COST EFFECTIVE ENHANCED ANT ALGORITHM BASED LOAD BALANCING TASK SCHEDULING IN GRID COMPUTING

A DISSERTATION

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

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

M. ARUN KUMAR

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2006

CANDIDATE'S DECLARATION

I here by declare that the work which is being presented in the dissertation entitled, **"Cost effective Enhanced Ant algorithm based Load Balancing Task scheduling in Grid Computing"** being submitted in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Information Technology**, in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my work, carried out from August 2005 to June 2006 under the guidance and supervision of **A.K. Sarje**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee.

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

M. Arun Kumar

(M. Arun Kumar)

Date : 19/6/06

Place: Roorkee

CERTIFICATE

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

[illegible] 19/6/06

A.K. Sarje
Professor,
E&C Department,
IIT-Roorkee.

Date : 19/6/06

Place: Roorkee

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude to my guide A.K. Sarje, Professor, Department of Electronics and Computer Engineering, IIT Roorkee, for encouraging me to undertake this dissertation as well as providing me all the necessary guidance and inspirational support throughout this dissertation work, without which this dissertation work would not have been in the present shape.

I am also thankful to the staff of the Computer Lab, Department of Electronics and Computer Engineering, IIT Roorkee for providing necessary facilities.

M. Arun Kumar
(M. Arun Kumar)

Enroll No: 049008

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.

In this dissertation work, an enhanced ant-algorithm for task scheduling in grid is proposed 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

Candidate's declaration i

Acknowledgement ii

Abstract iii

Chapter 1 **Introduction** **1**

1.1 Motivation........ 2

1.2 Problem Statement........ 2

1.3 Organization of the report 3

Chapter 2 **About Grid computing**........ **4**

2.1 Grid concepts and components........ 4

2.2 Grid computing capabilities 5

2.3 Grid architecture 7

2.4 Grid software components 8

Chapter 3 **Task Scheduling on Grid** **11**

3.1 Classification of scheduling & load balancing........ 11

3.2 Scheduler organization........ 12

3.3 State Estimation 13

3.4 Related Work 14

3.4.1 Ant algorithm 14

3.4.2 Ant algorithm based task scheduling algorithm........ 15

Chapter 4 **Design & Implementation**........ **17**

4.1 Enhancement to the algorithm 17

4.2 Proposed task Scheduling Algorithm 17

4.3 Simulation model 19

4.4 Simulation Environment 22

4.5 Implementation........23

Chapter 5 **Results & Discussions.. 26**

Chapter 6 **Conclusion 31**

References.. 32

Appendix A.. 34

Appendix B.. 41

Chapter 1

INTRODUCTION

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[1].

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.

The Grid system is responsible for the execution of the jobs 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. The scheduling policy followed by the scheduler determines the Grid system throughput and utilization of the resources in the grid. The goal of the scheduler is to maximize the system throughput and utilization of the resources by balancing the loads on them.

1.1 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 [2].

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. 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.

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 load balancing task scheduling method should be used which is distributable, scalable and fault tolerant [3,4,5]. Along with the effective utilization of the resources in the grid, the cost of the resources being utilized should also be minimized by the scheduling method.

1.2 PROBLEM STATEMENT

The aim of this dissertation work is to propose an improvement in ant-algorithm based load balancing dynamic task scheduling algorithm and also make it cost effective in complex grid environments where the jobs are assumed to be computation-intensive, that are totally independent with no communication between them.

1.3 ORGANIZATION OF THE REPORT

Including this introductory chapter, the dissertation is organized in 5 chapters. Chapter 2 contains the basics of grid computing. Chapter 3 describes the task scheduling on grid. It briefly explains the scheduler organization and state estimation which is required for adaptive scheduling. It also includes the basic ant-algorithm based task scheduling algorithm. Chapter 4 presents the proposed scheduling algorithm with the enhancement to the ant algorithm. This chapter also includes the simulation architecture and description about GridSim which is the simulator used. Chapter 5 contains the discussions of various experiment results and comparison graphs of different scheduling strategies. Finally, Chapter 6 includes concluding remarks and scope for future work.

Chapter 2

ABOUT GRID COMPUTING

2.1 Grid concepts and components [3]

In this section, the various grid concepts, components, and terms are introduced in more detail.

2.1.1 Types of resources

A grid is a collection of machines, sometimes referred to as "nodes," "resources," "members," "donors," "clients," "hosts," "engines," and many other such terms. They all contribute any combination of resources to the grid as a whole. Some resources may be used by all users of the grid while others may have specific restrictions.

Computation

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are three primary ways to exploit the computation resources of a grid. The first and simplest is to use it to run an existing application on an available machine on the grid rather than locally. The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors. The third is to run an application that needs to be executed many times, on many different machines in the grid.

Storage

The second most common resource used in a grid is data storage. A grid providing an integrated view of data storage is sometimes called a "data grid". Each machine on the grid usually provides some quantity of storage for grid use, even if temporary. Storage can be memory attached to the processor or it can be "secondary storage" using hard disk drives or other permanent storage media.

Communications

The rapid growth in communication capacity among machines today makes grid computing practical, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, it should not be a surprise that another important resource of a grid is data communication capacity. This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. External communication access to the Internet, for example, can be valuable when building search engines

Software and licenses

The grid may have software installed that may be too expensive to install on every grid machine. Using a grid, the jobs requiring this software are sent to the particular machines on which this software happens to be installed.

2.1.2 Jobs and applications

Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing "application" or "job.". Applications may be broken down into any number of individual jobs. Those, in turn, can be further broken down into "sub jobs."

2.1.3 Scheduling, reservation, and scavenging

The grid system is responsible for sending a job to a given machine to be executed. The advanced grid systems would include a job "scheduler" of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid.

In a "scavenging" grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job which is satisfied by the machine's resources.

As a further step, grid resources can be "reserved" in advance for a designated set of jobs. This is done to meet deadlines and guarantee quality of service. Thus, various combinations of scheduling, reservation, and scavenging can be used to more completely utilize the grid.

2.2 Grid computing capabilities [3]

The following are the things that are possible by implementing a grid.

2.2.1 Exploiting underutilized resources

The function of the grid is to better balance resource utilization. The easiest use of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid.

An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid-enabled, they can be moved to underutilized machines during such peaks.

2.2.2 Parallel CPU capacity

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others. The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts.

2.2.3 Applications

There are many factors to consider in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. Furthermore, there are no practical tools for transforming arbitrary applications to exploit the parallel capabilities of a grid. There are some practical tools that skilled application designers can use to write a parallel grid application. However, automatic transformation of applications is a science in its infancy.

2.2.4 Access to additional resources

In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity. For example, if a user needs to increase his total bandwidth to the Internet to implement a data mining search engine, the work can be split among grid machines that have independent connections to the Internet.

2.2.5 Resource balancing

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid-enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization. This feature can prove invaluable for handling occasional peak loads of activity in parts of an larger organization. This can happen in two ways:

- An unexpected peak can be routed to relatively idle machines in the grid.
- If the grid is already fully utilized, the lowest priority work being performed on the grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

2.2.6 Management

The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more disperse IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization.

2.3 Grid Architecture [5]

Virtual Organizations (VOs) enable disparate groups of organizations and/or individuals to share resources in a controlled fashion, so that members may collaborate to achieve a shared goal. Grid technologies comprise protocols, services, and tools that address the challenges that arise when we seek to build scalable VOs. These technologies include security solutions that support management of credentials and policies when computations span multiple institutions; resource management protocols and services that support secure remote access to computing and data resources and the co-allocation of multiple resources; information query protocols and services that provide configuration and status information about resources, organizations, and services; and data management services that locate and transport datasets between storage systems and applications.

The figure 2.3 illustrates a categorization that we have found useful when explaining the roles played by various Grid technologies.

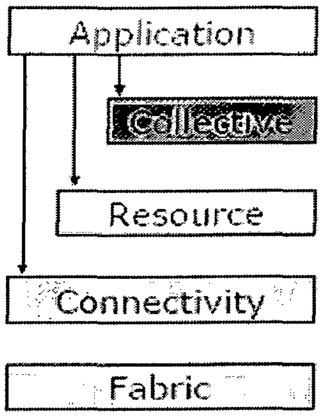


Figure *2.3 Grid architecture*

In the *Fabric*, we have the resources that we wish to share: computers, storage systems, data, catalogs, etc.

The *Connectivity* layer provides communication and authentication services needed to communicate with these resources.

Resource protocols (and, as in each layer, associated APIs) negotiate access to individual resources.

Collective protocols, APIs, and services are concerned with coordinating the use of multiple resources, and finally *application toolkits and Applications* themselves are defined in terms of services of these various kinds.

2.4 Grid software components [3]

The following are some of the key components that must be discussed before designing a grid computing architecture.

2.4.1 Management components

Any grid system has some management components. First, there is a component that keeps track of the resources available to the grid and which users are members of the grid. This information is used primarily to decide where grid jobs should be assigned.

Second, there are measurement components that determine both the capacities of the nodes on the grid and their current utilization rate at any given time. This information is used to schedule jobs in the grid.

2.4.2 Donor software

Each machine contributing resources typically needs to enroll as a member of the grid and install some software that manages the grid's use of its resources. Usually, some sort of identification and authentication procedure must be performed before a machine can join the grid. A Certificate Authority can be used to establish the identity of the donor machine as well as the users and the grid itself.

2.4.3 Submission software

Usually any member machine of a grid can be used to submit jobs to the grid and initiate grid queries. However, in some grid systems, this function is implemented as a separate component installed on "submission nodes" or "submission clients." When a grid is built using dedicated resources rather than scavenged resources, separate submission software is usually installed on the user's desktop or workstation.

2.4.4 Distributed grid management

Larger grids may have a hierarchical or other type of organizational topology usually matching the connectivity topology. That is, machines locally connected together with a LAN form a "cluster" of machines. The grid may be organized in a hierarchy consisting of clusters of clusters. The work involved in managing the grid is distributed to increase the scalability of the grid. The collection and grid operation and resource data as well as job scheduling is distributed to match the topology of the grid. For example, a central job scheduler will not schedule a submitted job directly to the machine which is to execute it. Instead the job is sent to a lower level scheduler which handles a set of machines (or further clusters). The lower level scheduler handles the assignment to the specific machine. Similarly, the collection of statistical information is distributed. Lower level clusters receive activity information from the individual machines, aggregate it, and send it to higher level management nodes in the hierarchy.

2.4.5 Schedulers

Most grid systems include some sort of job scheduling software. This software locates a machine on which to run a grid job that has been submitted by a user. In the simplest cases, it may just blindly assign jobs in a round-robin fashion to the next machine matching the resource requirements. However, there are advantages to using a more advanced scheduler. More advanced schedulers will monitor the progress of scheduled jobs managing the overall work-flow.

2.4.6 Communications

A grid system may include software to help jobs communicate with each other. For example, an application may split itself into a large number of sub jobs. Each of these sub jobs is a separate job in the grid. However, the application may implement an algorithm that requires that the sub jobs communicate some information among them. The sub jobs need to be able to locate other specific sub jobs, establish a communications connection with them, and send the appropriate data. The open standard Message Passing Interface (MPI) and any of several variations is often included as part of the grid system for just this kind of communication.

2.4.7 Observation, management, and measurement

We mentioned above the schedulers react to current loads on the grid. Usually, the donor software will include some tools that measure the current load and activity on a given machine using either operating system facilities or by direct measurement. This software is sometimes referred to as a "load sensor."

Such measurement information is useful not only for scheduling, but also for discovering overall usage patterns in the grid.. Also, measurement information about specific jobs can be collected and used to better predict the resource requirements of that job the next time it is run. The better the prediction, the more efficiently the grids workload can be managed.

Chapter 3

TASK SCHEDULING ON GRID

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.

3.1 Classification of Scheduling & load balancing [2]

Two important aspects of any wide area network scheduler are its transfer [12] and location policies. The transfer policy decides if there is need to initiate load balancing across the system, and is typically threshold based. Using workload information, it determines when a node becomes eligible to act as a sender (transfer a job to another node) or as a receiver (retrieve a job from another node). The location policy selects a partner node for a job transfer transaction. In other words, it locates complementary nodes to/from which a node can send/receive workload to improve overall system performance.

Location policies can be broadly classified as sender-initiated [9], receiver-initiated [9, 10], or symmetrically-initiated [11, 12, 13]. Sender-initiated policies are those where heavily-loaded nodes search for lightly-loaded nodes while receiver-initiated policies are those where lightly-loaded nodes search for suitable senders. Symmetrically-initiated policies combine the advantages of these two by requiring both sender and receiver to look for appropriate partners.

Load balancing policies can also be classified on the basis of how up-to-date each node's knowledge is about the state of the system. Dynamic [14, 15] policies make decisions based on the current system state and can rapidly adapt to workload fluctuations. On the other hand, policies that use static information and are not amenable to changes in the workload are known as static [8] policies. However, dynamic policies incur the overhead of communicating among the system nodes to keep them informed about the state of the system.

3.2 Scheduler organization [2]

The scheduling component of the Resource Management System (RMS) can be organized in three different ways as shown in Figure 3.2. In the centralized organization, there is only one scheduling controller that is responsible for the system-wide decision making. Such an organization has several advantages including easy management, simple deployment, and the ability to co-allocate resources. In a Grid RMS the disadvantages of this organization such as the lack of scalability, lack of fault-tolerance, and the difficulty in accommodating multiple policies outweigh the advantages.

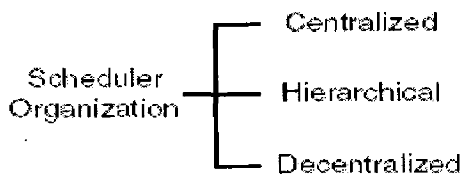


Fig3.2 Scheduler organization taxonomy.

The other two organizations, hierarchical and decentralized have more suitable properties for a Grid RMS scheduler organization. In a hierarchical organization, the scheduling controllers are organized in a hierarchy. One obvious way of organizing the controllers would be to let the higher level controllers manage larger sets of resources and lower level controllers manage smaller sets of resources. Compared with the centralized scheduling this mode of hierarchical scheduling addresses the scalability and fault-tolerance issues. It also retains some of the advantages of the centralized scheme such as co-allocation.

One of the key issues with the hierarchical scheme is that it still does not provide site autonomy and multi-policy scheduling. This might be a severe drawback for Grids because the various resources that participate in the Grid would want to preserve control over their usage to varying degrees. Many Grid resources would not dedicate themselves only to the Grid applications. Therefore hierarchical scheduling schemes should deal with dynamic resource usage policies.

The decentralized organization is another alternative. It naturally addresses several important issues such as fault-tolerance, scalability, site-autonomy, and multi-policy scheduling. However, decentralized organizations introduce several problems of their own some of them include management, usage tracking, and co-allocation. This scheme is expected to scale to large network sizes but it is necessary for the scheduling controllers to coordinate with each other via some form of resource discovery or resource trading protocols. The overhead of operation of these protocols will be the determining factor for the scalability of the overall system. Lack of such protocols may reduce the overhead but the efficiency of scheduling may also decrease.

3.3 State Estimation [2]

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.

Figure 3.3 shows the state estimation taxonomy.

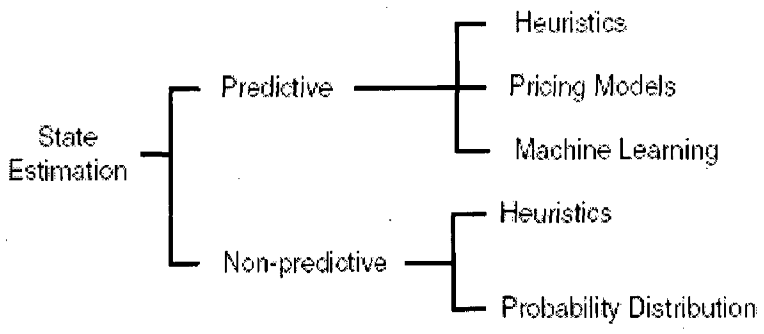


Fig3.3 State estimation taxonomy.

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.

3.4 Related Work

There are load balancing scheduling strategies [4, 5] which are mostly application specific and also they does not consider the cost factor of the resources. Minimizing the cost of resources being utilized is very important in real time grids.

3.4.1 Ant algorithm [7]

Ant algorithm 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, 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 the path. At last all the ants will choose the shortest path.

When the blind insects such as ants look for food, every moving ant lays some pheromone on the path. As soon as the ants get food, they return to their home by laying the pheromone on the same path on which they come. If the ants are served quickly at a food centre, then the pheromone from the home to the food centre increases quickly as more ants choose that path. This phenomenon is shown in the figure 3.4.

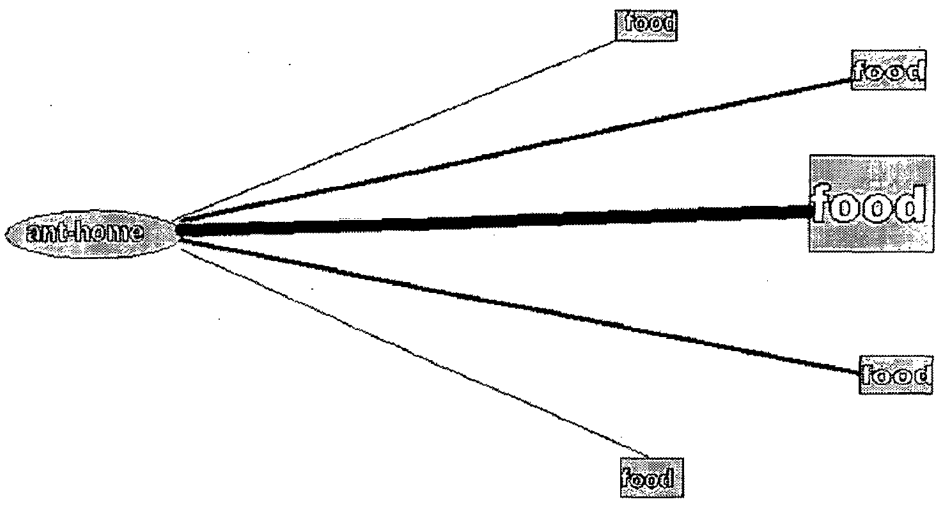


Fig 3.4 behavior of ants

This behavior of ants is suitable to be used in Grid computing task scheduling where the ant-home corresponds to the scheduler, ants correspond to tasks, and food sources correspond to the resources.

The algorithm has been successfully used to solve many NP problems, such as Traveling Salesman Problem (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.

3.4.2 Ant-algorithm based task scheduling algorithm [7]

1. When a resource 'j' enrolls the Grid, it is asked to submit its performance parameters, No. of PE, MIPS of every PE etc. Resource monitor tests these parameters for validation, and initialize the links pheromone as:

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

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.

2. 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 as

$$\tau_j^{new} = \rho.\tau_j^{old} + \Delta\tau_j$$

$\Delta\tau_j$ is the change of pheromone on path from the schedule center to resource j;

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

$1 - \rho$ is evaporation of pheromone.

when task is assigned to resource j,

$\Delta\tau_j$ = - k, k is the compute and transfer quality of the task.

when task successfully returned from resource 'j',

$\Delta\tau_j$ = Ce x k, C_e is the encourage factor.

when task failed returned from resource 'j',

$\Delta\tau_j$ = Cp x k, C_p is the punish factor.

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

$$\rho_j^k(t) = \frac{[\tau_j(t)]^\alpha * [\eta_j(t)]^\beta}{\sum_u [\tau_u(t)]^\alpha * [\eta_u]^\beta} \quad \text{j, u are online resources}$$

$$= 0 \qquad \text{others}$$

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

$\eta_j(t)$ is the innate performance of the resource, that is $\tau_j(0)$.

α is the importance of the pheromone.

β is the importance of resource innate attributes.

The parameter values α = 0.5, β = 0.5, ρ = 0.8, Ce = 1.1, Cp = 0.8 will be taken for here.

This gives the probabilities of the resources in the grid which helps the task scheduler in scheduling.

4. The scheduler assigns a new task 'k' on to a resource 'j' at time't' with a probability equal to it's corresponding $\rho_j^k(t)$.

Chapter 4

DESIGN AND IMPLEMENTATION

4.1 Enhancement to the ant algorithm

The scheduler explained in [7] 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. To avoid this, If the tasks are always scheduled to the resource with high possibility, then the load on the resource may be increased and the jobs may be kept waiting in the queue waiting for the resource to be free even though the other resources are free.

So, the algorithm can be modified in such a way that if the difference between the possibility of the resource selected for execution of a task using ant-algorithm and the possibility of the resource with highest possibility is less than certain threshold, then the task will be scheduled to the resource selected by the ant-algorithm. Otherwise, the scheduler selects another resource and the above procedure will be repeated. The selection of the threshold plays an important role. Since this modified algorithm takes the resources with highest possibility into consideration, although the processing time is reduced, the processing cost of the tasks may increase when compared to that of ant-algorithm. The inclusion of price factor into this modified algorithm which is the proposed algorithm minimizes the total execution time as well as the processing cost of the tasks. Also the algorithm is made into a de-centralized scheduling algorithm which can be executed on the scheduler of the node on which the task is submitted. The schedulers of every node run independently except that they fetch the status of the resources in the grid from a centralized resource state manager.

4.2 Proposed task scheduling algorithm

This algorithm is decentralized and improved version with the inclusion of cost factor of ant-algorithm based task scheduling.

1. When a resource 'j' enrolls the Grid, it is asked to submit its performance parameters, No. of PE, MIPS of every PE etc. Resource monitor tests these parameters for validation, and initialize the links pheromone as:

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

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.

2. 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 as

$\tau_j^{new} = \rho.\tau_j^{old} + \Delta\tau_j$

$\Delta\tau_j$ is the change of pheromone on path from the schedule center to resource j;

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

$1 - \rho$ is evaporation of pheromone

when task is assigned to resource j,

$\Delta\tau_j = -k$, k is the compute and transfer quality of the task.

when task successfully returned from resource 'j',

$\Delta\tau_j$ = Ce x k, C_e is the encourage factor.

when task failed returned from resource 'j',

$\Delta\tau_j$ = Cp x k, C_p is the punish factor.

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

$$\rho_j^k(t) = \frac{[\tau_j(t)]^\alpha * [\eta_j(t)]^\beta}{\sum_u [\tau_u(t)]^\alpha * [\eta_u]^\beta} \quad \text{j, u are online resources}$$

$$= 0 \quad \text{others}$$

$\tau_j(t)$ is the pheromone intensity on the path from schedule center to resource j, $\eta_j(t)$ is the innate performance of the resource, that is $\tau_j(0)$.

α is the importance of the pheromone.

β is the importance of resource innate attributes.

The parameter values $\alpha = 0.5, \beta = 0.5, \rho = 0.8$,

Ce = 1.1, Cp = 0.8 will be taken for here.

This gives the probabilities of the resources in the grid which helps the task scheduler in scheduling.

4. The scheduler schedules the task 'k' on the same resource 'i' which belong to the node on which the task is submitted when $(\rho_h^k(t) - \rho_i^k(t)) \leq T_h$. Otherwise steps 5,6,7 get executed.

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

'T_h' is the threshold which will be taken as 1/ (no. of resources).

5. The scheduler finds a resource 'j' on which a new task 'k' can be scheduled at time 't' with a probability equal to it's corresponding $\rho_j^k(t)$ until $(\rho_h^k(t) - \rho_j^k(t)) \leq T_h$.

6. The scheduler finds a resource 'i' taking one resource at a time in the ascending order of C_i / ($MIPS_i$) until $|\rho_j^k(t) - \rho_i^k(t)| \leq T_i * \ell$

where C_i is the cost of the resource 'i' per second

$MIPS_i$ is the total MIPS of the resource 'i'

$T_i = T_h - (\rho_h^k(t) - \rho_j^k(t))$ and

'ℓ' is the cost reduction factor which is selected by the grid user who submits the task. $0 \leq \ell \leq 1$.

7. The scheduler schedules the new task 'k' on to the resource 'i' by sending the task to the node of which resource 'i' belongs to.

4.3 Simulation model

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. The tasks are assumed to be computational intensive and are totally independent with no communication between them. There are five important modules in the schedule center; the architecture of the scheduler center of a node can be expressed as in the figure 4.3 below. Each node denotes a client or a Grid resource, and each edge denotes a link between nodes.

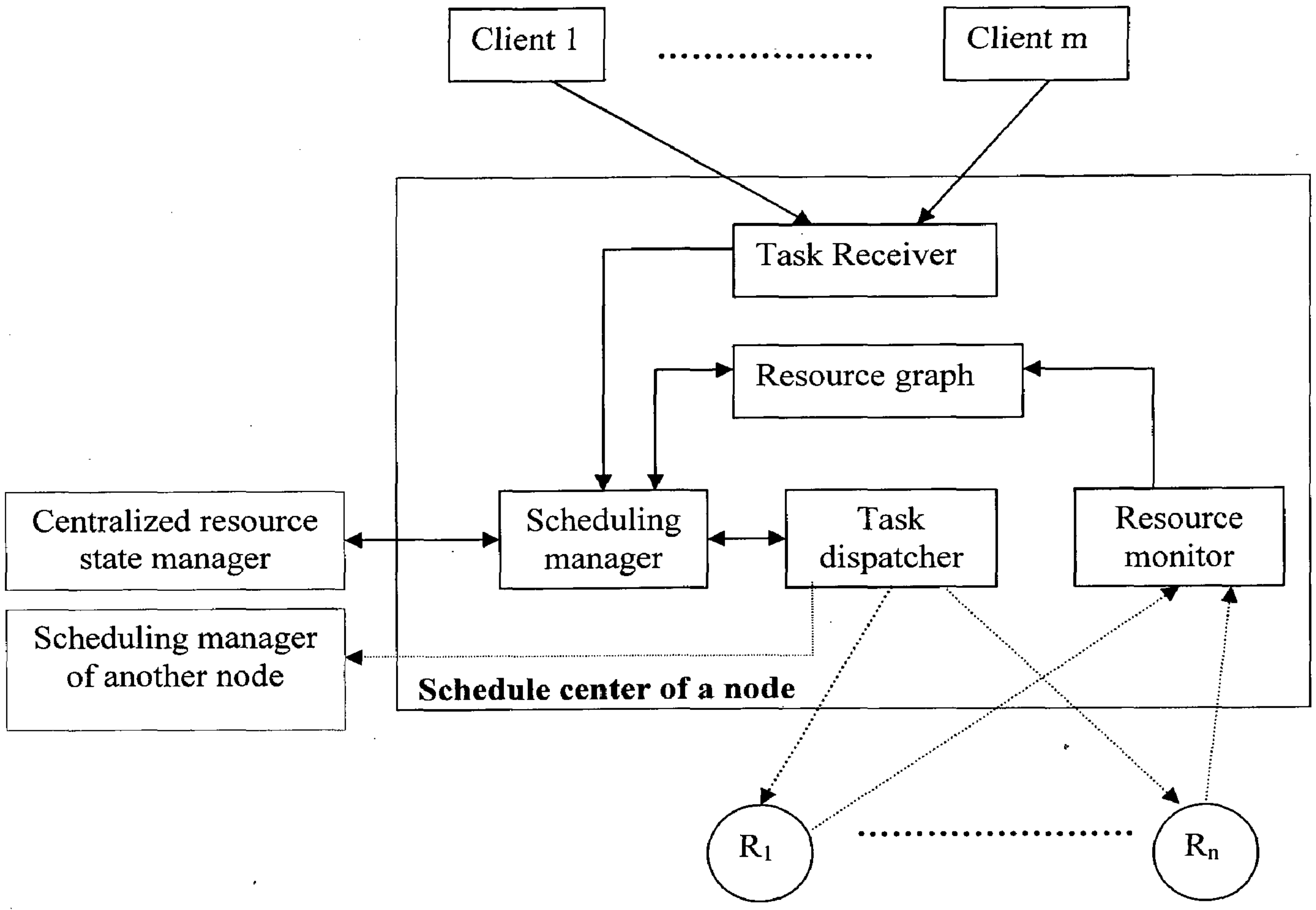


Fig4.3 Grid simulation architecture

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 to the task receiver. The description is about the work load of the application, communication load, and time limit, etc.

- 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, resources states and user requirement, then delivers the scheme to task dispatcher and inform resource monitor. This is the most important and complex scheduling in the schedule center, there are many scheduling strategies can be put in the scheduling manager. We design the ant algorithm based strategy, and will discuss it in the following section.
- Task dispatcher delivers the task to the selected resource, and gives transfer delay and actual task assignment result to the resource monitor. The selected resource can be on the same node or on different node. If it is on different node, the request for the resource will be sent to the scheduling manager of the node in which the selected resource belongs to. The dispatcher maintains a scheduled task queue; record the assigned resource name, application work load, 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 of every resource and revises the resource graph.
- A graphic is used 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.).

This ant algorithm for task scheduling in grid computing will be put in the scheduling manager of the simulation center.

4.4 Simulation Environment

Gridsim Simulator

GridSim[16] is a Java-based toolkit for modeling, and simulation of Grid resources, and application scheduling. 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. Other than GridSim, there are few tools available for application scheduling simulation in Grid computing environments, such as Bricks, MicroGrid and SimGrid toolkit.

GridSim entities are namely, user, broker, resource, information service, statistics, shutdown, and report writer. Once GridSim starts, the resource entities register themselves with the Grid Information Service (GIS) entity. The broker entity queries GIS entity for resource discovery, based on the user entity's request. The GIS entity returns a list of registered resources, and their contact details. The broker entity queries the resources for resource configuration, and properties. They respond with dynamic information such as resources cost, capability, availability, load, and other configuration parameters. Broker entity selects the appropriate resources, and sends user jobs (gridlets) to those resources for execution. The resources send back the processed gridlets to the I/O queue of the broker entity. Finally, the user will collect the processed gridlets from the I/O queue. **Figure 3.4** depicts the flowchart of the GridSim simulation [12] used in the project simulation

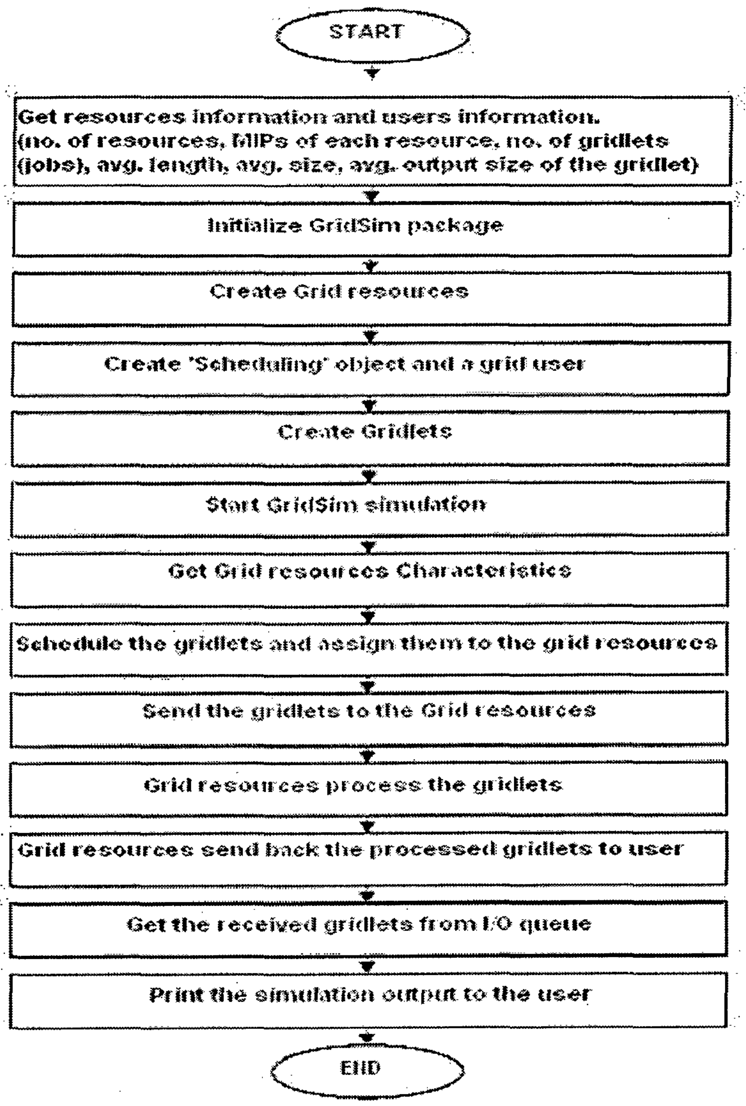

START
Get resources information and users information. (no. of resources, MIPs of each resource, no. of gridlets (jobs), avg. length, avg. size, avg. output size of the gridlet)
Initialize GridSim package
Create Grid resources
Create 'Scheduling' object and a grid user
Create Gridlets
Start GridSim simulation
Get Grid resources Characteristics
Schedule the gridlets and assign them to the grid resources
Send the gridlets to the Grid resources
Grid resources process the gridlets
Grid resources send back the processed gridlets to user
Get the received gridlets from I/O queue
Print the simulation output to the user
END

Fig 4.4 Process flow of GridSim simulation

4.5 Implementation

SOURCE CODE FILES:

***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. This information will be stored in a file called 'info.tmp'. This stored information in the file is used for multiple simulations of scheduling algorithms for the same input.

***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 the file 'info.tmp'. This file also contains some of the results of the simulations like total execution times, total processing costs.

ResultDisplay.java: This file contains the GUI code to display the results obtained after simulation. This displays the user-wise information as well as the resource-wise information. In user-wise information, the resource on which the task is executed, the time taken to execute the task, the cost of completing the task etc, will be there. In resource-wise information, the no. of tasks the resource executes, the utilization time etc. will be there.

Comparison.java: This contains the GUI code to display the comparison results after simulation of different scheduling algorithms. It compares the total execution time taken and cost of executing the total tasks on the grid for various scheduling strategies.

Graphs.java: This contains the GUI code to display the graphical information to compare the various scheduling algorithms performance using the multiple simulation results. The total execution time of tasks and the total cost of completing the tasks are taken on the y-axis while the no. of resources in the grid is taken on x-axis.

Scheduling.java: This file contains the core code of this work which implements the simulation of proposed task scheduling algorithm.

Grid resources information and Grid users information will be given as input to start the simulation. Then the simulation starts step by step as shown in the flow diagram. 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 task creation for the user and also implements the scheduling of those tasks on to the appropriate resources that are registered to the grid. The tasks are created based on user specified parameters (total number of tasks, average MI of each task, 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 task is to be scheduled. Then, the scheduler at the node on which a new task is submitted determines whether the task can be scheduled on the same node or should be sent to other nodes. If any of the processors in the node are free and the condition is satisfied then it will be scheduled on the same node. Otherwise, the scheduler selects a resource based on the proposed algorithm and schedules the task. When a task is

submitted to a resource, its pheromone value will be decreased by the compute and transfer quality of the task. The Grid resources process the received tasks and send back the processed tasks to the grid user.

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

Chapter 5

RESULTS & DISCUSSIONS

Simulations were conducted to analyze and compare the differences between various scheduling algorithms, in terms of processing times and processing costs. Resources R1 through R20 are used for these simulations. Users U1 to U40 submit tasks to the grid. The resource and user tasks characteristics taken for the simulation are shown in the table 5.1 and table 5.2 respectively.

Resource	No. of PEs	MIPS of	Communication rate (Mb/s)	Cost/sec
R1	1	50	10	1
R2	4	377	30	2
R3	2	380	10	1
R4	16	410	40	7
R5	4	410	20	2
R6	2	200	20	2
R7	6	410	20	3
R8	16	410	50	7
R9	4	377	20	2
R10	16	410	50	7
R11	1	50	10	1
R12	4	377	30	2
R13	2	380	10	1
R14	16	410	40	7
R15	4	410	20	2
R16	2	200	20	2
R17	6	410	20	3
R18	16	410	50	7
R19	4	377	20	2
R20	16	410	50	7

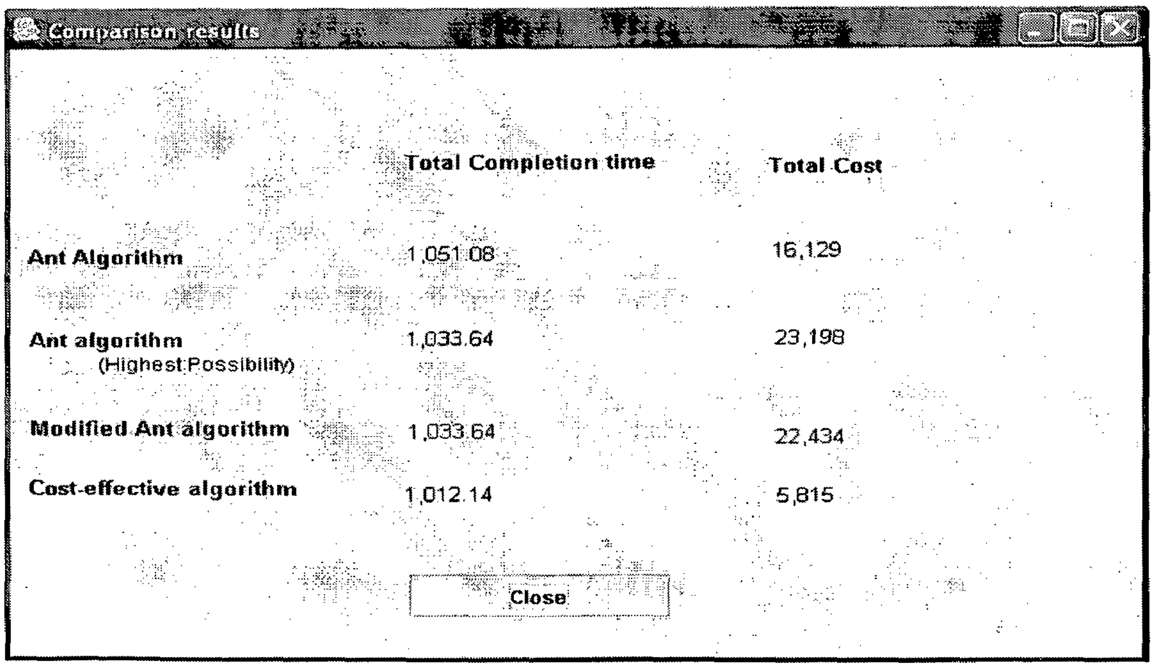
Table 5.1 Resource characteristics

User	Job length (MI)	Job size (Mb)	Output size (Mb)
U1&21	75000	100	50
U2&22	65000	80	40
U3&23	8000	15	10
U4&24	5000	15	10
U5&25	70000	80	35
U6&26	75000	85	50
U7&27	200	10	10

U8&28	45000	90	40
U9&29	3000	30	10
U10&30	100	10	10
U11&31	70000	95	50
U12&32	2200	50	20
U13&33	500	15	10
U14&34	75000	100	50
U15&35	32000	40	25
U16&36	68000	80	45
U17&37	7000	20	10
U18&38	300	10	10
U19&39	1000	10	10
U20&40	54000	60	40

Table 5.2 User task characteristics

From fig 5.1, 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. The results of ant algorithm (highest possibility) are the results with the scheduling algorithm in which a new task will always be scheduled on to the resource with highest possibility.



	Total Completion time	Total Cost
Ant Algorithm	1,051.08	16,129
Ant algorithm (Highest Possibility)	1,033.64	23,198
Modified Ant algorithm	1,033.64	22,434
Cost-effective algorithm	1,012.14	5,815

Fig 5.1 Results of different algorithms.

The following are the results of simulation of various algorithms for the 40 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. Here the ℓ of all the user tasks is taken as 1. The values in the graphs are taken as the mean of 20 simulation values.

5.2.1 Experiment 1: Simulation results of various scheduling strategies

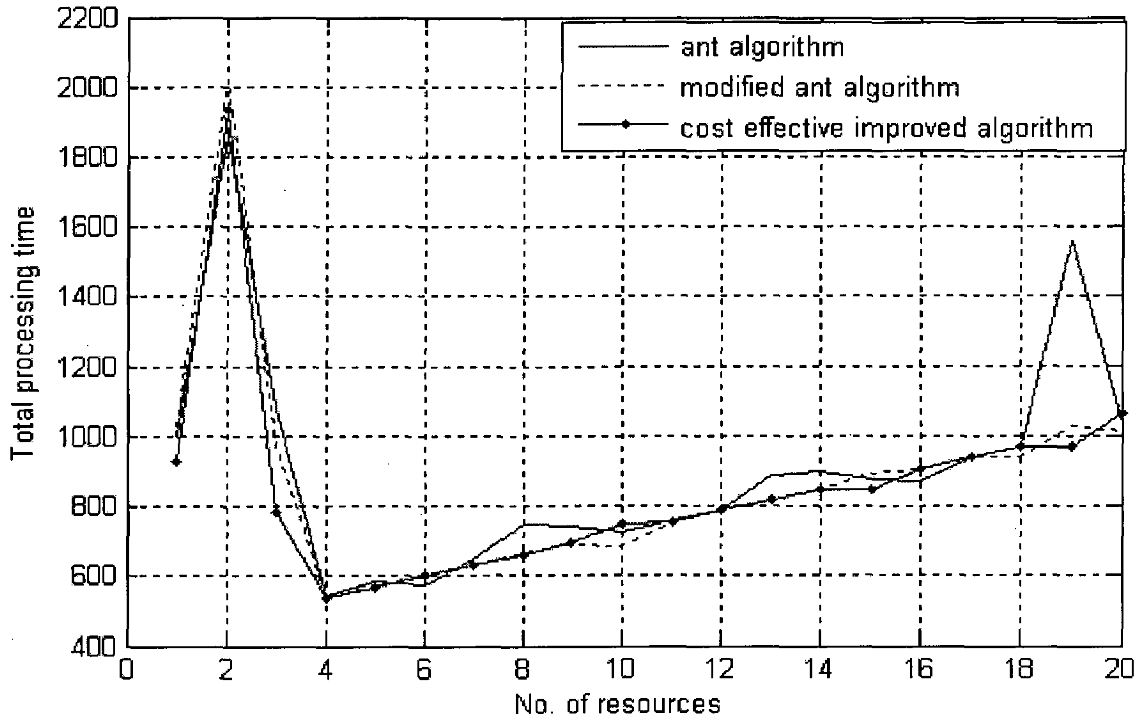


Fig 5.2 No. of resources Vs total processing time

As shown in the Fig 5.2, the processing time in the modified ant-algorithm (up to step 5 in the algorithm given in Chapter 4) and the developed algorithm which is inclusive of price factor is reduced when compared to that of ant algorithm.

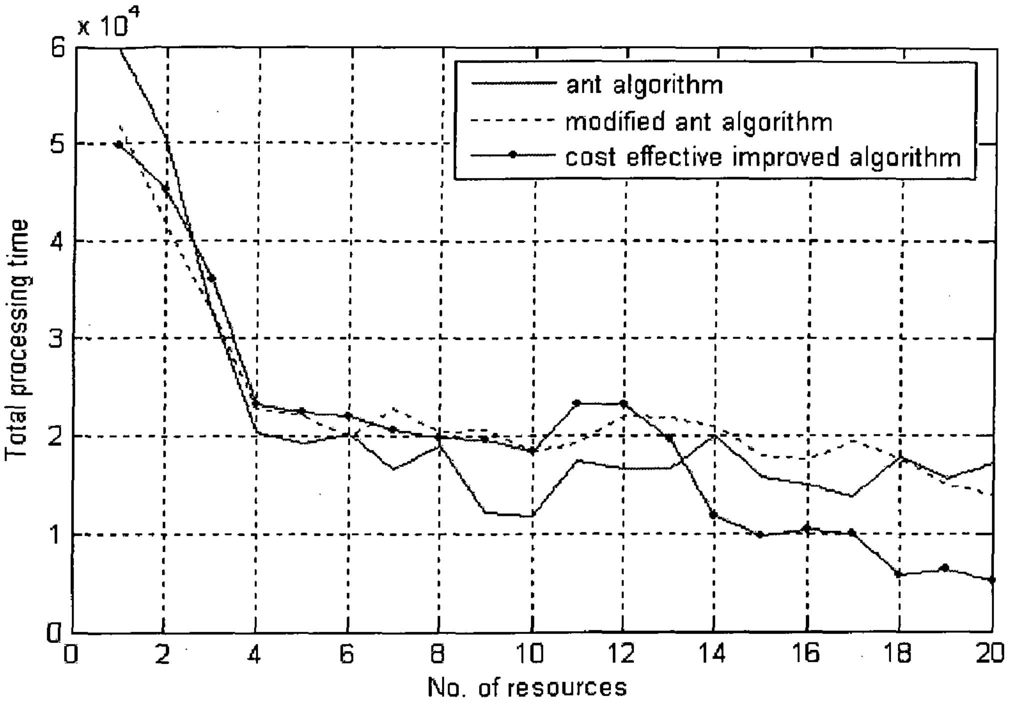


Fig 5.3 No. of resources Vs total processing time

As shown in the Fig 5.3, the processing costs in the developed algorithm is lot more controlled when compared to that of modified ant-algorithm in which the price factor is not included.

5.2.2 Experiment 2: Simulation results for different threshold values in the algorithm

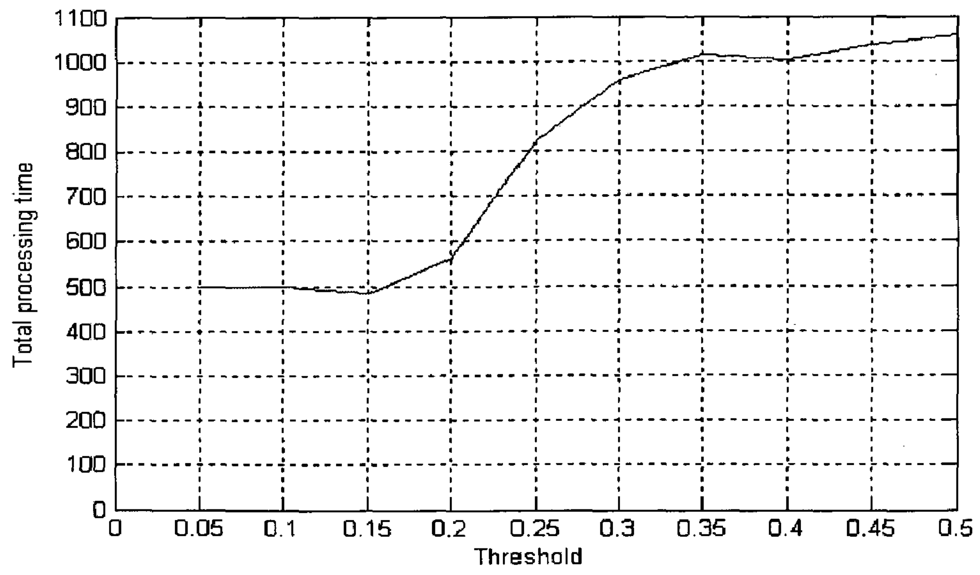


Fig 5.4 Threshold Vs total processing time

Fig 5.4 shows the total processing times for different values of threshold in the scheduling algorithm. If the threshold value increases certain value, then the algorithm results will be similar to that of ant-algorithm. The above are the results for 20 resources and 40 tasks.

5.2.3 Experiment3: Simulation results for different 'ℓ' values in the algorithm

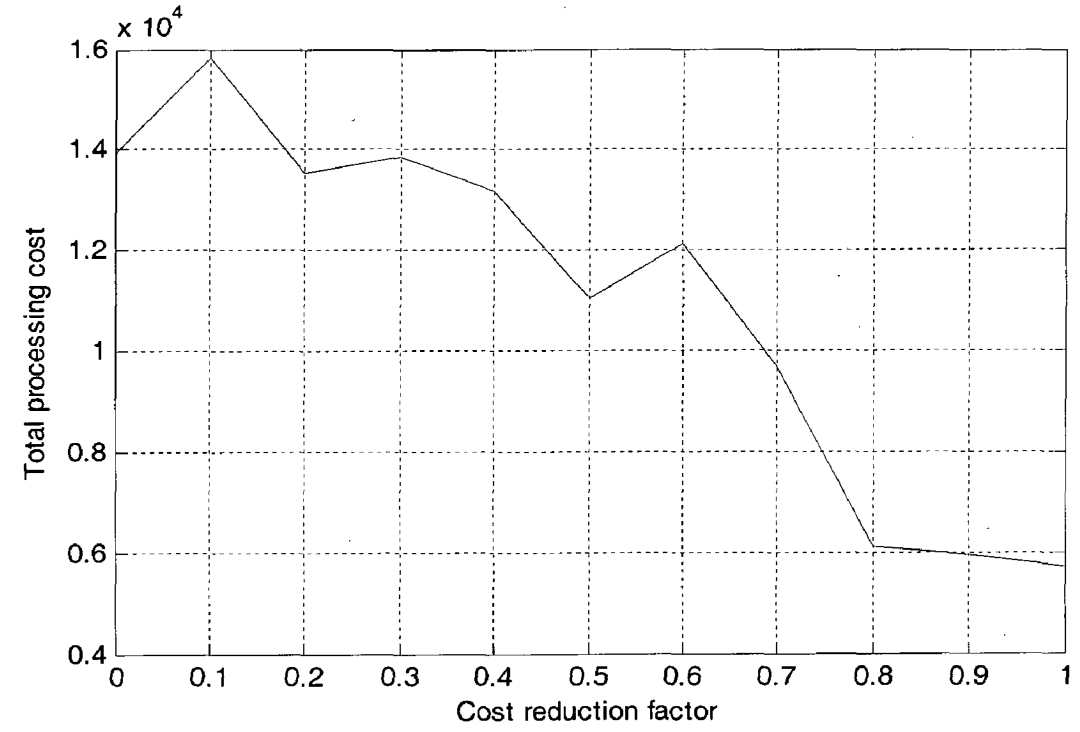


Fig 5.5 Cost Reduction Factor Vs total processing cost

From the Fig 5.5, it is shown that as the 'ℓ' value increases, the control on processing cost also increases. The results of scheduling algorithm with 'ℓ' value 0 are similar to the algorithm with out inclusion of price factor (up to step 5 of the algorithm in Chapter 4). The above are the results for 20 resources and 40 tasks.

Chapter 6

CONCLUSION

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 then 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 proposed scheduling strategy results in increased performance in terms of low processing time and low processing cost if it is applied to a Grid application with a large number of coarse granularity tasks like parameter sweeps application. This works effectively in minimizing both the processing time of the tasks as well as the processing cost of the tasks depending upon the CRF value chosen by the grid user who submits the task.

Future work would involve developing a more comprehensive distributive scheduling system that takes into account the hierarchy of clusters of resources. And also to reduce the transmission costs and delays in the applications with large number of light weight jobs, job grouping can be done before load balancing the jobs on to the resources.

References

[1] Mark Baker, Rajkumar Buyya, and Domenico Laforenza, "Grids and Grid Technologies for Wide-Area Distributed Computing", *Software: Practice and Experience (SPE) Journal,* Wiley Press, USA, 2002.

[2] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, "A Taxonomy and survey of Grid Resource Management Systems for Distributed Computing", *Software: Practice and Experience (SPE) journal,* Wiley Press, USA, 2001.

[3] Bart Jacob, Luis Ferreira, Norbert Bleberstein, Candice Gllzean, Jean-Yves Girard, Roman Strachowski, Seong(Steve) Yu, "Enabling applications for grid computing with Globus", *International Business Machines corporation Red Books 2003,* http://www.ibm/redbooks

[4] Quinn Snell, Kevin Tew, Joseph Ekstrom, Mark Clement, "An Enterprise-Based Grid Resource Management System", *Proceedings of the 11th IEEE International Symposium on High performance Distributed computing HPDC-11* 2002 (HPDC'02).

[5] F. De Turck, S. Vanhaste, B. Volckaert, P. Demeester, "A generic middleware-based platform for scalable cluster computing", *Future Generation Computer Systems*, vol. 18, 2002, pp.549-560.

[6] Ron Oldfield, David Kotzl, "Armada: a parallel I/O framework for computational grids", *Future Generation Computer Systems,* vol 18, 2002, pp.501-523.

[7] Xhihong Xu, Xiangdan Hou, Jizhou Sun, "Ant Algorithm based task scheduling in Grid Computing", *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, vol. 2, 2003, pp. 1107-1110, Montreal, May 2003.

[8] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I.Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *Journal of Parallel and Distributed Computing.* 61(6):810-837, June 2001.

[9] D.L. Eager, E.D.Lazowska, and J.Zahorjan, "A comparison of receiver-initiated and sender-initiated adaptive load sharing", *Performance Evaluation*, 6(1):53-68, 1986.

[10] H. Linand, C.Raghavendra, "A dynamic load-balancing policy with a central job dispatcher (LBC), *IEEE Trans. Software Engineering,* 18(2):pp 148-158, February 1992.

[11] Y.Feng, D. Li.H. Wu, and Y.Zhang, "A dynamic load balancing algorithm based on distributed database system, *Proceedings of 4th International Conference on High*

Performance Computing in the Asia-Pacific Region, pages 949-952, Beijing, China, May 2000.

[12] N. Shivaratri, P. Krueger and M. Singhal, "Load distributing for locally distributed systems, *IEEE Computer,* 25(12): pp 33-44, December 1992.

[13] M. Willebeek-LeMairand, A.Reeves, "Strategies for dynamic load balancing on highly parallel computers, *IEEE Trans. Parallel and Distributed Systems,* 9(4): pp 979-993, September 1993.

[14] T.Tzenand, L. Ni, "Trapezoid self-scheduling: A practical scheduling scheme for parallel computers", *IEEE Trans. Parallel and Distributed Systems,* 4(1): pp 87-98, January 1993.

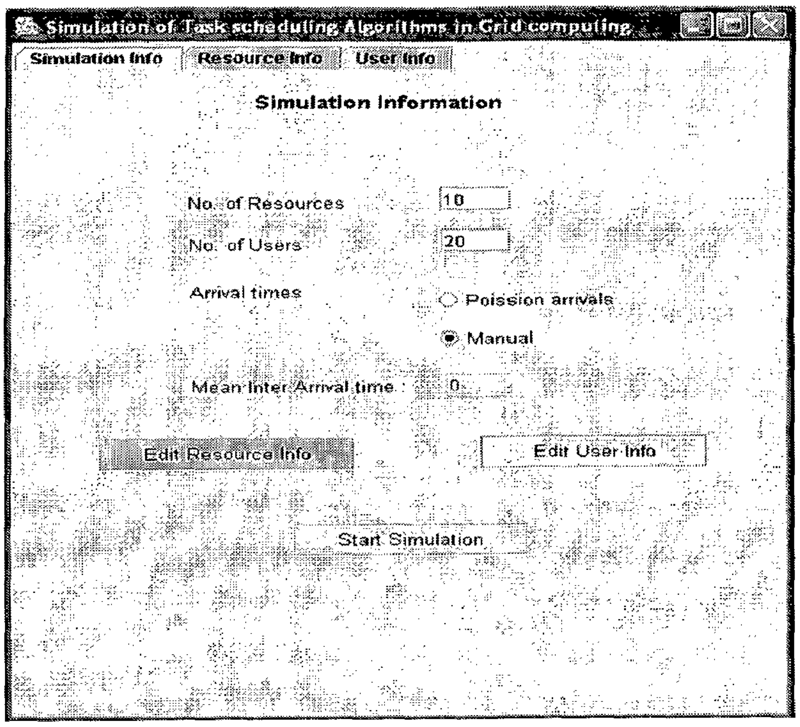
[15] J.Torrellas, A. Tucker, and A.Gupta, "Evaluating the performance of cache-affinity scheduling in shared-memory multi-processors, *Journal of Parallel and Distributed Computing,* 24(2): 139-151, February 1995.

[16] 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.

Appendix A

SNAPSHOTS

GUI for giving the basic input information like resource information and user information for the simulation:



Simulation of Task scheduling Algorithms in Grid computing

Simulation Info | Resource Info | User Info

Simulation Information

No. of Resources: 10

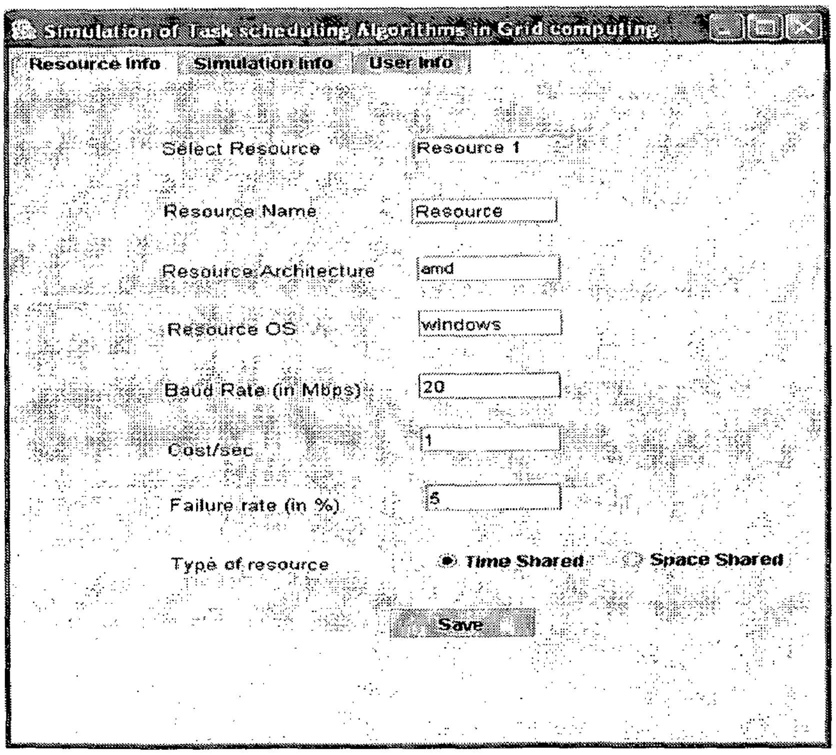
No. of Users: 20

Arrival times: ○ Poission arrivals [✓ Manual]

Mean Inter Arrival time : 0

Edit Resource Info | Edit User Info

Start Simulation



Simulation of Task scheduling Algorithms in Grid computing

Resource Info | Simulation Info | User Info

Select Resource: Resource 1

Resource Name: Resource

Resource Architecture: amd

Resource OS: windows

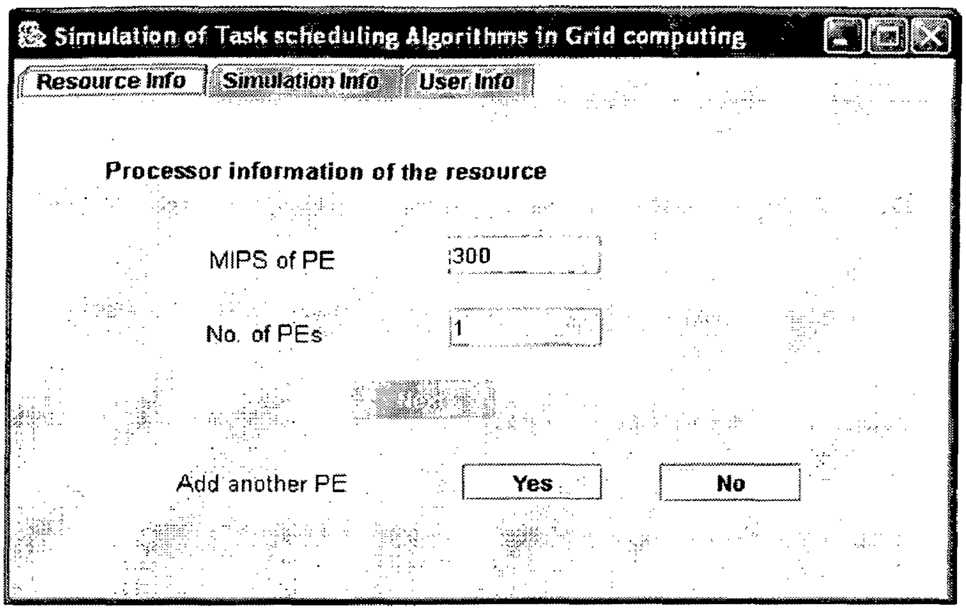
Baud Rate (in Mbps): 20

Cost/sec: 1

Failure rate (in %): 5

Type of resource: [✓ Time Shared] ○ Space Shared

Save


Simulation of Task scheduling Algorithms in Grid computing

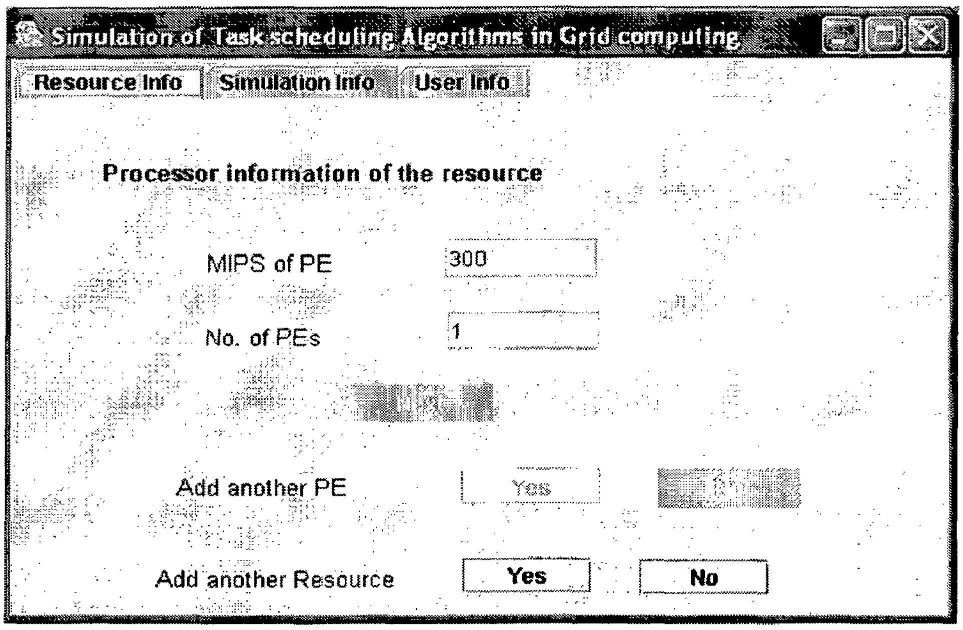
Resource Info | Simulation Info | User Info

Processor information of the resource

MIPS of PE 300

No. of PEs 1

Add another PE Yes No


Simulation of Task scheduling Algorithms in Grid computing

Resource Info | Simulation Info | User Info

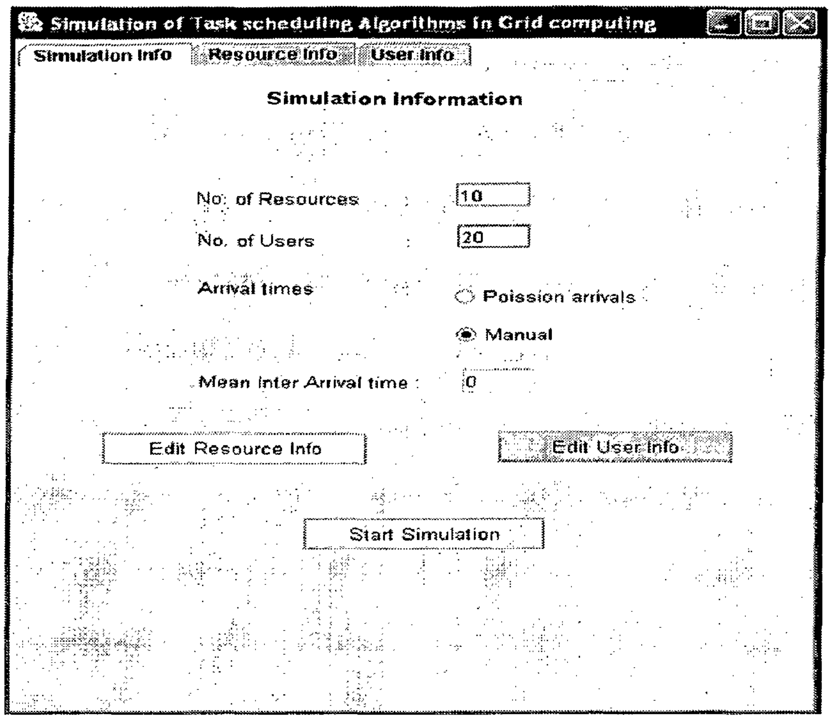
Processor information of the resource

MIPS of PE 300

No. of PEs 1

Add another PE Yes

Add another Resource Yes No



Simulation of Task scheduling Algorithms in Grid computing

Simulation Info | Resource Info | User Info

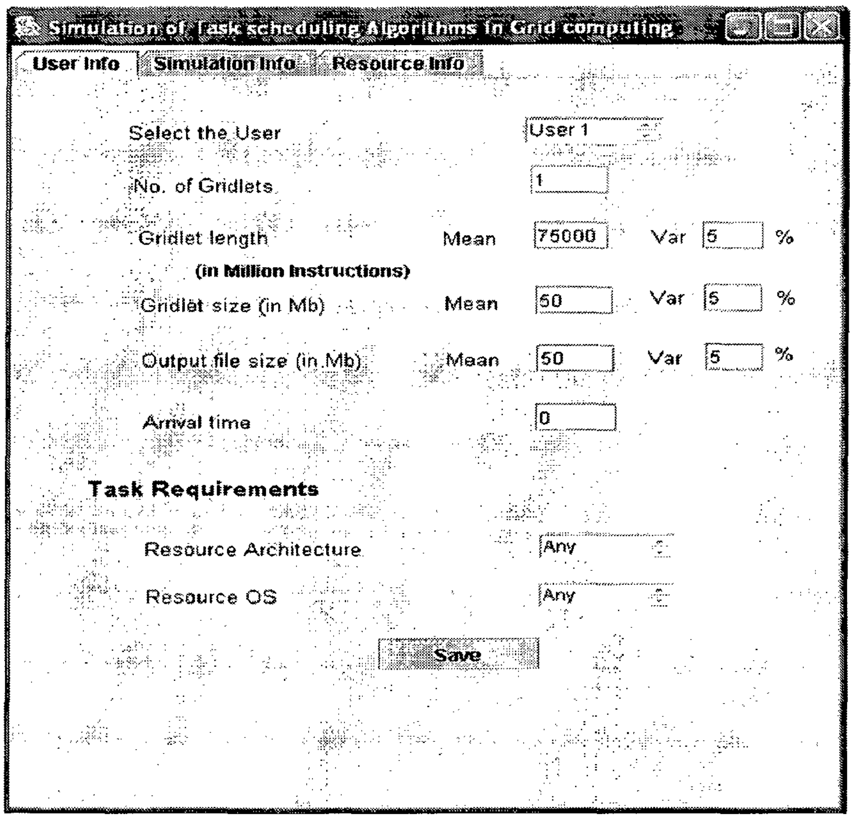
Simulation Information

No. of Resources : 10

No. of Users : 20

Arrival times : ○ Poission arrivals ● Manual

Mean Inter Arrival time : 0



Simulation of Task scheduling Algorithms in Grid computing

User Info | Simulation Info | Resource Info

Select the User: User 1

No. of Gridlets: 1

Gridlet length (in Million Instructions) Mean 75000 Var 5 %

Gridlet size (in Mb) Mean 50 Var 5 %

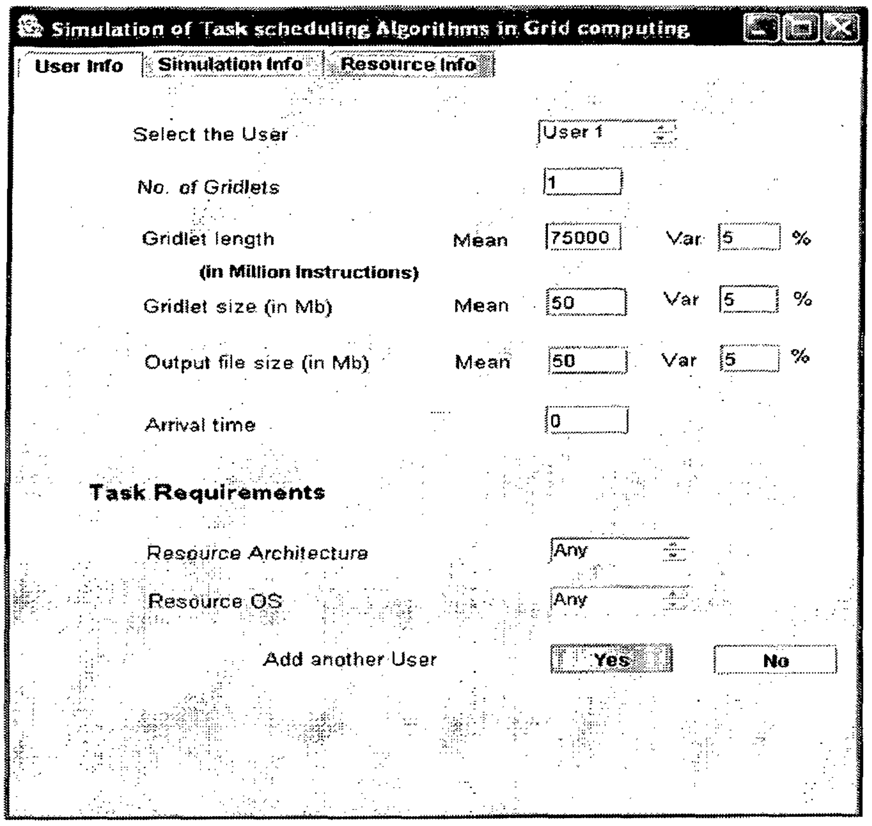
Output file size (in Mb) Mean 50 Var 5 %

Arrival time: 0

Task Requirements

Resource Architecture: Any

Resource OS: Any



Simulation of Task scheduling Algorithms in Grid computing

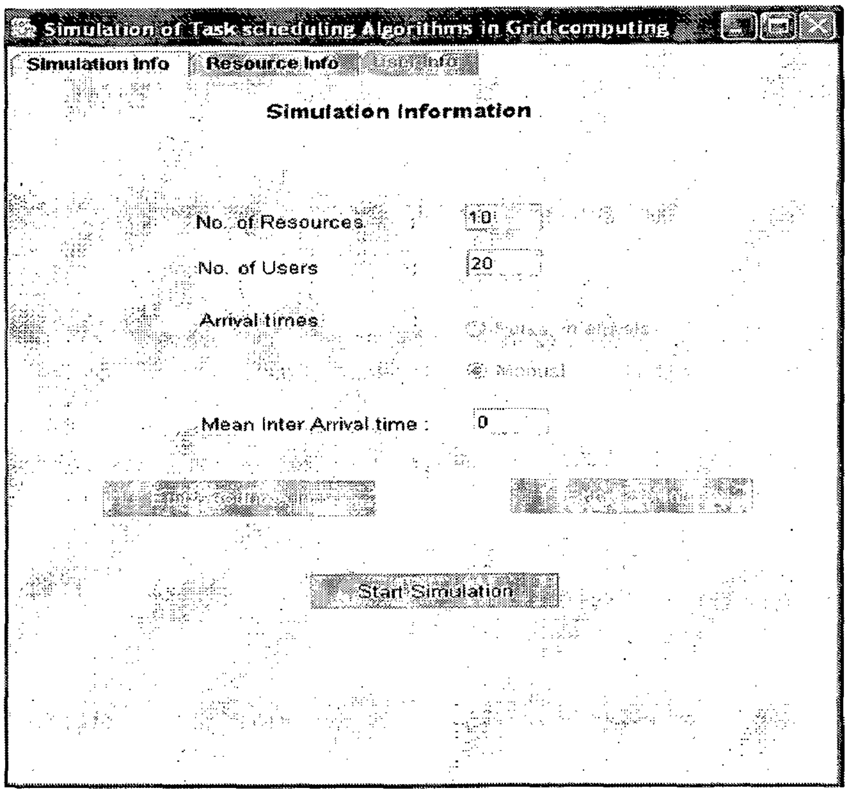
User Info | Simulation Info | Resource Info

Select the User	User 1		
No. of Gridlets	1		
Gridlet length (in Million Instructions)	Mean 75000	Var 5 %	
Gridlet size (in Mb)	Mean 50	Var 5 %	
Output file size (in Mb)	Mean 50	Var 5 %	
Arrival time	0		

Task Requirements

Resource Architecture	Any
Resource OS	Any

Add another User [✓ Yes] No



Simulation of Task scheduling Algorithms in Grid computing

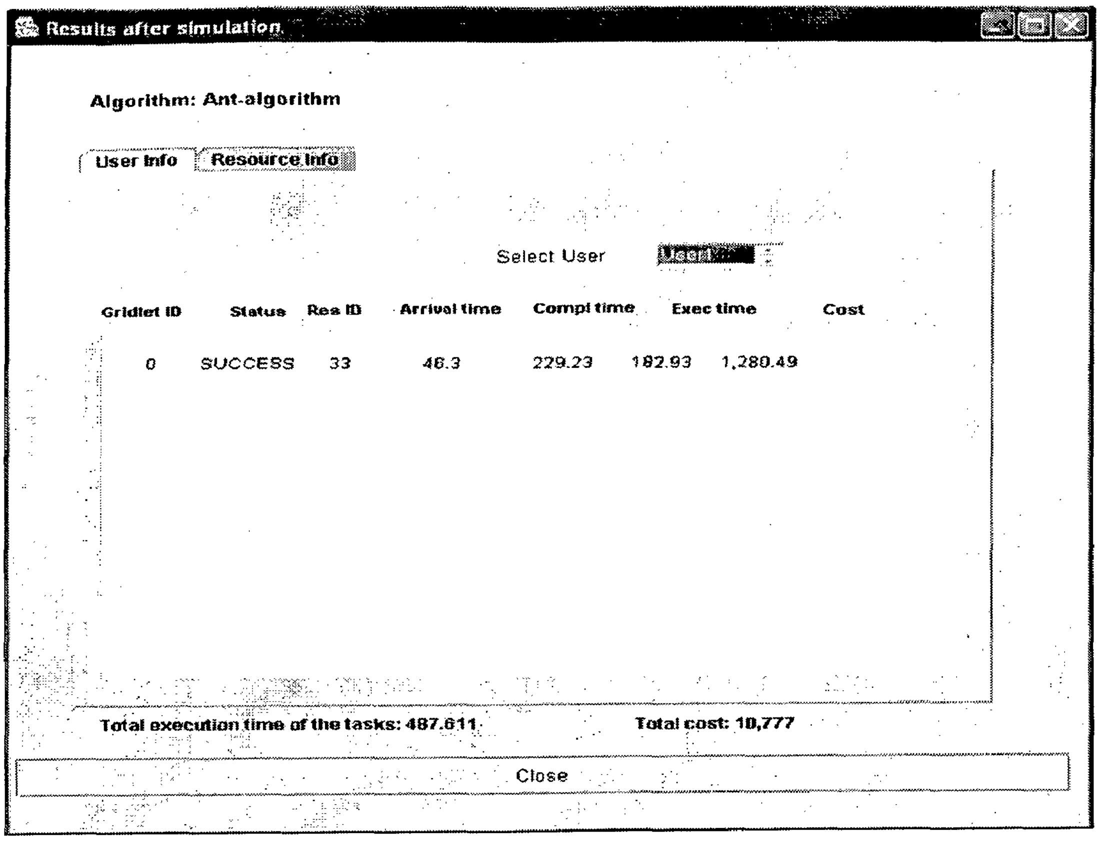
Simulation Info | Resource Info | User Info

Simulation Information

No. of Resources :	10
No. of Users :	20
Arrival times :	[illegible] [✓ Manual]
Mean Inter Arrival time :	0

Start Simulation

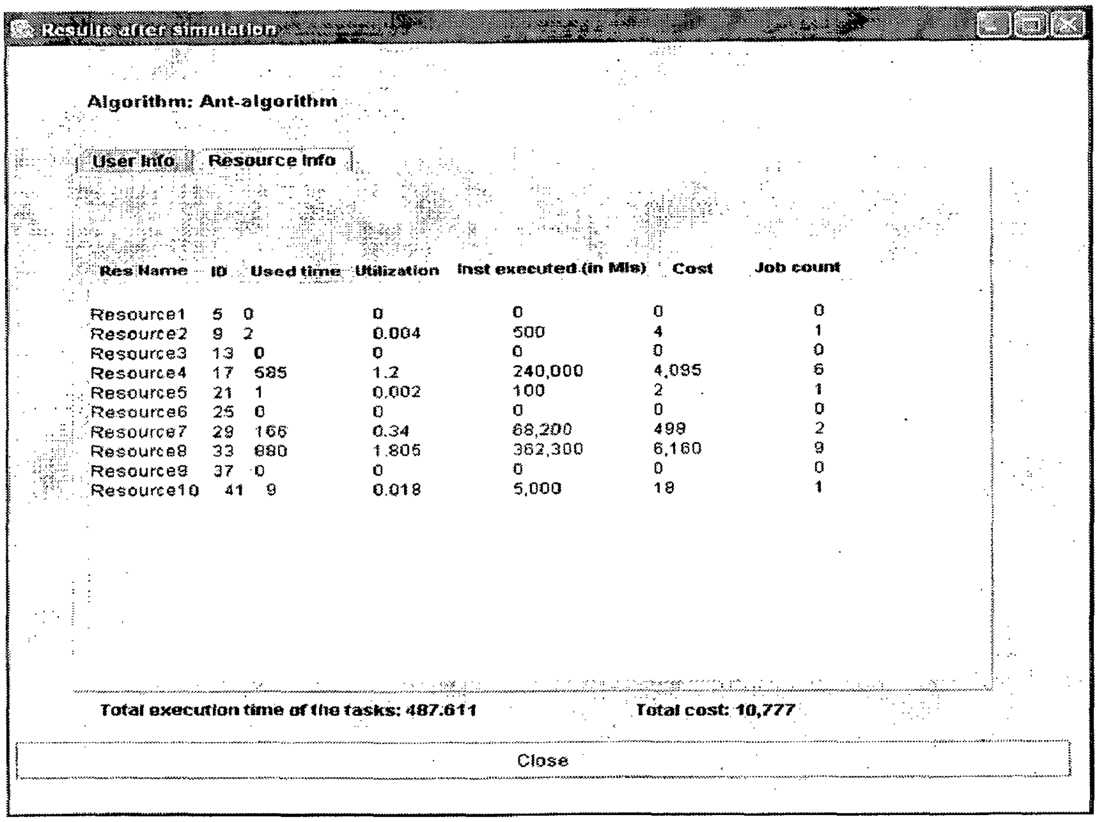
The results after the simulation of ant-algorithm scheduling:



Algorithm: Ant-algorithm

Gridlet ID	Status	Res ID	Arrival time	Compl time	Exec time	Cost
0	SUCCESS	33	46.3	229.23	182.93	1,280.49

Total execution time of the tasks: 487.611   Total cost: 10,777

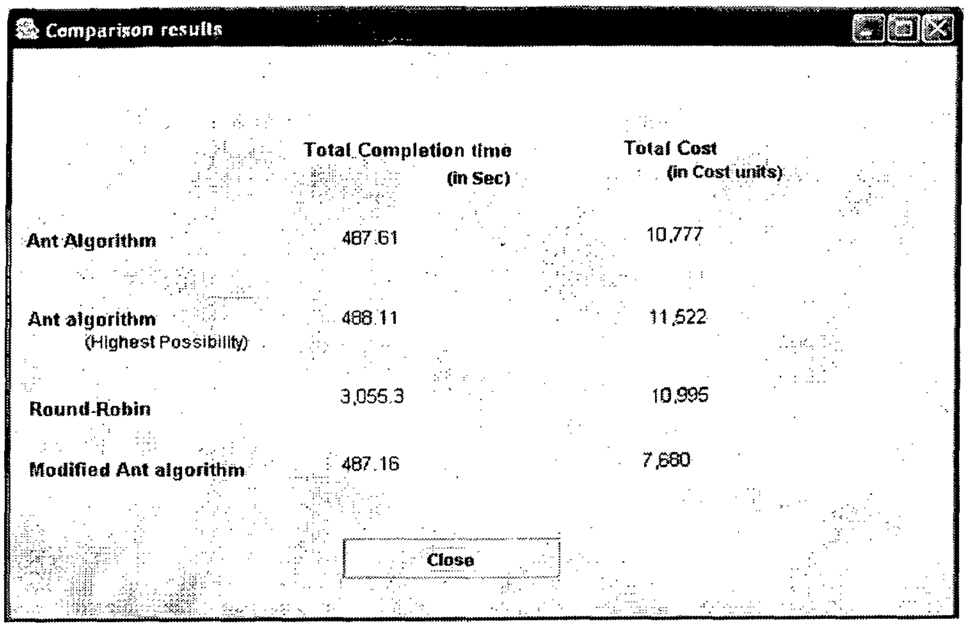


Algorithm: Ant-algorithm

Res Name	ID	Used time	Utilization	Inst executed (in MIs)	Cost	Job count
Resource1	5	0	0	0	0	0
Resource2	9	2	0.004	500	4	1
Resource3	13	0	0	0	0	0
Resource4	17	585	1.2	240,000	4,095	6
Resource5	21	1	0.002	100	2	1
Resource6	25	0	0	0	0	0
Resource7	29	166	0.34	68,200	499	2
Resource8	33	880	1.805	362,300	6,160	9
Resource9	37	0	0	0	0	0
Resource10	41	9	0.018	5,000	18	1

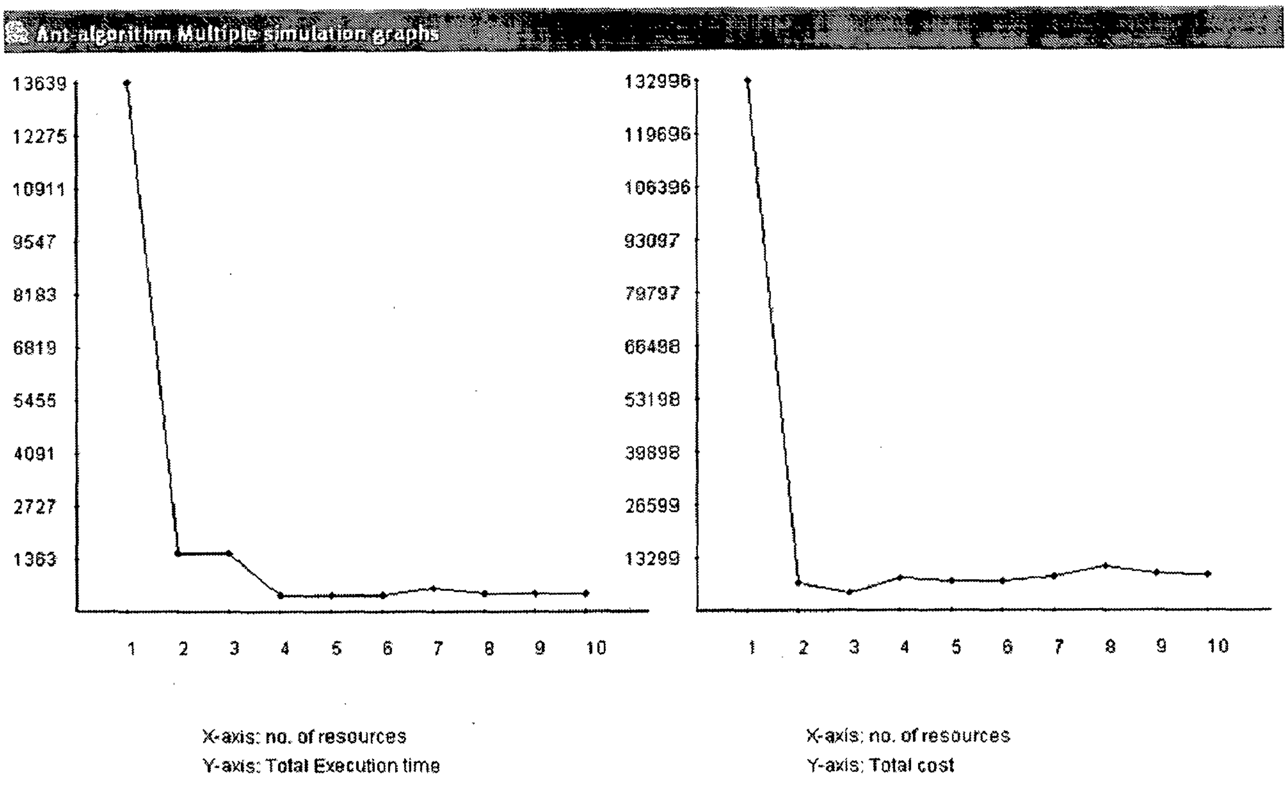
Total execution time of the tasks: 487.611   Total cost: 10,777

Comparison of different scheduling algorithms results:

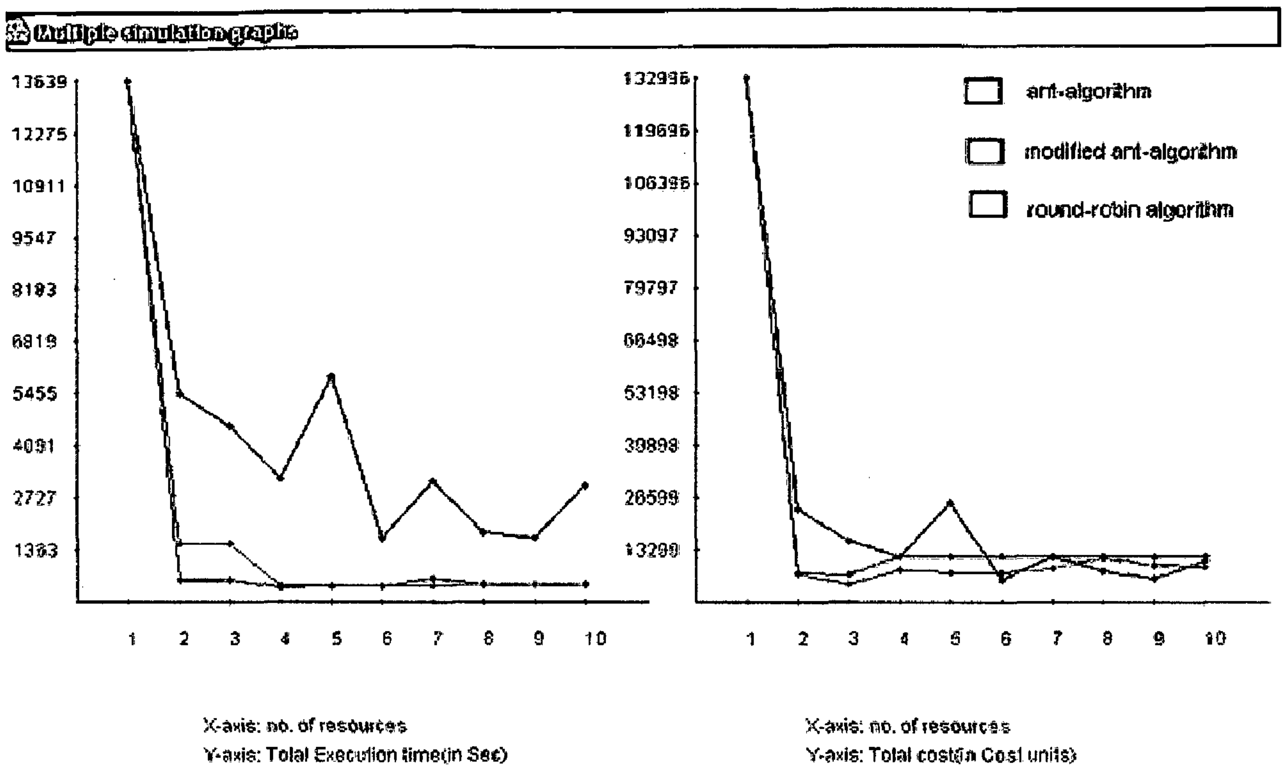


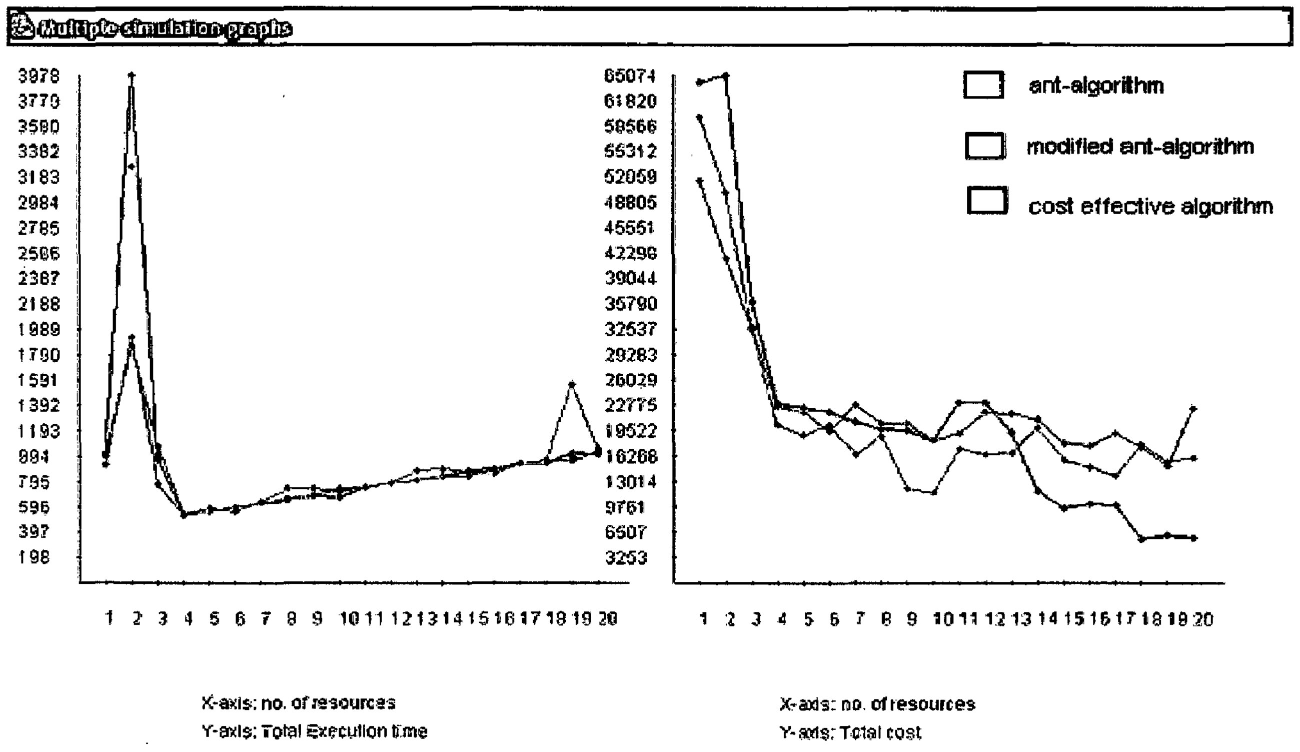
	Total Completion time (in Sec)	Total Cost (in Cost units)
Ant Algorithm	487.61	10,777
Ant algorithm (Highest Possibility)	488.11	11,522
Round-Robin	3,055.3	10,995
Modified Ant algorithm	487.16	7,680

Graph showing the results of ant-algorithm:



Graphs showing the comparison of different scheduling algorithm results





Appendix B

SOURCE CODE

Note that the GUI source code is not included here.

//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 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,nodeNo;
      private static int count_r,count_u;
      private static double threshold;
      private static ResultDisplay gui;

      Scheduling(String name, double baud_rate,int total_res,LinkedList resP,LinkedList
resPos,double at,int algo,int u,int nodeNo)
                  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;
      this.nodeNo=nodeNo;

      failList=new GridletList();
      resPher=resP;
      resPoss=resPos;
      this.ID_ = new Integer( getEntityId(name) );
      this.list_ = createGridlet( this.ID_.intValue());
}
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) {
   super.gridSimHold(1.0); // hold by 1 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++) {
   // Resource list contains list of resource IDs not grid resource
   // objects.
   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();
   // 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 alogorithm
  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;
   //this also has to be changed later
   setPossibility(j, pos);
  } //int j=0
  for (int t = 0; t < totalResource_; t++)
   flag[t] = 0;
  double rand = StrictMath.random();
  //System.out.println("random number:"+rand);
  if (algo == 0) {
   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 = getPossibility(p);
        if (max < val) {
         max = val;
         mpos = p;
        } //if max<val
       } //for
       flag[mpos] = 1;
       prob += getPossibility(mpos);
      } //while
      id=mpos;
     } //algo==0
     if (algo == 1) {
      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 = getPossibility(p);
       if (max < val) {
        max = val;
        mpos = p;
       }
      } //for
     } //algo=1
     id = mpos;
     if (algo == 2) {
      /*
      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
}*/
```

```
    max = 0;
                    int maxp=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 = getPossibility(p);
                       if (max < val) {
                         max = val;
                         maxp = p;
                       }
                     }//for

if((gridlet.getNumPE()<=resCharList[nodeNo].getNumFreePE())&&((max-
getPossibility(nodeNo))<=threshold))
                     {
                       id=nodeNo;
                     }
                     else
                     {
                     prob = 0;
                     while(true)
                     {
                       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 = getPossibility(p);
                           if (max < val) {
                             max = val;
                             mpos = p;
                           } //if max<val
                         } //for
                         flag[mpos] = 1;
                         prob += getPossibility(mpos);

                       } //while
```

```
              id = mpos;

              if (getPossibility(maxp) - getPossibility(id) <=gui.threshold)
              {
               id = mpos;
               break;
              }
              else {
               prob = 0;
               rand = StrictMath.random();
               for (int t = 0; t < totalResource_; t++)
                 flag[t] = 0;

              }
            }//while(true)

       for (int t = 0; t < totalResource_; t++)
          flag[t] = 0;
      int f=0;
      double min=0;
      int minp=0;
       while(true)
       {
        int p;
        for ( p = 0; p < totalResource_; p++) {
         if(flag[p]==1)
           continue;
        if(resCharList[p].getCostPerMI()<min||f==0)
        {
         f=1;
         min=resCharList[p].getCostPerMI();
         minp=p;
        }
       }//for

        if(Math.abs(getPossibility(id)-getPossibility(minp))<=(gui.threshold-
(getPossibility(maxp)
                       -getPossibility(id)))*1)
                 {
                  id=minp;
                  break;
                 }
       flag[minp]=1;
       minp=0;
       f=0;
```

```
        }//while
            }//else nodeNo
    } //algo ==2
    if (algo == 3) { //ant-algo
     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
     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]++;
        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 the resouce fails

        double expTime = 1;
        expTime = gridlet.getGridletLength() / resCharList[id].getMIPSRating();
        double rndTime;
        if (StrictMath.random() <= inf.resFr[id] / 100.0) {
          rndTime = StrictMath.random() * expTime;

          //cancelling the gridlet
          super.gridletCancel(gridlet, resourceID[id], rndTime);
          //System.out.println(this.name_ + ":Gridlet :" +
          //gridlet.getGridletID() + "  cancelled");
          failList.add(gridlet);
          newPher = getPheremone(mpos);
          newPher = 0.8 * newPher + 0.8 * K;
          //newPher = 0.8 * newPher + 1.6 * newPher;
          //updating the pheremone
          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;
          //this also has to be changed later
          setPossibility(j, pos);
          //System.out.println("possibility for resource "+j+":"+numFormat.format(pos));
          //System.out.println("pheremone for resource
"+j+":"+numFormat.format(getPheremone(j)));
        } //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 < count_r; p++)
            if ( (inf.resName[p]).equals(subResName)) {
              ResourceCharacteristics rc = resCharList[p];
              noInst[p]+=gridlet.getGridletLength();
              usedTime[p] += gridlet.getProcessingCost() / inf.resCost[p];
              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);
            newPher = 1.1 * newPher + 1.1 * K;
            //newPher = 1.1 * (newPher + 0.4 * newPher);
            //updating the pheremone
            setPheremone(mpos, newPher);
          }
        } //try
        catch (Exception e) {
          System.out.println(e);
        }
        // when a grid resource entity finished processing the Gridlet,
        // then set the resource id and its cost to do the job
        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;
        //this also has to be changed later
        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();
   }//body
    /**
     * Gets the list of Gridlets
     * @return a list of Gridlets
     */
    public GridletList getGridletList() {
         return this.receiveList_;
    }
    public GridletList getFailList()
    {
         return failList;
    }
    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);
        //int count = GridSim.rand.intSample(max);
        for (int i = 0; i < inf.nog[0]; 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);
        }

         count_u=inf.ucount;
         count_r=inf.rcount;
     res=new GridResource[count_r];
     ResUtil=new double[count_r];
    arrTimes=new long[count_u];
    gridletCount=new double[count_u];
    resGridletCount=new int[count_r];
    noInst=new int[count_r];
  resCharList=new ResourceCharacteristics[count_r];
  resName=new String[count_r];
  resId=new int[count_r];
  usedTime=new int[count_r];
  resCost=new double[count_r];
```

```
    }

    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=count_r;;
    try{
     int users=count_u;
     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 <count_u;u++)
      {
       if(inf.arrType==0)
       sh_time+=arrtime.sample();
       else
```

```
        //int count = GridSim.rand.intSample(max);
        for (int i = 0; i < inf.nog[0]; 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);
        }

        count_u=inf.ucount;
        count_r=inf.rcount;
     res=new GridResource[count_r];
     ResUtil=new double[count_r];
     arrTimes=new long[count_u];
     gridletCount=new double[count_u];
     resGridletCount=new int[count_r];
     noInst=new int[count_r];
  resCharList=new ResourceCharacteristics[count_r];
  resName=new String[count_r];
  resId=new int[count_r];
  usedTime=new int[count_r];
  resCost=new double[count_r];
```

```
        System.out.println("Starting simulation of Ant algorithm");
        try
        {
            int num_user =count_u;  // 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
            //System.out.println("Initializing 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<count_r;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];
          }
            // Third step: Creates grid users
            startSimulation();
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.out.println("Unwanted errors happen"+e);
        }
```

```
    }

    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=count_r;;
    try{
     int users=count_u;
     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 <count_u;u++)
      {
       if(inf.arrType==0)
       sh_time+=arrtime.sample();
       else
```

```
          sh_time=inf.arrTime[0];
         user[u]=new
Scheduling("User_"+String.valueOf(u),560.00,total_resource,resP,resPos,sh_time,algo,u,
count_u%count_r);
       }
       // Fourth step: Starts the simulation
       GridSim.startGridSimulation();
       SimTime=GridSim.clock();
       //writing results
       NumberFormat numFormat=NumberFormat.getInstance();
       int totCost=0;
       for(int i=0;i<count_r;i++)
       {
        totCost += resCost[i];
        ResUtil[i]=usedTime[i]/SimTime;
        //System.out.println(resNames[i]+"          "+resUtil[i]+"
"+(resUtil[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]));

       }

      gui.l_tcost.setText("Total cost: "+numFormat.format(totCost));
      gui.l_time.setText("Total execution time of the tasks:
"+numFormat.format(SimTime));
       System.out.println(count_r+" exec time:"+SimTime+"     cost:"+totCost);

       inf.totResCost[count_r-1][algo]=totCost;
       inf.totExecTime[count_r-1][algo]=SimTime;

       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];


       // Final step: 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();
    }
  }
     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 getPheremone(int i)
     {
          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
```