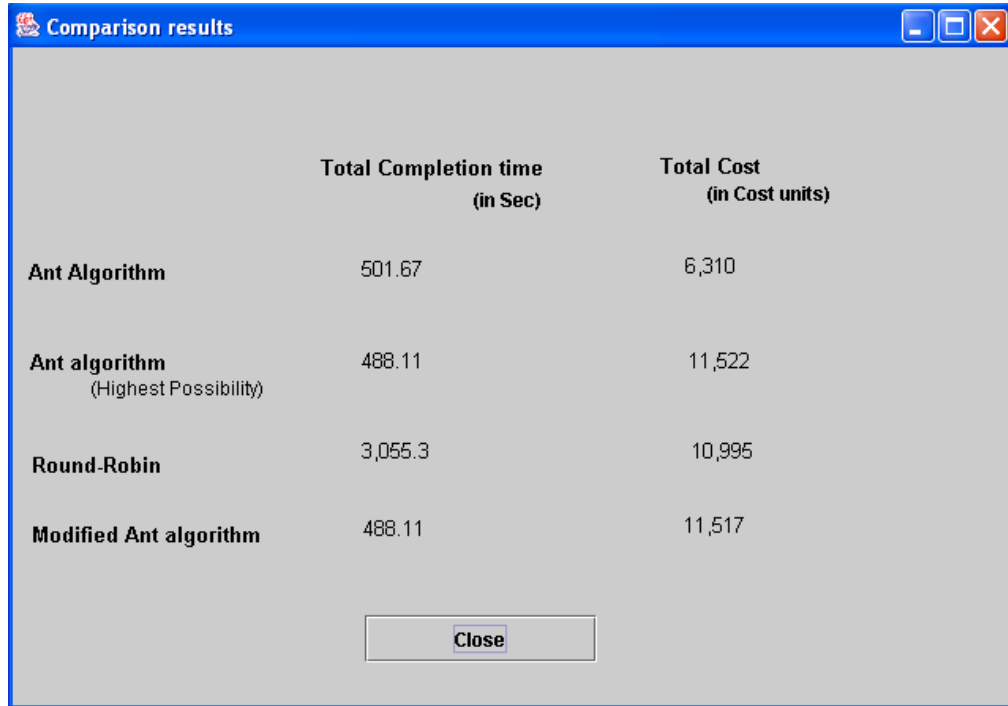
Fig 5.8 Starting the Simulation

Chapter 6

Results and Discussion

6.1 Simulation Results

Results after simulation using the above inputs are given as below:



	Total Completion time (in Sec)	Total Cost (in Cost units)
Ant Algorithm	501.67	6,310
Ant algorithm (Highest Possibility)	488.11	11,522
Round-Robin	3,055.3	10,995
Modified Ant algorithm	488.11	11,517

Fig 6.1 Results of different algorithms

From fig 6.1, shows the results of different algorithms with the same input of 10 resources and 20 grid users. The total completion time of the modified ant algorithm is lesser than that of ant-algorithm. But the cost of executing the tasks in modified algorithm is more than that of ant-algorithm.

Threshold Vs Total Execution time

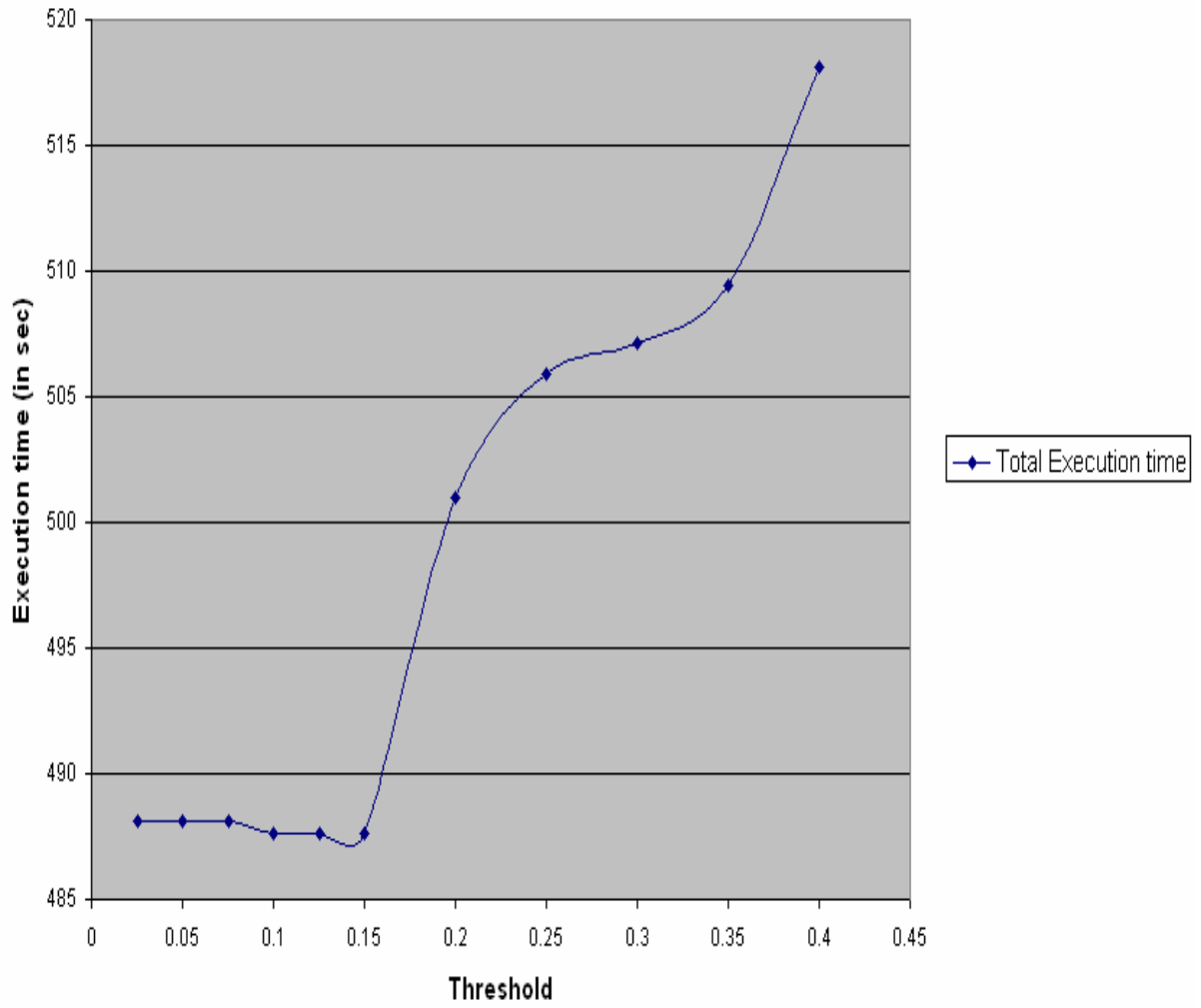


Fig 6.2 Threshold Vs Execution time in Modified ant algorithm

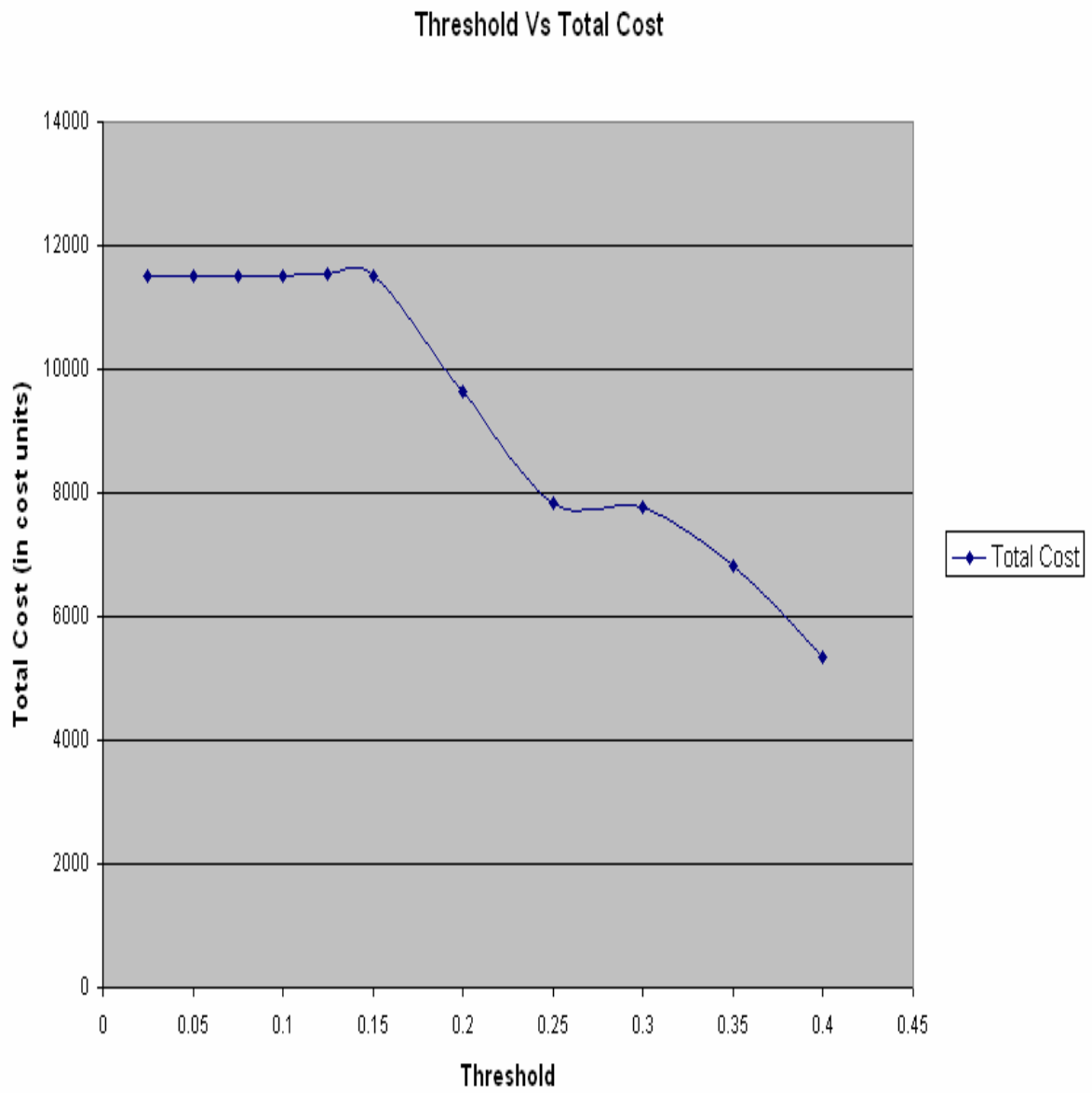


Fig 6.3 Threshold Vs Total Cost in Modified ant algorithm

Fig 6.2, 6.3 shows the total execution time and total cost for different values of threshold in the modified ant-algorithm. If the threshold value increases certain value, then the algorithm results will be similar to that of ant-algorithm.

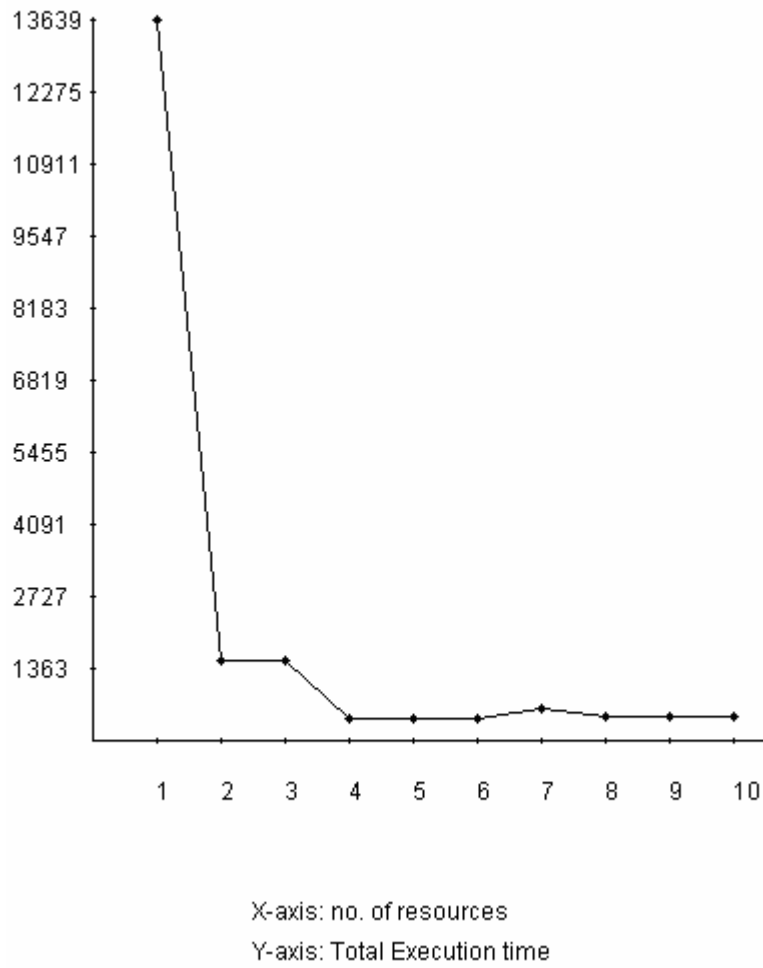


Fig 6.4(i) Graph showing scalability of Ant Algorithm(Execution Time)

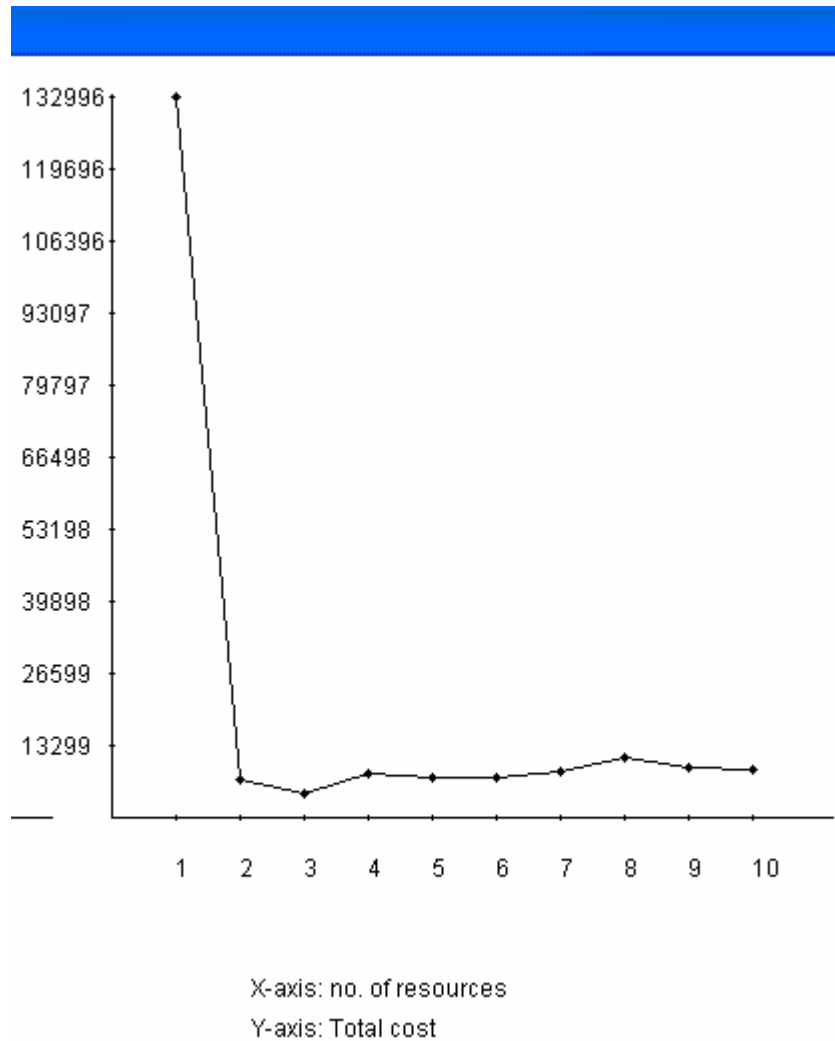
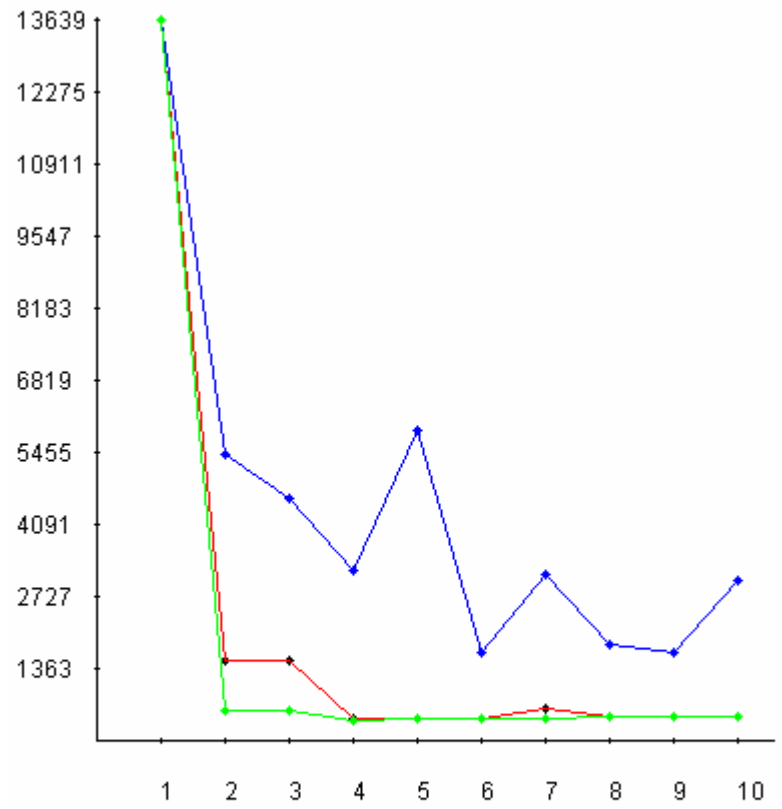


Fig 6.4(ii) Graph showing scalability of Ant Algorithm(Total Cost)

The figure 6.4 is showing the graph of No. of resources Vs Total Execution time and Total Cost the Ant Algorithm.

The following are the results of simulation of various algorithms for the 20 grid users as the number of resources in the grid increases. Adding of resources to the grid is taken in the same order as mentioned in the input resource characteristics above.

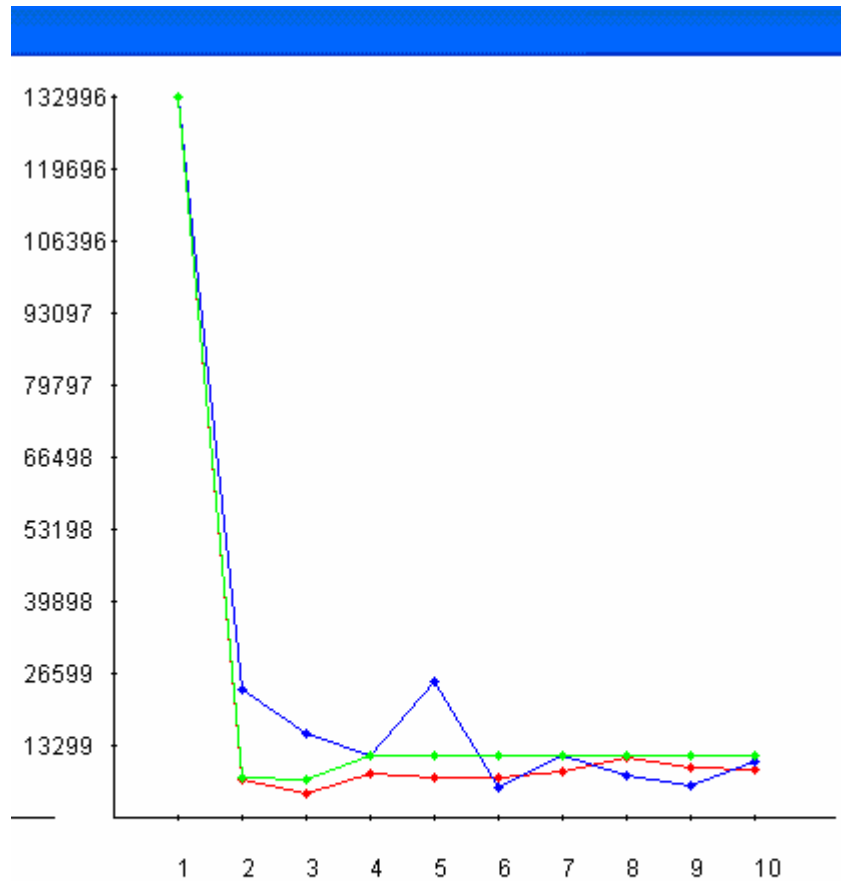
Multiple simulation graphs



X-axis: no. of resources
Y-axis: Total Execution time(in Sec)

- ant algorithm
- modified ant-algorithm
- round-robin algorithm

Fig 6.5(i) Comparison Graphs of various algorithm results(Execution Time)



X-axis: no. of resources
 Y-axis: Total cost(in Cost units)

- ant algorithm
- modified ant-algorithm
- round-robin algorithm

Fig 6.5(ii) Comparison Graphs of various algorithm results(Total Cost)

The above figure shows the algorithm results for various algorithms. The results from the fig 6.5 show that the total execution time of tasks in modified ant-algorithm is lesser than that of ant-algorithm.

Chapter 7

Conclusion and Future Work

Since the structure of the grid dynamically changes, there is no particular scheduling algorithm which can effectively utilize all the resources in a grid. But, the algorithm should be distributable, scalable and fault tolerant. It should also estimate the state of the resources which are currently in the grid. And predictive state estimation is better than non-predictive state estimation because it uses the current state as well as the historical status of the resources. So, the static algorithms like round-robin scheduling are no longer suitable.

The Ant algorithm is a heuristic predictive state estimating scheduling algorithm which is distributable, scalable and fault tolerant. The inherent parallelism and scalability make the algorithm very suitable to be used in grid computing task scheduling. The modified form of ant-algorithm can be used if cost of using the resources is not a concern. The selection of threshold value in the modified ant-algorithm is very important.

As a future work, this scheduling method can be put into actual Grid environment for validation to make QOS scheduling. This algorithm can be made more suitable for wide use if pricing factor will also be included.

References

1. 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 (CCPE), Volume 14, Issue 13-15, Pages: 1175-1220, Wiley Press, USA, November - December 2002.
2. Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya, “ *Visual Modeler for Grid Modeling and Simulation (GridSim) Toolkit*”, P.M.A. Sloot et al. (Eds): ICCS 2003, LNCS 2659, pp. 1123-1132, 2003.
3. Graham Ritchie and John Levine, “ *A hybrid ant algorithm for task scheduling independent jobs in heterogeneous computing environments*”, American Association for artificial Intelligence, 2004.
4. Zhihong XU, Xiangdan HOU, Jizhou SUN, “ *Ant- Algorithm-Based Task scheduling in Grid Computing*”, Montreal, IEEE, CCECE2003- CCGEI2003, May 2003.
5. Li Chunlin, Li Layuan, “ *QoS based resource scheduling by computational economy in computational grid*”, Elsevier, 2006.
6. Manzur Murshed and Rajkumar Buyya, “ *Using the GridSim Toolkit for Enabling Grid Computing Education*”, IEEE, 2002.
7. David Abramson, Rajkumar Buyya, Manzur Murshed, “ *A Deadline and Budget Constrained Cost-Time optimization Algorithm for Scheduling Task Framing Applications on Global Grids*”, IEEE, 2003.
8. Sathish S. Vadhiyar, Jack J. Dongarra, ” *A MetaScheduler for the Grid*”, Natinal Science Foundation, HPDC, IEEE, 2002.
9. Anthony Sulistio, Uros Cibej, Borut Robic and Rajkumar Buyya, “ *A Toolkit for modeling and Simulation of Data Grids with Integration of Data Storage, Replication and Analysis*”, Elsevier Science, Jan 2006.

10. Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, Graham R. Nudd, “ *Grid Load Balancing using Intelligent Agents*”, Elsevier, Oct-2004.
11. Nikolaos Doulamis, Anastasios Doulamis, Antonios Litke, Athanasios Panagakis, Theodora Varvarigou, Emmanuel Varvarigos, “ *Adjusted fair Scheduling and non-linear Workload Prediction for Qos guarantees inGrid Computing*”, Elsevier, 2005.
12. Herbert Schildt, “ *The Complete Refrence JAVA*”, Edition 2005.
13. Web Site <http://www.gridbus.org/gridsim>.
14. Web Site <http://www.buyya.com/gridsim>.
15. Web Site <http://gridbus.cs.mu.oz.au/~raj/gridsim>.

Appendix

Source Code

SimInfo.java: This file contains the code for the GUI to accept input data about the resource characteristics of the resources in the grid and job characteristics of the users.

Information.java: This contains a java class 'Information' of which object is used for storing the resource and job characteristics as well as the simulation results into a file.

ResultDisplay.java: This file contains the GUI code to display the results obtained after simulation.

Comparison.java: This contains the GUI code to display the comparison results after simulation of different scheduling algorithms.

Graphs.java: This contains the GUI code to display the graphical information using the multiple simulation results.

Scheduling.java: This file contains the core code of the project which implements the simulation of ant-algorithm based task scheduling.

```
//Scheduling.java

import java.util.*;
import java.text.*;
import gridsim.*;
import gridsim.util.*;
import java.io.*;

public class Scheduling extends GridSim
{
    private Integer ID_;
    private String name_;
    private GridletList list_;
    private GridletList receiveList_ failList;
    private int totalResource_;
    private LinkedList resPher,resPoss,innate;
    private static GridResource res[];
    public static ResourceCharacteristics resCharList[];
    private static int rcount,algo;
    private static ResultDisplay gui;
    private double arr_time;
    public static double gridletCount[];
    private static int userCount;
    public static long arrTimes[];
    private double K;
    public static int resGridletCount[];
    private static int roundRobin;
    private static Information inf;
    private static double ResUtil[];
```

```

private static String resName[];
private static int resId[],usedTime[],noInst[];
private static double resCost[];
private static double SimTime;
private int uno;

Scheduling(String name, double baud_rate,int total_res,LinkedList resP,LinkedList resPos,double at,int algo,int u)
    throws Exception
{
    super(name, baud_rate);
    this.name_ = name;
    this.totalResource_ = total_res;
    this.receiveList_ = new GridletList();
    this.arr_time=at;
    this.algo=algo;
    this.uno=u;

    failList=new GridletList();
    resPher=resP;
    resPoss=resPos;

    // Gets an ID for this entity
    this.ID_ = new Integer( getEntityId(name) );
    // Creates a list of Gridlets or Tasks for this grid user
    this.list_ = createGridlet( this.ID_.intValue());
}

/**
 * The core method that handles communications among GridSim entities.
 */
public void body()
{
    int resourceID[] = new int[this.totalResource_];
    double resourceCost[] = new double[this.totalResource_];
    String resourceName[] = new String[this.totalResource_];

    LinkedList resList;
    ResourceCharacteristics resChar;

    while (true) {
        // need to pause for a while to wait GridResources finish
        // registering to GIS
        super.gridSimHold(2.0); // hold by 2 seconds
        resList = (LinkedList) getGridResourceList();
        if (resList.size() == this.totalResource_)
            break;
    } //while
    // a loop to get all the resources available
    int i = 0;
    for (i = 0; i < resList.size(); i++) {
        resourceID[i] = ( Integer) resList.get(i).intValue();
        resId[i]=resourceID[i];
        // Requests to resource entity to send its characteristics
        send(resourceID[i], arr_time,GridSimTags.RESOURCE_CHARACTERISTICS, this.ID_);
    }
}

```

```

// waiting to get a resource characteristics
resChar = (ResourceCharacteristics) receiveEventObject();
resCharList[i]=resChar;
resourceName[i] = resChar.getResourceName();
resourceCost[i] = resChar.getCostPerSec();
// record this event into "stat.txt" file
recordStatistics("\Received ResourceCharacteristics " +
                "from " + resourceName[i] + "\", ");
Double pher = new Double(resChar.getMIPSRating()+inf.resBr[i]);
resPher.add(pher);
setPheremone(i,pher.doubleValue());
resPoss.add(new Double(0));
} //for
//computing the denominator
innate = resPher;
Gridlet gridlet;
String info;
// a loop to get one Gridlet at one time and sends it to a random grid
// resource entity. Then waits for a reply
int id = 0;
int flag[] = new int[totalResource_];
double prob = 0, max = 0, val;
int mpos = 0;
for (i = 0; i < this.list_.size(); i++)
{
    gridlet = (Gridlet)this.list_.get(i);
    info = "Gridlet_" + gridlet.getGridletID();

//Scheduling algorithm
    NumberFormat numFormat=NumberFormat.getInstance();
    numFormat.setMaximumFractionDigits(2);
    double tot = 0, pos;

    for (int j = 0; j < totalResource_; j++) {
        tot += StrictMath.pow(getPheremone(j), 0.5) *
            StrictMath.pow(getInnate(j), 0.5);
    }
    for (int j = 0; j < totalResource_; j++) {
        pos = StrictMath.pow(getPheremone(j), 0.5) *
            StrictMath.pow(getInnate(j), 0.5);
        pos /= tot;
        setPossibility(j, pos);
    } //int j=0
    for (int t = 0; t < totalResource_; t++)
        flag[t] = 0;
    double rand = StrictMath.random();
    if (algo == 0) //ant-algorithm
    {
        prob = 0;
        while (prob < rand) {
            max = 0;
            for (int p = 0; p < totalResource_; p++) {
                if (flag[p] == 1)
                    continue;

```

```

if(!inf.userArch[uno].equals("Any")&&!inf.userArch[uno].equals(inf.resArch[p]))|
  !inf.userOs[uno].equals("Any")&&!inf.userOs[uno].equals(inf.resOs[p]))
  continue;
val = getPheremone(p);
if (max < val) {
  max = val;
  mpos = p;
} //if max<val
} //for
flag[mpos] = 1;
prob += getPossibility(mpos);
} //while
} //algo==0
if (algo == 1) //Highest possibility algorithm
{
  max = 0;
  for (int p = 0; p < totalResource_; p++) {
    if(!inf.userArch[uno].equals("Any")&&!inf.userArch[uno].equals(inf.resArch[p]))|
      (!inf.userOs[uno].equals("Any")&&!inf.userOs[uno].equals(inf.resOs[p])))
      continue;
    val = getPheremone(p);
    if (max < val) {
      max = val;
      mpos = p;
    }
  } //for
} //algo=1
id = mpos;
if (algo == 2) //round-robin algorithm
{
  while(true)
  {
    id = roundRobin;
    roundRobin++;
    if (roundRobin == this.rcount)
      roundRobin = 0;
    if(!inf.userArch[uno].equals("Any")&&!inf.userArch[uno].equals(inf.resArch[id]))|
      !inf.userOs[uno].equals("Any")&&!inf.userOs[uno].equals(inf.resOs[id]))
      continue;
    break;
  } //while
} //algo ==2
if(algo==3) //modified ant-algorithm
{
  prob = 0;
  while (prob < rand) {
    max = 0;
    for (int p = 0; p < totalResource_; p++) {
      if (flag[p] == 1)
        continue;
    if(!inf.userArch[uno].equals("Any")&&!inf.userArch[uno].equals(inf.resArch[p]))|
      !inf.userOs[uno].equals("Any")&&!inf.userOs[uno].equals(inf.resOs[p]))
        continue;
    val = getPheremone(p);

```

```

        if (max < val) {
            max = val;
            mpos = p;
        } //if max<val
    } //for
    flag[mpos] = 1;
    prob += getPossibility(mpos);
} //while
id=mpos;
//high probability
max = 0;
for (int p = 0; p < totalResource_; p++) {
    if ((!inf.userArch[uno].equals("Any"))&&!inf.userArch[uno].equals(inf.resArch[p]))||
        (inf.userOs[uno].equals("Any"))&&!inf.userOs[uno].equals(inf.resOs[p]))
        continue;
    val = getPheremone(p);
    if (max < val) {
        max = val;
        mpos = p;
    }
} //for
double v=1/inf.rcount;
if(getPossibility(mpos)-getPossibility(id)>=gui.threshold)
    id=mpos;
} //algo=3

//changing the pheremone after submitting
double newPher = getPheremone(mpos);
K=(gridlet.getGridletFileSize()/inf.resBr[id]+
    gridlet.getGridletLength()/resCharList[id].getMIPSRating());
newPher = 0.8*newPher - K;
//updating the pheremone
setPheremone(mpos, newPher);
//
//          Sends one Gridlet to a grid resource specified in "resourceID"
gridletSubmit(gridlet, resourceID[id],arr_time,true);
resGridletCount[id]++;
//System.out.println(this.name_ + "After submitting Gridlet:" + i);

tot = 0;
for (int j = 0; j < totalResource_; j++) {
    tot += StrictMath.pow(getPheremone(j), 0.5) *
        StrictMath.pow(getInnate(j), 0.5);
}
for (int j = 0; j < totalResource_; j++) {
    pos = StrictMath.pow(getPheremone(j), 0.5) *
        StrictMath.pow(getInnate(j), 0.5);
    pos /= tot;
    setPossibility(j, pos);
} //int j=0
// Recods this event into "stat.txt" file for statistical purposes
recordStatistics("\nSubmit " + info + " to " +
    resourceName[id] + "\n", "");
//
//          if the resouce fails

```

```

double expTime = 1;
expTime=gridlet.getGridletLength()/resCharList[id].getMIPSRating();
double rndTime;
if (StrictMath.random()<=0.02)
{
    rndTime = StrictMath.random() * expTime;
    //cancelling the gridlet
    super.gridletCancel(gridlet, resourceID[id], rndTime);
    failList.add(gridlet);
    newPher = getPheremone(mpos);
    //updating the pheremone
    newPher = 0.8 * newPher+ 0.8 * K;
    setPheremone(mpos, newPher);
    tot = 0;
    for (int j = 0; j < totalResource_; j++) {
        tot += StrictMath.pow(getPheremone(j), 0.5) *
            StrictMath.pow(getInnate(j), 0.5);
    }
    for (int j = 0; j < totalResource_; j++) {
        pos = StrictMath.pow(getPheremone(j), 0.5) *
            StrictMath.pow(getInnate(j), 0.5);
        pos /= tot;
        setPossibility(j, pos);
    } //int j=0
} //if
else {
    // waiting to receive a Gridlet back from resource entity
    gridlet = this.gridletReceive();
    if(resourceID[id]>-1)
    {
        String subResName;
        subResName=gridlet.getResourceName(resourceID[id]);
        int p;
        for(p=0;p<inf.rcount;p++)
        if((inf.resName[p]).equals(subResName))
        {
            ResourceCharacteristics rc=resCharList[p];
            usedTime[p]+=gridlet.getProcessingCost()/inf.resCost[p];
            noInst[p]+=gridlet.getGridletLength();
            resCost[p]=usedTime[p]*inf.resCost[p];
        }
    } //id>-1

    // if the sent gridlet is successfully executed
    try {
        if (gridlet.getGridletStatus() == Gridlet.SUCCESS) {
            newPher = getPheremone(mpos);
            //updating the pheremone
            newPher = 0.8 * newPher + 1.1 * K;
            setPheremone(mpos, newPher);
        }
    } //try
    catch (Exception e) {
        System.out.println(e);
    }
}

```

```

    }
    gridlet.setResourceParameter(resourceID[id], resourceCost[id]);
    tot = 0;
    for (int j = 0; j < totalResource_; j++) {
        tot += StrictMath.pow(getPheremone(j), 0.5) *
            StrictMath.pow(getInnate(j), 0.5);
    }
    for (int j = 0; j < totalResource_; j++) {
        pos = StrictMath.pow(getPheremone(j), 0.5) *
            StrictMath.pow(getInnate(j), 0.5);
        pos /= tot;
        setPossibility(j, pos);
    } //int j=0
    // Recods this event into "stat.txt" file for statistical purposes
    recordStatistics("\Received " + info + " from " +
        resourceName[id] + "\", gridlet.getProcessingCost());
    // stores the received Gridlet into a new GridletList object
    this.receiveList_.add(gridlet);
}
} //else
// shut down all the entities, including GridStatistics entity since
// we used it to record certain events.
shutdownGridStatisticsEntity();
shutdownUserEntity();
terminateIOEntities();
// System.out.println(this.name_ + ":%%% Exiting body()");

} //body
/**
 * Gets the list of Gridlets
 * @return a list of Gridlets
 */
public GridletList getGridletList() {
    return this.receiveList_;
}
public GridletList getFailList()
{
    return failList;
}
/**
 * This method will show you how to create Gridlets with and without
 * GridSimRandom class.
 * @param userID the user entity ID that owns these Gridlets
 * @return a GridletList object
 */
private GridletList createGridlet(int userID)
{
    // Creates a container to store Gridlets
    GridletList list = new GridletList();
    // We create three Gridlets or jobs/tasks manually without the help
    // of GridSimRandom
    int id = 0;
    double length;
    long file_size;

```



```

    long output_size;
    // sets the PE MIPS Rating
    GridSimStandardPE.setRating(100);
    for (int i = 0; i < inf.nog[uno]; i++)
    {
        double agl=inf.glLength[uno];
        length=agl;
        file_size=inf.glSize[uno];
        output_size=inf.glOutput[uno];
        // creates a new Gridlet object
        Gridlet gridlet = new Gridlet(id + i, length, file_size,output_size);
        gridlet.setUserID(userID);
        // add the Gridlet into a list
        list.add(gridlet);
    }
    userCount++;
    return list;
}

public static void initSim(ResultDisplay g,int x)
{
    gui=g;
    algo=x;
    try
    {
        FileInputStream fis=new FileInputStream("c:\\proj\\info.tmp");
        ObjectInputStream ois=new ObjectInputStream(fis);
        inf=(Information)ois.readObject();
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
    res=new GridResource[inf.rcount];
    ResUtil=new double[inf.rcount];
    arrTimes=new long[inf.ucount];
    gridletCount=new double[inf.ucount];
    resGridletCount=new int[inf.rcount];
    resCharList=new ResourceCharacteristics[inf.rcount];
    resName=new String[inf.rcount];
    resId=new int[inf.rcount];
    usedTime=new int[inf.rcount];
    noInst=new int[inf.rcount];
    resCost=new double[inf.rcount];
    //System.out.println("Starting simulation of Ant algorithm");
    try
    {
        // First step: Initialize the GridSim package. It should be called
        // before creating any entities. We can't run this example without
        // initializing GridSim first. We will get run-time exception
        // error.
        int num_user =inf.ucount; // number of grid users
        Calendar calendar = Calendar.getInstance();
        boolean trace_flag = true; // mean trace GridSim events

```

```

// list of files or processing names to be excluded from any
// statistical measures
String[] exclude_from_file = { "" };
String[] exclude_from_processing = { "" };
// the name of a report file to be written. We don't want to write
// anything here. See other examples of using the ReportWriter
// class
String report_name = null;
// Initialize the GridSim package
GridSim.init(num_user, calendar, trace_flag, exclude_from_file,
            exclude_from_processing, report_name);
// Second step: Creates one or more GridResource objects
for(int c=0;c<inf.rcount;c++)
{
    MachineList ml=new MachineList();
    PEList plist=new PEList();
    for(int pc=0;pc<inf.resNpe[c];pc++)
        plist.add(new PE(pc,inf.mips[c][pc]));
    Machine m=new Machine(0,plist);
    ml.add(m);
    int policy;
    if(inf.resType[c]==0)
        policy=ResourceCharacteristics.TIME_SHARED;
    else
        policy=ResourceCharacteristics.SPACE_SHARED;
    ResourceCharacteristics resChar=new ResourceCharacteristics(inf.resArch[c],inf.resOs[c],ml,policy,0,inf.resCost[c]);

    res[rcount++] = createGridResource(resChar, inf.resName[c], inf.resBr[c]);
    resName[c]=inf.resName[c];
}
startSimulation();
}
catch (Exception e)
{
    e.printStackTrace();
    System.out.println("Unwanted errors happen"+e);
}
}

/**
 * Creates one Grid resource.
 * @param name a Grid Resource name
 * @return a GridResource object
 */
private static GridResource createGridResource(ResourceCharacteristics resChar,String name
        ,double baud_rate)
{
    double peakLoad = 0.0; // the resource load during peak hour
    double offPeakLoad = 0.0; // the resource load during off-peak hr
    double holidayLoad = 0.0; // the resource load during holiday
    // incorporates weekends so the grid resource is on 7 days a week
    LinkedList Weekends = new LinkedList();
    Weekends.add(new Integer(Calendar.SATURDAY));
    Weekends.add(new Integer(Calendar.SUNDAY));

```

```

// incorporates holidays. However, no holidays are set in this example
LinkedList Holidays = new LinkedList();
GridResource gridRes = null;
try {
    gridRes = new GridResource(name, baud_rate, 11L*13*17*19*23+1,
        resChar, 0, 0, 0, Weekends, Holidays);
}
catch (Exception e) {
    e.printStackTrace();
}
return gridRes;
}
public static void startSimulation()
{
LinkedList resP, resPos;
resP=new LinkedList();
resPos=new LinkedList();
int total_resource=inf.rcount;
try{
int users=inf.ucount;
Scheduling user[]=new Scheduling[users];
double iat=inf.miat;
long sh_time = 0;
Poisson arrtime;
if(iat<=0)
arrtime = new Poisson("inter-arrival time", 1);
else
arrtime = new Poisson("inter-arrival time", iat);
for(int u=0;u <users;u++)
{
if(inf.arrType==0)
sh_time+=arrtime.sample();
else
sh_time=inf.arrTime[u];
user[u]=new Scheduling("User_"+String.valueOf(u),560.00,total_resource,resP,resPos,sh_time,algo,u);
}
//Starts the simulation
GridSim.startGridSimulation();
SimTime=GridSim.clock();
//writing results
NumberFormat numFormat=NumberFormat.getInstance();
int totCost=0;
for(int i=0;i<rcount;i++)
{
ResUtil[i]=usedTime[i]/SimTime;
gui.ta_res.append("\n'+ resName[i]+' "+resId[i]+' "+numFormat.format(usedTime[i])+"\t'+numFormat.format(ResUtil[i])+"\t'+
numFormat.format(noInst[i])+"\t'+numFormat.format(resCost[i])+"\t'+ " +numFormat.format(resGridletCount[i]));
totCost+=resCost[i];
}
gui.l_totcost.setText("Total cost: "+numFormat.format(totCost));
inf.totResCost[0][algo]=totCost;
for(int i=0;i<userCount;i++)
gui.list_users.add("User"+(i+1),i);
gui.antGletList=new GridletList[userCount];

```

```

gui.antGfailList=new GridletList[userCount];
gui.l_time.setText("Total execution time of the tasks: "+numFormat.format(SimTime));
inf.totExecTime[0][algo]=SimTime;
System.out.println("exec time:"+SimTime+" cost:"+totCost);
// Prints the Gridlets when simulation is over
for(int i=0;i<users;i++)
{
GridletList newList = null;
GridletList fList=null;
newList = user[i].getGridletList();
gui.antGletList[i]=newList;
printGridletList(newList, user[i].name_,true);
fList = user[i].getFailList();
gui.antGfailList[i]=fList;
printGridletList(fList, user[i].name_,false);
}
try{
FileOutputStream fos = new FileOutputStream("c:\\proj\\info.tmp");
ObjectOutputStream oos= new ObjectOutputStream(fos);
oos.writeObject(inf);
oos.close();
}
catch(Exception excep)
{
System.out.println(excep);
}
} //try
catch(Exception e)
{
System.out.println("error in startsimulation");
e.printStackTrace();
}
}
/**
 * Prints the Gridlet objects
 * @param list list of Gridlets
 */
public static void printGridletList(GridletList list, String name,boolean status)
{
int size = list.size();
Gridlet gridlet;
if(list.size()==0)
return;

String indent = " ";
for (int i = 0; i < size; i++)
{
gridlet = (Gridlet) list.get(i);
}
}
synchronized public double getPheromone(int i) // Getting pheromone
{
return ((Double)resPher.get(i)).doubleValue();
}
}

```

```
synchronized public double getPossibility(int i)
{
    return ((Double)resPoss.get(i)).doubleValue();
}
synchronized public void setPheremone(int i,double v)
{
    resPher.set(i, new Double(v));
}
synchronized public void setPossibility(int i,double v)
{
    resPoss.set(i, new Double(v));
}
synchronized public double getInnate(int i)
{
    return ((Double)innate.get(i)).doubleValue();
}
} // end of Scheduling class
```

```

// ResultDisplay.java
// This file is for displaying the result after simulation
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import gridsim.*;
import java.text.*;

class ResultDisplay extends JFrame implements ActionListener { // Result Display class
    public static int rcount,ucount;
    public static double threshold;
    JPanel p1=new JPanel(); //Creates Panel
    JPanel p2=new JPanel();
    JLabel l_algo = new JLabel(); // Provide label
    JTabbedPane jTabbedPane1 = new JTabbedPane();
    JTextArea ta_res = new JTextArea();
    JTextArea ta_users = new JTextArea(); // Area for the text
    JLabel jLabel1 = new JLabel();
    java.awt.List list_users = new java.awt.List();

    public GridletList antGletList[];
    public GridletList antGfailList[];
    JLabel l_time = new JLabel();
    JLabel jLabel2 = new JLabel();
    JLabel jLabel3 = new JLabel();
    JLabel jLabel4 = new JLabel();
    JLabel jLabel5 = new JLabel();
    JLabel jLabel6 = new JLabel();
    JLabel jLabel7 = new JLabel();
    JLabel jLabel8 = new JLabel();
    JLabel jLabel9 = new JLabel();
    JLabel jLabel10 = new JLabel();
    JLabel jLabel11 = new JLabel();
    JLabel jLabel12 = new JLabel();
    JLabel jLabel13 = new JLabel();
    JLabel l_tcost = new JLabel();
    JLabel jLabel14 = new JLabel();
    JToggleButton tb_close = new JToggleButton(); // Creating instance of toggle button
    JLabel jLabel15 = new JLabel();

    public static void main(String a[]) // Main fuction which starts execution
    {
        ResultDisplay rd=new ResultDisplay();
        int algo=3;
        algo=Integer.parseInt(a[0]);
        threshold=Double.parseDouble(a[1]);
        Scheduling.initSim(rd,algo);
        if(algo==0)
            rd.l_algo.setText("Algorithm: Ant-algorithm");
        if(algo==1)
            rd.l_algo.setText("Algorithm: Highest probability algorithm");
        if(algo==2)
            rd.l_algo.setText("Algorithm: Round-Robin algorithm");
        if(algo==3)
            rd.l_algo.setText("Algorithm: Modified ant-algorithm");
    }
}

```

```

if(algo==4)
    rd.l_algo.setText("Algorithm: Enhanced ant-algorithm");
rd.setSize(700,545);
rd.setVisible(true);
} //main

public ResultDisplay() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
} //constructor

private void jbInit() throws Exception {
    setTitle("Results after simulation");
    l_algo.setFont(new java.awt.Font("Dialog", 1, 13));
    l_algo.setText("Algorithm: Modified Ant-Algorithm:");
    l_algo.setBounds(new Rectangle(50, 29, 313, 33));
    this.getContentPane().setLayout(null);
    jTabbedPane1.setBounds(new Rectangle(41, 80, 567, 343));
    ta_users.setText("");
    jLabel1.setFont(new java.awt.Font("Dialog", 0, 13));
    jLabel1.setText("Select User");
    jLabel1.setBounds(new Rectangle(261, 55, 95, 22));
    list_users.setBounds(new Rectangle(361, 56, 81, 21));
    list_users.addActionListener(new ResultDisplay_list_users_actionAdapter(this));
    l_time.setBounds(new Rectangle(49, 439, 322, 23));
    jLabel2.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel2.setText("Gridlet ID");
    jLabel2.setBounds(new Rectangle(15, 101, 62, 16));
    jLabel3.setBounds(new Rectangle(95, 100, 62, 19));
    jLabel3.setText("Status");
    jLabel3.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel4.setBounds(new Rectangle(143, 97, 61, 23));
    jLabel4.setText("Res ID");
    jLabel4.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel5.setBounds(new Rectangle(201, 90, 76, 34));
    jLabel5.setText("Arrival time");
    jLabel5.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel6.setBounds(new Rectangle(285, 98, 79, 18)); // Setting bounds and creating rectangle
    jLabel6.setText("Compl time");
    jLabel6.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel7.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel7.setText("Cost");
    jLabel7.setBounds(new Rectangle(466, 100, 72, 19));
    jLabel8.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel8.setText("Res Name");
    jLabel8.setBounds(new Rectangle(15, 68, 67, 19));
    jLabel9.setBounds(new Rectangle(84, 69, 26, 19));
    jLabel9.setText("ID");
    jLabel9.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel10.setBounds(new Rectangle(174, 69, 75, 19));
    jLabel10.setText("Utilization");
}

```

```

jLabel10.setFont(new java.awt.Font("Dialog", 1, 11));
jLabel11.setBounds(new Rectangle(109, 69, 75, 19));
jLabel11.setText("Used time");
jLabel11.setFont(new java.awt.Font("Dialog", 1, 11));
jLabel12.setBounds(new Rectangle(238, 68, 135, 19));
jLabel12.setText("Inst executed (in MIs)");
jLabel12.setFont(new java.awt.Font("Dialog", 1, 11));
jLabel13.setFont(new java.awt.Font("Dialog", 1, 11));
jLabel13.setText("Cost");
jLabel13.setBounds(new Rectangle(373, 69, 45, 19));
l_tcost.setFont(new java.awt.Font("Dialog", 1, 12));
l_tcost.setText("cost");
l_tcost.setBounds(new Rectangle(384, 439, 222, 19));
jLabel14.setBounds(new Rectangle(424, 68, 72, 19));
jLabel14.setText("Job count");
jLabel14.setFont(new java.awt.Font("Dialog", 1, 11));
tb_close.setFont(new java.awt.Font("Dialog", 0, 13));
tb_close.setText("Close");
tb_close.setBounds(new Rectangle(18, 469, 659, 31));
tb_close.addActionListener(new ResultDisplay_tb_close_actionAdapter(this));
jLabel15.setFont(new java.awt.Font("Dialog", 1, 11));
jLabel15.setText("Exec time");
jLabel15.setBounds(new Rectangle(372, 99, 79, 18));
p2.add(ta_users, null);
p2.setLayout(null);
ta_users.setBounds(new Rectangle(15, 127, 540, 274));
p1.add(ta_res, null);
p1.add(jLabel8, null);
p1.add(jLabel9, null);
p1.add(jLabel11, null);
p1.add(jLabel10, null);
p1.add(jLabel12, null);
p1.add(jLabel14, null);
p1.add(jLabel13, null);
this.getContentPane().add(l_tcost, null);
this.getContentPane().add(l_time, null);
this.getContentPane().add(tb_close, null);
jTabbedPane1.insertTab("User Info",null,p2,null,0);
jTabbedPane1.insertTab("Resource Info",null,p1,null,1);
p1.setLayout(null);
ta_res.setBounds(new Rectangle(8, 89, 548, 220));
this.getContentPane().add(l_algo, null);
this.getContentPane().add(jTabbedPane1, null);
p2.add(list_users, null);
p2.add(jLabel1, null);
p2.add(jLabel2, null);
p2.add(jLabel3, null);
p2.add(jLabel4, null);
p2.add(jLabel5, null);
p2.add(jLabel6, null);
p2.add(jLabel15, null);
p2.add(jLabel7, null); // Formatting output
ta_res.setEditable(false);
ta_users.setEditable(false);

```



```

tb_close.addActionListener(this);
}

void list_users_actionPerformed(ActionEvent e) {
    ta_users.setText("");
    int uno=list_users.getSelectedIndex();
    GridletList list=antGletList[uno];
    int size = list.size();
        Gridlet gridlet;
        NumberFormat numFormat=NumberFormat.getInstance();
        numFormat.setMaximumFractionDigits(2);
        String indent = "  ";

    for (int i = 0; i < size; i++)
    {

        gridlet = (Gridlet) list.get(i);
        ta_users.append("\n'+indent + "    "+ gridlet.getGridletID() + indent
            + indent);

        // if (gridlet.getGridletStatus() == Gridlet.SUCCESS)

            ta_users.append(" SUCCESS");

            ta_users.append( indent + " "+gridlet.getResourceID() +
                indent + indent +indent+" " +numFormat.format(gridlet.getExecStartTime()+indent+indent+indent+" "+
                    numFormat.format(gridlet.getFinishTime()+
"+indent+numFormat.format(gridlet.getProcessingCost()/gridlet.getCostPerSec(gridlet.getResourceID()))+indent+"
"+numFormat.format(gridlet.getProcessingCost()) );
        }
        list=antGfailList[uno];
        size=list.size();
        for (int i = 0; i < size; i++)
            {

                gridlet = (Gridlet) list.get(i);

                ta_users.append("\n'+indent + "    "+ gridlet.getGridletID() + indent
                    + indent);

                // if (gridlet.getGridletStatus() == Gridlet.SUCCESS)

                    ta_users.append(" FAILED  ");

                    ta_users.append( indent + " "+gridlet.getResourceID() +
                        indent +indent+" " +numFormat.format(gridlet.getExecStartTime()+indent+indent+indent+" "++"Not
Finished"+indent+
                            numFormat.format(gridlet.getProcessingCost()/gridlet.getCostPerSec()+
"+indent+numFormat.format(gridlet.getProcessingCost()) );
            }

    }
}

```

```

public void actionPerformed(ActionEvent e)
{
    dispose();
}

void tb_close_actionPerformed(ActionEvent e) {

}

} //class
class ResultDisplay_list_users_actionAdapter implements java.awt.event.ActionListener {
    ResultDisplay adaptee;

    ResultDisplay_list_users_actionAdapter(ResultDisplay adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.list_users_actionPerformed(e);
    }
}

class ResultDisplay_tb_close_actionAdapter implements java.awt.event.ActionListener {
    ResultDisplay adaptee;

    ResultDisplay_tb_close_actionAdapter(ResultDisplay adaptee) {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e) {
        adaptee.tb_close_actionPerformed(e);
    }
}
// This is the main file which executes first and calls the scheduling

```

// Information.java

```
import java.io.*;

class Information implements Serializable{
    public int lrcount,luccount,rcount,uccount,miat,arrType,resType[],nog[],glLength[],glSize[],glOutput[],arrTime[],resNpe[];
    public String resName[],resArch[],resOs[];
    public double resBr[],resCost[],resFr[];
    public String userOs[],userArch[];
    public double totResCost[][],totExecTime[][]; // Resource cost and total execution time
    public int noOfExec=0;
    public int mips[][]; // Million Instructions per second
    Information()
    {
        totResCost=new double[20][4]; // Total Cost
        totExecTime=new double[20][4]; // Total execution Time
    }
}

// This file creates the input values object. The values which are taken as input no. of PE, arrival time ets.
```

// Comaprison.java

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.text.*;
import java.awt.event.*;

// This file compares all the algorithms i.e ant, modified ant, round robin, highest possibility
class Comparison extends JFrame implements ActionListener {
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();
    JLabel jLabel3 = new JLabel();
    JLabel jLabel4 = new JLabel();
    JLabel jLabel5 = new JLabel();
    JLabel jLabel6 = new JLabel(); // Creates label
    JLabel l_aatct = new JLabel();
    JLabel l_aahct = new JLabel();
    JLabel l_rrtct = new JLabel();
    JLabel l_aatc = new JLabel();
    JLabel l_aahc = new JLabel();
    JLabel l_rrtc = new JLabel();
    Information inf;
    JLabel jLabel7 = new JLabel();
    JLabel l_maatct = new JLabel();
    JLabel l_maatc = new JLabel();
    JButton b_close = new JButton();

    public Comparison() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        setTitle("Comparison results");
        jLabel1.setFont(new java.awt.Font("Dialog", 1, 13));
        jLabel1.setText("Total Completion time ");
        jLabel1.setBounds(new Rectangle(220, 60, 169, 35));
        this.getContentPane().setLayout(null);
        jLabel2.setFont(new java.awt.Font("Dialog", 1, 13));
        jLabel2.setText("Total Cost");
        jLabel2.setBounds(new Rectangle(424, 69, 100, 23)); // Drawing rectangle and setting bounds
        jLabel3.setBounds(new Rectangle(10, 125, 169, 35));
        jLabel3.setText("Ant Algorithm");
        jLabel3.setFont(new java.awt.Font("Dialog", 1, 13));
        jLabel4.setBounds(new Rectangle(49, 200, 123, 31));
        jLabel4.setText("(Highest Possibility)");
        jLabel4.setFont(new java.awt.Font("Dialog", 0, 12));
        jLabel5.setBounds(new Rectangle(12, 247, 120, 35));
        jLabel5.setText("Round-Robin");
        jLabel5.setFont(new java.awt.Font("Dialog", 1, 13));
        jLabel6.setFont(new java.awt.Font("Dialog", 1, 13));
```

```

jLabel6.setText("Ant algorithm");
jLabel6.setBounds(new Rectangle(11, 182, 123, 35));
l_aatct.setFont(new java.awt.Font("Dialog", 0, 13));
l_aatct.setText("Ant Algorithm tct");
l_aatct.setBounds(new Rectangle(221, 126, 145, 29));
l_ahtct.setBounds(new Rectangle(221, 184, 145, 29));
l_ahtct.setText("Ant Algorithm htct");
l_ahtct.setFont(new java.awt.Font("Dialog", 0, 13));
l_rrtct.setBounds(new Rectangle(221, 241, 145, 29));
l_rrtct.setText("Ant Algorithm rrtct");
l_rrtct.setFont(new java.awt.Font("Dialog", 0, 13));
l_aatc.setBounds(new Rectangle(426, 124, 145, 29));
l_aatc.setText("Ant Algorithm tc");
l_aatc.setFont(new java.awt.Font("Dialog", 0, 13));
l_aahtc.setBounds(new Rectangle(428, 184, 145, 29));
l_aahtc.setText("Ant Algorithm htc");
l_aahtc.setFont(new java.awt.Font("Dialog", 0, 13));
l_rrtc.setBounds(new Rectangle(430, 241, 145, 29));
l_rrtc.setText("Ant Algorithm rrtc");
l_rrtc.setFont(new java.awt.Font("Dialog", 0, 13));
jLabel7.setFont(new java.awt.Font("Dialog", 1, 13));
jLabel7.setText("Modified Ant algorithm");
jLabel7.setBounds(new Rectangle(15, 291, 168, 35));
l_maatct.setFont(new java.awt.Font("Dialog", 0, 13));
l_maatct.setText("Ant Algorithm rrtct");
l_maatct.setBounds(new Rectangle(222, 290, 145, 29));
l_maatc.setFont(new java.awt.Font("Dialog", 0, 13));
l_maatc.setText("Ant Algorithm rrtct");
l_maatc.setBounds(new Rectangle(425, 288, 145, 29));
b_close.setBounds(new Rectangle(223, 359, 147, 30));
b_close.setText("Close");
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel6, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(l_rrtc, null);
this.getContentPane().add(l_aahtc, null);
this.getContentPane().add(l_ahtct, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(l_rrtct, null);
this.getContentPane().add(jLabel7, null);
this.getContentPane().add(l_maatct, null);
this.getContentPane().add(l_maatc, null);
this.getContentPane().add(b_close, null);
this.getContentPane().add(l_aatc, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(l_aatct, null);
b_close.addActionListener(this);
try{
    FileInputStream fos = new FileInputStream("c:\\proj\\info.tmp");
    ObjectInputStream ios= new ObjectInputStream(fos);
    inf=(Information)ios.readObject();
    NumberFormat nf=NumberFormat.getInstance();
    nf.setMaximumFractionDigits(2);

```

```

int c=inf.rcount-1;
l_aatct.setText(""+nf.format(inf.totExecTime[c][0])); //setting text output
l_aatc.setText(""+nf.format(inf.totResCost[c][0]));
l_aahtct.setText(""+nf.format(inf.totExecTime[c][1]));
l_aahtc.setText(""+nf.format(inf.totResCost[c][1]));
l_rrtct.setText(""+nf.format(inf.totExecTime[c][2]));
l_rrtc.setText(""+nf.format(inf.totResCost[c][2]));
l_maatct.setText(""+nf.format(inf.totExecTime[c][3]));
l_maatc.setText(""+nf.format(inf.totResCost[c][3]));
ios.close();
}
catch(Exception excep)
{
System.out.println(excep);
}

}

public static void main(String a[])
{
Comparison cmp=new Comparison();
cmp.setSize(640,450);
cmp.setVisible(true);
}

public void actionPerformed(ActionEvent ae)
{
dispose();}
}

```

// Displys the output in tabular form for total cost and total execution time.

// Graphs.java

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
// Generates the comparison graphs
class graphs extends Frame{
    Button button1 = new Button();
    int rcount=10,ucount=20;
    public static int algo,flag;
    Information inf;

    public graphs() {
        if(algo==0) // Take appropriate value for algorithm
            setTitle("Ant-algorithm Multiple simulation graphs");
        if(algo==1)
            setTitle("Highest probability algorithm Multiple simulation graphs");
        if(algo==2)
            setTitle("Round-Robin algorithm Multiple simulation graphs");
        if(algo==3)
            setTitle("Modified Ant-algorithm Multiple simulation graphs");
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception {
        button1.setLabel("close");
        button1.setBounds(new Rectangle(356, 494, 105, 26));
        button1.addActionListener(new graphs_button1_actionAdapter(this));
        this.setLocale(java.util.Locale.getDefault());
        this.setLayout(null);
        this.add(button1, null);
    }
    public static void main(String a[])
    {
        algo=Integer.parseInt(a[0]);
        graphs gs=new graphs();
        gs.setSize(950,600);
        gs.setVisible(true);
    }
    public void paint(Graphics g)
    {
        try{
            FileInputStream fis = new FileInputStream("c:\\proj\\info.tmp");
            ObjectInputStream ois= new ObjectInputStream(fis);
            inf=(Information)ois.readObject();
            ois.close();
        }
        catch(Exception excep)
```

```

{
    System.out.println(excep);
}
double maxTime=0,maxCost=0;
rcount=inf.rcount;
ucount=inf.ucount;
for(int i=0;i<rcount;i++)
{
    if(maxTime<inf.totExecTime[i][algo])
        maxTime=inf.totExecTime[i][algo];
    if(maxCost<inf.totResCost[i][algo])
        maxCost=inf.totResCost[i][algo];
}

//horizontal lines
g.drawLine(50,410,410,410);
g.drawLine(440,410,800,410);

//vertical lines
g.drawLine(50,410,50,50);
g.drawLine(440,410,440,50);

int xinc,yinc=360/(inf.rcount);
xinc=360/(rcount+1);

int pexecTime=0,pcost=0;
for (int i = 0; i < rcount; i++) {

    g.fillOval(50 + (i+1) * xinc-1, 410-1, 3, 3);
    g.fillOval(440 + (i+1) * xinc-1, 410-1, 3, 3);

    g.drawString(""+(i+1),50+(i+1)*xinc,440);
    g.drawString(""+(i+1),440+(i+1)*xinc,440);
    //plotting values
    int execTime = (int) (inf.totExecTime[i][algo] * 360 / maxTime);
    g.drawString(""+(int)((i+1)*maxTime/inf.rcount),50-40,410-(i+1)*yinc+5);

    g.fillOval(50 + (i+1) * xinc-2, 410 - execTime-2, 5, 5);

    int cost = (int) (inf.totResCost[i][algo] * 360 / maxCost); //Cost

    g.drawString(""+(int)((i+1)*maxCost/inf.rcount),440-45,410-(i+1)*yinc+5);

    g.fillOval(440 + (i+1)* xinc-2, 410 - cost-2, 5, 5);
    if(flag==0)
        System.out.println("time:" + inf.totExecTime[i][algo] + "   cost:" +
            inf.totResCost[i][algo]);
    if(i==0)
    {
        pexecTime=execTime; // Execution Time
        pcost=cost; // Cost
        continue;
    }
}
g.drawLine(50+i*xinc,410-pexecTime,50+(i+1)*xinc,410-execTime); // Drawing graph

```



```

pexecTime=execTime;
g.drawLine(440+i*xinc,410-pcost,440+(i+1)*xinc,410-cost); // Drawing graph
pcost=cost;

}
if(flag==0)
    flag=1;
for(int i=0;i<inf.rcount;i++)
{
    g.fillOval(50-1,410-(i+1)*yinc-1,3,3);
    g.fillOval(440-1,410-(i+1)*yinc-1,3,3);
}
g.drawString("X-axis: no. of resources",130,500);
g.drawString("X-axis: no. of resources",510,500);
g.drawString("Y-axis: Total Execution time",130,520);
g.drawString("Y-axis: Total cost",510,520);

}

void button1_actionPerformed(ActionEvent e) {
dispose();
}
}

class graphs_button1_actionAdapter implements java.awt.event.ActionListener {
graphs adaptee;

graphs_button1_actionAdapter(graphs adaptee) {
    this.adaptee = adaptee;
}
public void actionPerformed(ActionEvent e) {
    adaptee.button1_actionPerformed(e);
}
}
// Displays the No. of resources vs total cost and total execution time graphs

```