Genetic Algorithm Based Load and Time Prediction Technique for Dynamic Load Balancing In Grid Computing

Submitted By

ZAHIDA AKHTAR
141-CS/MS/2003

Supervised By

Prof. Dr. Afaq Hussain

Department of Computer Science
International Islamic University,
Islamabad

(2007)

Department of Computer Sciences

Faculty of Basic and Applied Sciences

International Islamic University Islamabad

Date: June 2nd, 2007

Final Approval

This is to certify that we have read the thesis entitled "Genetic Algorithm Based Load and Time Prediction Technique for Dynamic Load Balancing in Grid Computing" submitted by **Zahida Akhtar**, Reg # 141-CS/MS/03. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad for the degree of MS Computer Science.

Committee:

External Examiner
Dr. Abdus Settar
Ex. Director General
Pakistan Computer Bureau

[illegible]

Internal Examiner
Prof. Dr. Sakandar Hayat Khail
Head Department of Computer Sciences,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad

[illegible]

Supervisor
Prof. Dr. Afaq Hussain
Head Department of Electronics Engineering,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad

[illegible]

**In the name of *ALMIGHTY ALLAH*,
The most beneficent, the most
Gracious.**

A dissertation submitted to the

DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF BASIC AND APPLIED SCIENCES,

INTERNATIONAL ISLAMIC UNIVERSITY, ISLAMABAD

as

a partial fulfillment of the requirements for the award of the

Degree of MS Computer Science

Declaration

I hereby declare that this S/W, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that I have developed this S/W entirely on the basis of my personal effort made under the sincere guidance of my teachers and supervisor.

No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Zahida Akhtar
141-CS/MS/2003

Dedication

To My Parents and Grand Parents

Acknowledgements

In the name of Allah, the compassionate, the merciful, who made the human being super creatures, blessed us with knowledge and enabled me to accomplish this work.

First and foremost I would like to express my profound and sincerest gratitude to my supervisor Prof. Dr. Afaq Hussain for constant guidance and encouragement during the research work for this thesis. He was always available for consultation and invaluable advice. His patience and dedication has been unlimited.

I also like to thank Mr. Saeedullah for his assistance in this research project.

I was most indebted to all my family members, especially my parents, whose affection has been the source of encouragement for me, and whose prayers have always been the key to my success.

Zahida Akhtar
141-CS/MS/2003

Abstract

Present research aims to solve the grid load-balancing problem using Genetic Algorithms.

Grid computing has emerged as a leading research area. Load balancing in grid computation is an NP-complete problem. GA based scheduling algorithm named "Genetic Load and Time Prediction Technique for Dynamic Load Balancing in Grid Computing" is proposed for better resource optimization and task scheduling. The algorithm has been tested on a multi-node grid environment and the experimental results shows that this new technique can lead to significant performance gain in various applications.

Experimental results also show that GA outperforms other algorithms. The research also highlights some future areas for research.

Project in Brief

Project Title:	Genetic Load and Time Prediction Technique for Dynamic Load Balancing in Grid Computing
Objective:	To develop a GA based algorithm for grid resource optimization and task scheduling
Undertaken By:	Zahida Akhtar Reg. No. 141-CS/MS/2003
Supervised By:	Prof. Dr. Afaq Hussain Head Department of Electronics Engineering, Faculty of Basic and Applied Sciences, International Islamic University, Islamabad
Date Started:	September, 2004
Date Completed	November, 2006
Tools Used:	OpenMosix POVRAY 3.0 QT Designer Gimp Visio 2002 Rational Rose 2000 Adobe Acrobat 6.0 Professional MS Office 2005
Operating System Used:	Linux Red Hat 9
System Used:	Pentium II, Pentium III , Pentium IV

Table of Contents

Chapter #	Contents	Page #
1.	***Introduction***	***1***
1.1	Grid Computing	2
1.1.1	Grid Vs. Cluster Computing	3
1.1.2	Benefits of Grid Computing	4
1.2	Load Balancing	4
1.2.1	Different Load Balancing Problems	5
1.2.1.1	Task Costs	5
1.2.1.2	Task Dependencies	6
1.2.1.3	Locality	6
1.2.2	Load Balancing Solutions	6
1.2.2.1	Static Load Balancing	6
1.2.2.2	Dynamic Load Balancing	6
1.3	Genetic Algorithms	7
1.3.1	Simple Genetic Algorithm	7
1.3.2	Operators of Genetic Algorithms	9
1.3.3	Parameters of Genetic Algorithms	10
1.3.4	Encoding of Chromosomes	11
1.3.5	Selection	11
1.3.6	Elitism	11
1.4	Motivation	11
2.	***Literature Survey***	***13***
2.1	Load Balancing in Grid Computing	13
2.2	Genetic Algorithms for Load Balancing	15
2.3	Genetic Algorithms for Load Balancing in Grid Computing	17
2.4	Problem Statement	18
3.	***Proposed Methodology***	***20***
3.1	Proposed Solution	20
3.2	Functional Modules	21
3.2.1	Resource Collector	22
3.2.2	Resource Analyzer	22
3.2.3	GA Load Balancer/Re-Balancer	22
3.2.4	Task Collector	22
4.	***System Design***	***24***
4.1	Object Oriented Design Methods	24
4.1.1	Class Diagram	24
4.1.2	Sequence Diagram	26
4.1.3	System Architecture	37
5.	***Implementation***	***38***
5.1	Tools	38
5.1.1	Languages	38
5.1.2	Editors	38
5.1.3	Office Tools	38

Chapter # ***Contents*** ***Page #***

5.1.4 Grid Management Tools 39
5.2 Generating Code *39*
5.2.1 Resource Collector Methods *39*
5.2.2 Resource Analyzer Methods *39*
5.2.3 GA Based Analyzer and Load Balancer Methods *39*
5.2.4 Task Collector Methods *40*

6. Results and Discussion 43
6.1 Experimental Setup *43*
6.2 Experimental Results *43*
6.3 Sample Output *46*
6.4 Conclusion and Future Enhancements *48*
6.4.1 Conclusion *48*
6.4.2 Future Enhancements *49*
Appendix A – Bibliography and References A-1
Appendix B – Publications B-1
Appendix C – User Manual C-1

CHAPTER 1

INTRODUCTION

INTRODUCTION

Due to such a tremendous growth in technology, computational powers are reaching to their saturation and demand from software industry is increasing day by day. In order to address the sheer scale and complexity of the issues faced in pervasive computing; it is necessary that software engineers achieve new competences that are beyond those, which are required in past. In order to gain such computational powers, is required a system that divides a job into smaller tasks which can run simultaneously on multiple systems. Thus getting high computational powers and less execution time or in other words you can simply say the solution lies in distributed computing. Distributed computing is the next step in computer progress, where computers are not only networked but also smartly distribute their workload across each computer so that they stay busy and don't squander the resources. As shown in figure 1-1 in distributed computing environment task is distributed among multiple computers and thus leads to significant performance gain. This setup rivals even the fastest commercial supercomputers built by companies like IBM or Cray. When you combine the concept of distributed computing with the tens of millions of computers connected to the Internet, you've got the fastest computer on Earth.

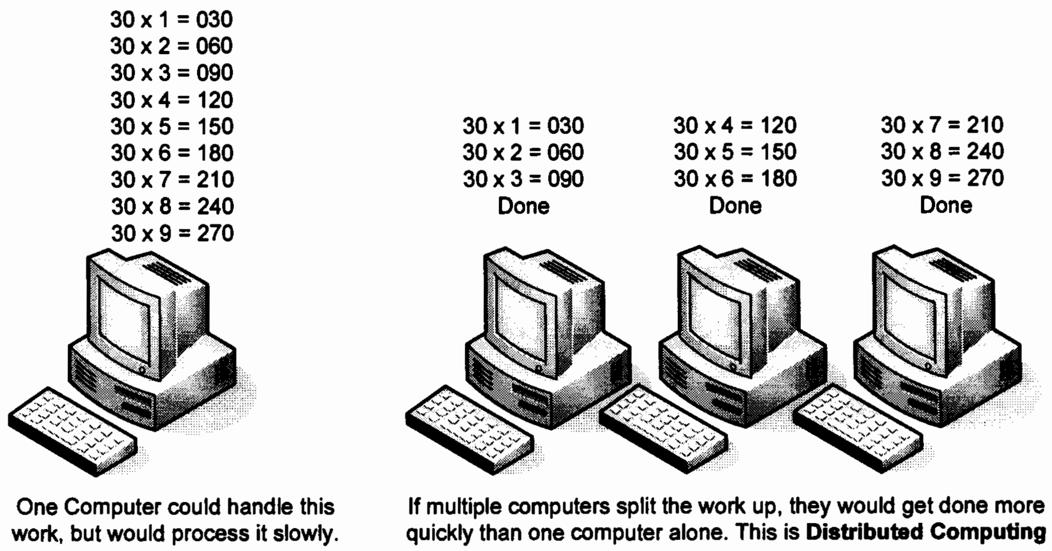


Fig. 1-1 Difference between Distributed Computing and Non-Distributed Computing

The demand for autonomous resource management for distributed computing resources has increased in recent years. Distributed computing requires an efficient and powerful communication mechanism between applications running on different hosts and networks.

There are many different types of distributed computing systems and many challenges to overcome in successfully designing one. The main goal of a distributed computing system is to connect users and resources in a transparent, open, and scalable way. Ideally this arrangement is drastically more fault tolerant and more powerful than many combinations of stand-alone computer systems. An example of a distributed system is the World Wide Web. As you are reading a web page, you are actually using the distributed system that comprises the site. As you are browsing the web, your web browser running on your own computer communicates with different web servers that provide web pages. Possibly, your browser uses a proxy server to access the web contents stored on web servers faster and more securely. To find these servers, it also uses the distributed domain name system. Your web browser communicates with all of these servers over the internet via a system of routers which are themselves part of a large distributed system.

At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of that network being printed onto a circuit board or made up of several loosely-coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system. Various hardware and software architectures exist that are usually used for distributed computing for example Client-Server architecture, 3-Tier architecture, N-Tier architecture, Cluster Computing etc. In Client-Server Computing, smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change. Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier. N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers. Cluster Computing refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in parts that are made individually by each one, and then put back together to make the final result.

Although clustering can provide significant improvements in total computing power, a cluster remains a dedicated facility, built at a single location. Financial, political and technical constraints place limits on how large such systems can become. For example, ASCI White cost $110 million and needed an expensive new building [1]. Few individual institutions or research groups can afford this level of investment. Cluster computing is not truly distributed computing. Cluster computing ties together similar types of resources in a data center with similar operating systems through special purpose connectors to deliver a specific application. Grid computing, in contrast, offers heterogeneity by supporting different software without the need for special connectors. Grid computing elevates these clusters to the next level by connecting multiple clusters over geographically dispersed areas for enhanced collaboration and resource sharing.

1.1 GRID COMPUTING

Surendra Reddy [2] defines Grid computing as an emerging technology that transforms a computer infrastructure into an integrated, pervasive virtual environment for dynamic collaboration and shared resources anywhere in the world providing users, especially in science, with unprecedented computing power, services and information. For scientists planning a research career, grid computing offers the prospect of much more computing power in a collaborative environment at a small fraction of the cost than before. Grid computing are a technology closely related to cluster computing. The key differences between grids and traditional clusters are that

grids connect collections of computers which do not fully trust each other, and hence operate more like a computing utility than like a single computer. In addition, grids typically support more heterogeneous collections than are commonly supported in clusters.
Grid computing is optimised for workloads, which consist of many independent jobs or packets of work, which do not have to share data between the jobs during the computation process. Grids serve to manage the allocation of jobs to computers, which will perform the work independently of the rest of the grid. All the nodes may share resources such as storage, but intermediate results of one job do not affect other jobs in progress on other nodes of the grid.

Grid computing is based on three simple concepts:

- *Virtualization*, severing the hard-coded association of resources to systems
- *Resource allocation and management*, dynamically allocating resources on demand, and managing them
- *Provisioning*, configuring resources whenever and wherever needed.

Grid computing adapts to and dynamically aligns with research or business needs; it takes advantage of widespread spare computing capacity and the ubiquity of the Internet, which can tie these resources together. But the advantages of grid computing don't stop with this dynamic alignment. Today, applications are independently constructed, custom configured, and sized for peak load. That peak load may only occur once a month, once a quarter, or even once a year, leaving resources under utilized most of the time. This under utilization is accepted as a trade off and forces people to choose between having enough scalability for peak loads and not buying too much so that they've invested a lot of money in idle capacity. This is the problem that grid computing solves.
Dynamic scaling is another basic component of grid computing based on the idea that because the grid computer infrastructure can be built of small, standard, interchangeable components, computer users can start small and then simply add more components as they scale. This means that grid computing can happen incrementally

1.1.1 GRID Vs. CLUSTER COMPUTING

According to Wikipedia Encyclopedia [3] many people confuse Grid computing with cluster-based computing, but cluster computing is not truly distributed computing. Cluster computing ties together similar types of resources in a data center with similar operating systems through special purpose connectors to deliver a specific application. Grid computing, in contrast, offers heterogeneity by supporting different software without the need for special connectors.

The key difference as mentioned in Wikipedia Encyclopedia [3] between Grids and traditional clusters are that Grids connect collections of computers which do not fully trust each other, and hence operate more like a computing utility than like a single computer. In addition, grids typically support more heterogeneous collections than are commonly supported in clusters.
Further more Grids are dynamic in nature, while clusters typically contain a static number of processors. Grids are distributed over local or wide area networks, and can

also dynamically adding and removing resources without interrupting the application services.

1.1.2 BENEFITS OF GRID COMPUTING

According to Matt Haynos [4] the impact of leveraging Grid computing will be dramatic. With the inherent ability of a Grid computing to provide nearly 100% uptime at an expected fraction of the costs of managing today's more static and fixed environment, both enterprises and institutional service providers can reap tremendous benefits. Following benefits of grid computing are listed on "*cronos e-business integrator*" web site [5]:

1 Grid computing enables organizations to aggregate resources within an entire IT infrastructure no matter where in the world they are located. It eliminates situations where one site is running on maximum capacity, while others have cycles to spare. Organizations can dramatically improve the quality and speed of the products and services they deliver, while reducing IT costs by enabling transparent collaboration and resource sharing.

2 Grid computing enables companies to access and share remote databases. This is especially beneficial to the life sciences and research communities, where enormous volumes of data are generated and analyzed during any given day.

3 Grid computing enables widely dispersed organizations to easily collaborate on projects by creating the ability to share everything from software applications and data, to engineering blueprints.

4 Grid computing can create a more robust and resilient IT infrastructure better able to respond to minor or major disasters.

5 A grid can harness the idle processing cycles that are available in desktop PCs located in various locations across multiple time zones. For example, PCs that would typically remain idle overnight at a company's Asian manufacturing plant could be utilized during the day by its European operations.

1.2 LOAD BALANCING

In distributed computing systems, scheduling and load balancing techniques are critical issues for achieving good performance improvement. Load balancing is defined as techniques which aim to spread tasks among the processors in a parallel processor to avoid some processors being idle while others have tasks queuing for execution. Load balancing may be performed either by heavily loaded processors (with many tasks in their queues) sending tasks to other processors, by idle processors requesting work from others, by some centralized task distribution mechanism, or some combination of these. Some systems allow tasks to be moved after they have started executing ("task migration") others do not.

Load balancing techniques assume practically no information/knowledge at compile time of the runtime parameters of an application, such as task execution times or communication delays. It mainly relies on the runtime distribution of processes among Processing Elements (PE) to achieve defined performance goals. Note that while load balancing refers to the redistribution as assigning of processes to different PE's, load sharing refers to the sharing of the systems processing power among the tasks.

1.2.1 DIFFERENT LOAD BALANCING PROBLEMS

Load balancing is very important for distributed applications. Making sure that each host is doing its fair share of work can be a real performance enhancer. Simplest key question is when the certain information about the load-balancing problem is known. Mainly load balancing problems differs in three areas, which are:

TASKS COSTS

Task Cost means, time the task will take to execute completely. The problem is, in the distributed and parallel task execution environment we don't know and we can't even assume that either all the task have equal cost. Definitely not all the tasks can have equal cost so the natural question is when the task cost is going know before starting the task or at the time the task is created or only at the time the task ends, all these are the task cost problem which are faced during the load balancing. This can easily be understood by the following task cost spectrum.

TASK DEPENDENCIES

Task Dependencies mean a task is dependent for its execution on some other task. These dependencies can be either data dependency (e.g. task u requires the results generated by task v) or some other kind (e.g. task v can't be executed until task u completes its execution) of dependency. The problem is, to find out that can all tasks be run in any order (without any dependencies) or sequence and if not then when these dependencies will be known, before task starting or when task created or only when the task ends. Task dependencies can be clearer by the following task dependency spectrum.

LOCALITY

Locality means, span of the task execution over the distributed and parallel machines e.g. some tasks have the requirement to run on the local processor only while some other tasks can run on distributed machines, the question is how far they can be spread over the machines in the distributed environment in order to gain the parallelism while keeping in view the communication and other costs and dependencies. The real problem for the load balancer is that it does not know that when this locality information about the task will be known to it. This will be understood by the following locality spectrum.

1.2.2 LOAD BALANCING SOLUTIONS

STATIC LOAD BALANCING

Processes in fixed set are statically assigned to processors, either at compile-time or at start-up (i.e. partitioning). Avoids the typical 5-20% overhead of load balancing, but is useless when the problem does not divide cleanly such as for problems involving irregularly or unpredictability such as: mesh generation, game playing (chess), and many optimization problems.

The simplest method is static load balancing. In this method, the problem is divided up and tasks assigned to processors only once. The partitioning may occur before the job starts, or as an early step in the application. The size and number of tasks can be varied depending on the processing power of a given machine. On a lightly loaded network, this scheme can be quite effective.

DYNAMIC LOAD BALANCING

When computational loads vary, a more sophisticated dynamic method of load balancing is required. The most popular method is called the Pool of Tasks paradigm. This is typically implemented as a master/slave program where the master manages a set of tasks. It sends slaves jobs to do as they become idle. This method is used in the sample program supplied with the distribution. This method is not suited for applications which require task to task communication, since tasks will start and stop at arbitrary times. In this case, a third method may be used. At some predetermined time, all the processes stop; the work loads are then reexamined and redistributed as needed. Variations of these methods are possible for specific applications.

KEY ISSUES IN THE DYNAMIC LOAD BALANCING

1. Load Measurement - *load index* is a simple measurement usually based on counting ready (and executing) processes on a processor. Other factors (communication, memory requirements, and multiple processors at an SMP node) are more difficult to address.
2. Information Exchange - the load at a node is meaningful only when compared to other nodes, often the neighbors. Information exchange may occur in anticipation of load balancing, may be periodic, or may be based on a significant change in a node's load index.
3. Initiation Rule - designed such that benefit exceeds cost. If balancing is initiated by an overloaded node, then designated as *sender-initiated.* If initiated by an under loaded node, then known as *receiver-initiated.* *Symmetrical* policies are also possible.
4. Load Balancing Operation - Defined by having rules for location, distribution, and selection.
    a. Location Rule determines which nodes participate.
    b. Distribution Rule determines the redistribution of load among the participants.
    c. Selection Rule determines the processes to move. A *non-preemptive* rule moves newly spawned processes that have not progressed on the

node of the parent processor. A *preemptive* rule can also migrate a process that has progressed.

For many practical problems it is not necessary to actually migrate processes, especially when a non-preemptive rule is used. Instead, just data that describes a *task* is migrated.

Even though this sub-area of parallel processing has been called load balancing, practical cases can often emphasize idleness-avoidance over fairness. Having all processors busy between load balancing operations is a reasonable goal.

Since load balancing schemes do not incorporate precedence's between tasks explicitly, some scenarios may lead to idle processors later in execution. Dynamic load balancing, however, is usually used in situations where the task graph is not available in advance. If the information exchange and location rules operate locally, then the technique is called a *nearest-neighbor* algorithm.

To move a process several hops, the technique acts as an *iterative* algorithm. Note that the iterations only manipulate load indices; only after these have converged does the redistribution occur.

A *direct* algorithm does not depend on iteration, but depends on having a communication system that supports the increased load. Direct algorithms behave as though the network was a complete graph. Broadcasting and wormhole routing are useful in supporting direct algorithms, but iterative techniques are still often preferable in large systems.

1.3 GENETIC ALGORITHMS

Genetic algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Evolutionary computing began by lifting ideas from biological evolutionary theory into computer science, and continues to look toward new biological research findings for inspiration. However, an over enthusiastic 'biology envy' can only be to the detriment of both disciplines by masking the broader potential for two-way intellectual traffic of shared insights and analogizing from one another [6].

"Genetic algorithms are based on a biological metaphor: They view learning as a competition among a population of evolving candidate problem solutions. A 'fitness' function evaluates each solution to decide whether it will contribute to the next generation of solutions. Then, through operations analogous to gene transfer in sexual reproduction, the algorithm creates a new population of candidate solutions."

1.3.1 SIMPLE GENETIC ALGORITHM

Solution to a problem solved by genetic algorithms is evolved. Algorithm is started with a **set of solutions** (represented by **chromosomes**) called **population**. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (**offspring**) are selected according to their **fitness** - the more suitable they are the more chances they have to succeed. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

SEARCH SPACE

If we are solving some problem, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (it means objects among those the desired solution is) is called **search space** (also state space) as shown in figure 1-2. Each point in the search space represents one feasible solution. Each feasible solution can be "marked" by its value or fitness for the problem. We are looking for our solution, which is one point (or more) among feasible solutions - that is one point in the search space. The looking for a solution is then equal to a looking for some extreme (minimum or maximum) in the search space. The search space can be whole known by the time of solving a problem, but usually we know only a few points from it and we are generating other points as the process of finding solution continues.

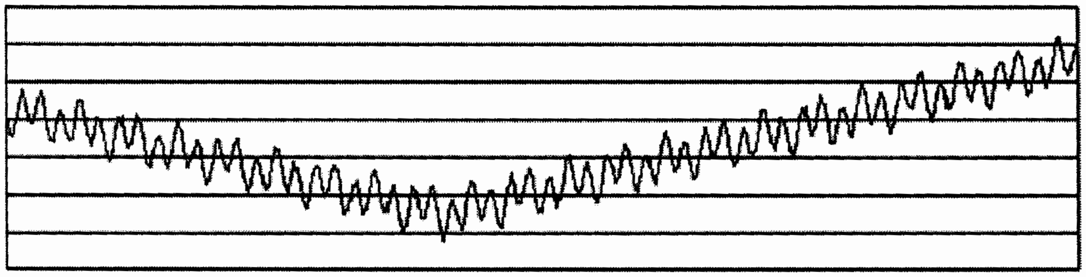


Fig. 1-2 Example of Search Space

The problem is that the search can be very complicated. One does not know where to look for the solution and where to start. There are many methods, how to find some **suitable solution** (ie. not necessarily the **best solution**), for example HILL CLIMBING, TABU SEARCH, SIMULATED ANNEALING and GENETIC ALGORITHM. The solution found by these methods is often considered as a good solution, because it is not often possible to prove what is the real optimum?

OUTLINE OF BASIC GENETIC ALGORITHM

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)

2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population

3. **[New population]** Create a new population by repeating following steps until the new population is complete

    - **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

- **[Crossover]** Cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
- **[Mutation]** Mutate new offspring at each locus (position in chromosome).
- **[Accepting]** Place new offspring in a new population

4. **[Replace]** Use new generated population for a further run of algorithm
5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Go to step **2**

SOME COMMENTS

The outline of Basic GA is very *general.* There are many things that can be implemented differently in various problems. First question is *how to create chromosomes*, what type of *encodings* to choose. With this is connected *crossover* and *mutation*, the two basic **Operators of GA**. Next question is *how to select parents* for crossover. This can be done in many ways, but the main idea is to select the better parents (in hope that the better parents will produce better offspring). Also you may think, that making new population only by new offspring can cause lost of the best chromosome from the last population. This is true; hence the so called Elitism is often used. This means, that at least one best solution is copied without changes to a new population, so the best solution found can survive to end of run.

1.3.2 OPERATORS OF GA

The performance of a GA is influenced mainly by the following these two operators.

- Crossover
- Mutation

CROSSOVER

After we have decided what encoding we will use, we can make a step to crossover. **Crossover selects genes from parent chromosomes and creates a new offspring.** The simplest way how to do this is to choose randomly some **crossover point** and everything before this point is copied from the first parent and then everything after a crossover point is copied from the second parent.
Crossover can then look like this (| is the crossover point):

Chromosome 1	11011 | 00100110110
Chromosome 2	11011 | 11000011110

Offspring 1	11011 | 11000011110
Offspring 2	11011 | 00100110110

There are other ways to make crossover, for example we can choose more crossover points. Crossover can be rather complicated and very depends on encoding of the encoding of chromosome. Specific crossover made for a specific problem can improve performance of the genetic algorithm.

MUTATION

After a crossover is performed, mutation takes place. **Mutation changes randomly the new offspring**. This is to prevent falling all solutions in population into a local optimum of solved problem. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1.
Mutation can then be following:

Original offspring 1	1101111000011110
Original offspring 2	1101100100110110
Mutated offspring 1	1100111000011110
Mutated offspring 2	1101101100110100

The mutation depends on the encoding as well as the crossover. For example when we are encoding permutations, mutation could be exchanging two genes.

1.3.3 PARAMETERS OF GENETIC ALGORITHMS

CROSSOVER PROBABILITY

Crossover Probability says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is **100%**, then all offspring is made by crossover. If it is **0%**, whole new generation is made from exact copies of chromosomes from old. Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation.

MUTATION PROBABILITY

Mutation Probability says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is **100%**, whole chromosome is changed, if it is **0%**, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to **random search**.

OTHER PARAMETERS

There are also some other parameters of GA. One also important parameter is population size. **Population Size** says how many chromosomes are in population (in one generation). If there are too few chromosomes, GAs has a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster

1.3.4 ENCODING OF CHROMOSOMES

Encoding of chromosomes is one of the problems, when you are starting to solve problem with GA. Encoding very depends on the problem. There are several types of encoding which are as follows

- Binary Encoding
- Permutation Encoding
- Value Encoding
- Tree Encoding

1.3.5 SELECTION

Chromosomes are selected from the population for the crossover. The problem is how to **select** these chromosomes. According to Darwin's evolution theory the **best ones** should survive and create new offspring. There are many methods how to select the best chromosomes, for example roulette wheel selection, rank selection, steady state selection and some others.

1.3.6 ELITISM

When creating new population by crossover and mutation, we have a big chance, that we will loose the best chromosome. Elitism is name of method, which first copies the best chromosome (or a few best chromosomes) to new population. The rest is done in classical way. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution.

1.4 MOTIVATION

Thanks to advances in wide-area network technologies and the low cost of computing resources, Grid computing came into being and is currently an active research area. One motivation of Grid computing is to aggregate the power of widely distributed resources, and provide non-trivial services to users. To achieve this goal, an efficient

Grid scheduling system is an essential part of the Grid which not only schedules the jobs on grid but also keeps the balance between all resources in terms of their utilization.
Most of the research on load-balancing focused on static scenarios that, in most of the cases, employ heuristic methods. However, genetic algorithms have gained immense popularity over the last few years as a robust and easily adaptable search technique. The work proposed here investigates how a genetic algorithm can be employed to solve the dynamic load-balancing problem. A dynamic load-balancing algorithm is developed whereby optimal or near-optimal task allocations can "evolve" during the operation of the parallel computing system.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

Since our research is focused on three different paradigms namely Genetic Algorithms, Load Balancing and Grid Computing therefore the following paragraphs gradually builds the Literature survey in the same mode.

2.1 LOAD BALANCING IN GRID COMPUTING

Grid computing has become an increasingly popular solution to optimize resource allocation in highly charged IT environments. In research study of M. Wieczorek et al. [6] three different algorithms (namely HEFT, Genetic and simple Myopic algorithm) are compared in terms of incremental versus full-graph scheduling for balanced versus unbalanced workflows. The results provided by authors shows that HEFT algorithm works better than the other two algorithms. But the future work of M. Wieczorek et al. shows that it does not consider the impact of typical network scenarios, comparison is done on homogeneous cluster nodes and last but not least no communication is considered.

D.P. Spooner et al. [7] primarily focus on implementing full operability with Globus to enable performance-based 'global' scheduling in addition to 'local' scheduling and for that purpose the authors primarily concentrated on the iterative heuristic algorithm and performance prediction techniques that have been developed for the local schedulers. But the main flaw in [7] is that its work is blocked with Globus toolkit only and will not be helpful if other grid projects like Nimrod [8], NetSolve [9], AppleS [10], Ninf [11], Condor [12], LSF [13] and Grid Resource broker (GRB) [14] are in use. Further more the system is tested on homogeneous cluster nodes with non real world data and embarrassingly parallel jobs and scheduling cost is not considered for evaluating system performance.

J. Cao et al. [15] addresses grid load balancing issues using a combination of intelligent agents and multi-agent approaches. In this research study, for local grid load balancing the iterative heuristic algorithm is proven to be more efficient than the first-come-first-served algorithm and for global grid load balancing a peer-to-peer service advertisement and discovery technique is proven to be effective. The experimental results proves that the use of a distributed agent strategy can reduce the network overhead significantly and make the system scale well rather than using a centralized control, as well as achieving a reasonable good resource utilization and meeting application execution deadlines. This research work is the initial attempt towards a distributed framework for building such and intelligent grid environment. The extensions of the agent framework with new features e.g automatic QoS negotiation, self-organizing coordination, semantic integration, knowledge-based reasoning and ontology-based service brokering has been identified but not yet implement in the research.

R. Buyya et al. [16] addressed the hybridization of the three popular nature's heuristics namely Genetic Algorithms (GA), Simulated Annealing (SA) and Tabu Search (TS) for dynamic job scheduling on large-scale distributed systems. The comparison of these 3 hybrid algorithms that is GA-SA and GA-TS with the pure GA was done and gain in efficiency of GA (in case of GA-TS) and better convergence (in case of GA-SA) was claimed but no experimental results are shown to prove the claim of the paper.

A novel mapping heuristic based on the cross-entropy (CE) method, for mapping a set of interacting tasks of a parallel application onto a heterogeneous computing platform, was proposed by S. Sanyal and S. K. Das [17]. The experimental results on 50 nodes resource graph outperforms the GA over a factor of about 38. The only shortcoming which is intended to remove as a future work in this research study is the slowness of cross-entropy method which is inherently slow in its execution and the fact that this slowness of CE in generating the appropriate mapping can decrease the performance gain for the large set of tasks.

R. A. Moreno [18] had done the detailed analysis of main tasks that the grid resource broker has to tackle (like resource discovery and selection, job scheduling, job monitoring and migration etc.) in his research work, and different approaches to perform these tasks is carefully examined. Finally, the problem of dynamic resource brokering in an economic grid environment is discussed and new re-scheduling policies for job migration under cost constraints are proposed with the promise that economy-based migration policies can significantly reduce the final price that the user pays for the resources used, without compromising performance.

According to S. Wagner et al [19] in contrast to other existing grid computing or parallel optimization projects, Heuristic-Lab Grid offers the possibility of rapid and easy use of existing optimization algorithms and problems in a parallel way without the need of complex installation and maintenance. In this research study authors present a new environment for parallel heuristic optimization based upon the already proposed Heuristic-Lab. The Integration of some more parallel heuristic optimization algorithms, new project dedicated to a common problem concerning heuristic optimization and the adjustment of parameter values are planned for future work.

2.2 GENETIC ALGORITHMS FOR LOAD BALANCING

In research paper of N. Navet et al. [20] a new genetic algorithms based technique is proposed to best set the scheduling of tasks according to a chosen criterion such as the minimizing of end-of-execution jitter or the maximizing of the freshness or the consistency of a set of input data. The proposed GA makes use of the schedule-ability analysis, as well as simulation because of the stochastic measurements required by the fitness criteria. The only little shortcoming seems to present in this research work is that at present only feasible schedules are allowed in the population of solutions and the possibility to allow non-feasible schedules in the population of solutions is still under work. Non-feasible schedules are those which require a measure of the feasibility finer than just yes or no. Other possible extensions as identified by this study include taking into account of shared resources and the extension to the distributed case. Moreover the possibility that the GA gives some feedback when the algorithm fails in order to help the designer to identify the bottlenecks of the system is also recognized as future work but not yet done in the present research.

A Formal model which allows multiple schedule optimizations and a new efficient heuristic approach based on genetic algorithms and list scheduling is presented by M. Grajcar [21]. According to the concluding remark of the paper, it does misses some important features e.g. selecting from multiple genetic operators and advanced parents selection mechanism but the proposed algorithm still performs well both in terms of running speed and result quality. Removal of some programming inefficiencies are planed as the future work.

A. Y. Zomaya et al.[22] investigates how a genetic algorithm can be employed to solve the dynamic load balancing problem. The dynamic load-balancing algorithm is developed whereby optimal or near-optimal task allocations can "evolve" during the operation of the parallel computing system. The algorithm considers other load balancing issues such as threshold policies, information exchange criteria and inter-processor communication. According to the paper the GA-based scheme works better when the number of tasks is large and where the consistent performance is observed while the other heuristics fail. The goals of minimum total completion time and maximum processor utilization are claimed to be achieved in this research work.

The idea of building composite sorting algorithms from primitives to adapt to the target platforms and the input data is proposed by X. Li et al. [23]. Genetic Algorithms are used to search for the sorting routines and the results shows that the best sorting routines are obtained when GAs are applied for the generation of classifier system. The generated sorting algorithm is on the average 36% faster than the best pure sorting algorithm when experimented on seven different platforms including IBM ESSL, the INTEL MKL and the STL of C++. According to authors this research study on the average, generated routines are 26% and 62% faster than the IBM ESSL in an IBM Power 3 and IBM Power 4, respectively.

A scheduling routine based upon a genetic algorithm is developed by W. A. Greene [24] which is claimed to be very effective and has relatively low cost. Two important aspects of this research paper are: loads on the processors are well balanced and scheduling per se remains cheap in comparison to the actual productive work of the processors. This scheduling routine is tested using 48 different scheduling experiments, incorporating 8 different task distributions and in all experiments the scheduler produces well-balanced schedules.

Dynamic Distributed Genetic Algorithm is proposed by W. Yi et al. [25]. According to the paper dynamic distributed GA with directed migration has great potential to overcome premature convergence. By employing directed migration, the research paper claims that genetic diversity could be maintained through the global living space. Since the size of each sub-population changes responding to the change of its performance, better species are encouraged and poorer ones punished. There is no global comparison so the overhead brought by the central monitor is negligible.

2.3 GENETIC ALGORITHMS FOR LOAD BALANCING IN GRID COMPUTING

The contribution of S. Song et al.'s [26] research study is two-fold: first the Min-Min and Sufferage heuristics are enhanced under three risk modes driven by security concerns and secondly a new Space-Time Genetic Algorithm for trusted job scheduling is proposed. The results provided in this research work shows that there is a need of more research study in order to over come the shortcoming of security driven Min-Min and Sufferage heuristics who are unstable when apply to different types of workloads plus the little flaw of Space-Time Genetic Algorithm that there is a high number of risk-taking jobs experience.

A novel GA-based approach is proposed by S. Kim et al. [27] to address the problem of scheduling a divisible Data Grid application while considering communication and computation at the same time in wide area data intensive environment. The results from the experiments on GA-related parameters suggest that the initialization of population with chromosomes of good quality is critical to GA-based approach in terms of the quality of solution and the convergence rate. The problem of multiple jobs competing for shared resources is identified as future work but not yet implemented in the present research.

Five heuristics are designed, developed, and simulated using the HC environment, in research work of S. Shivle et al. [28]. Application tasks composed of communicating subtasks with data dependencies and multiple versions were mapped using the heuristics described in this research. According to authors, the GA, on average produced the best mappings. The results this research can be used in the development of ad hoc grids as claimed by the paper.

In the research study of Y. Tanimura et al. [29], the EVOLVE/G system, which is a Grid tool for developer of evolutionary computation, is proposed. This system consists of Agent and multiple Workers. Since the data can be exchanged between Agent and Workers freely, any logical models of EC can be integrated. The system also has the mechanism of clustering nodes on the Grid, which are placed in the tree topology. Using the EVOLVE/G, the Grid model of the PSA/GAc (Parallel Simulated Annealing using the Genetic Crossover) is implemented which is one of the applications of EC. Two types of logical models of the PSA/GAc are prepared; the general model and hybrid model. Through the experiment, it is shown that the hybrid model has a good performance. In the hybrid model, the fine grained communication is performed in a PC cluster and the coarse grained communication is occurred between PC clusters. As a result, it presents that the EVOLVE/G system is useful to develop systems of evolutionary computation. No future extensions are identified.

T. Jing et al. [30] in their research work describes a parallel hybrid-GA (PHGA) for combinatorial optimization using an island model running in a networked computing environment. Basically two-island PHGA implemented in a distributed computing environment has been studied in the paper, and a new algorithm that embeds island PHGA model with NetSolve is presented. Several QAP (quadratic assignment problem) benchmarks are run and compared with the serial GA in order to verify the performance of PHGA and the results show that the PHGA offers both improved solution quality and speedup due to parallel processing. Several issues for future research are opens like extensive study on scalability of parallel GA in a distributed computing framework is required plus there is room to explore the applicability of parallelizing the local search of a serial GA. Along with these, other heuristics for local search can be explored to enhance the performance of the parallel GA.

In research work of J. Cao et al. [31] a GA-based scheduler has been developed for fine-grained load balancing at the local level, which is then coupled with an agent-based mechanism that is applied to load balance at a higher level. The experimental results demonstrate that the agent-based mechanism coupled with performance-driven task scheduling is suitable for grid load balancing. Future enhancement to the system will include the integration with other grid toolkits (e.g. Globus MDS and NWS).

2.4 PROBLEM STATEMENT

Since the scheduling problem is of crucial importance to the effective utilization of large
scale parallel computers and distributed computer networks (grid is covering of course both categories) , many different forms of scheduling have been studied. In a broad sense, the general scheduling problem can be divided into two categories—*job scheduling* and *scheduling and mapping*. In the former category, independent jobs are to be scheduled among the processors of a distributed computing system to optimize overall system performance. In contrast, the scheduling and mapping problem requires the allocation of multiple interacting tasks of a single parallel program in order to minimize the completion time on the parallel computer system. While job scheduling requires dynamic run-time scheduling that is not *a priori* decidable, the scheduling and mapping problem can be addressed in both static as well as dynamic contexts. When the characteristics of the parallel program, including its task execution times, task dependencies, task communications and synchronization, are known *a priori*, scheduling can be accomplished offline during compile-time. On the contrary, dynamic scheduling is required when *a priori* information is not available and scheduling is done on the fly according to the state of the system.

While different approaches have used GAs for solving load balancing problems yet the issues that remain to be addressed can be broadly categorized as following.

a) The execution time for load balancing has not been considered or has not been quantitatively described.
b) A few dynamic load balancing algorithms that we have come across in our literature review still not completely and comprehensively define for real world scenarios.
c) Although the good schedule were claimed to be achieved but performance is still compromised in term schedule cost.
d) To the best of our knowledge, no algorithm has been designed to prevent resubmission in case of load failure. The algorithms that incorporate fault tolerance use a simple strategy for restarting the task which in some cases requires extensive overheads.
e) Most of the algorithms are restricted to static load balancing and as such require a prior knowledge of various parameters. While this approach may

work in problems of equivalent nature but cannot be broadly applied to different applications.

Efficient execution in a distributed system can require, in the general case, mechanisms for the *discovery* of available resources, the *selection* of an application-appropriate subset of those resources, and the *mapping* of data or tasks onto selected resources.

CHAPTER 3

PROPOSED METHODOLOGY

3. PROPOSED METHODOLOGY

Scheduling, performance prediction and resource management are important but challenging tasks for grid computing efforts. In particular, there has been existing and ongoing work on network batch queuing and wide area scheduling. Ian Foster and others have done researches on the nature of the grid scheduling environment with its multiple possible performance measures and constraints, its large number of jobs and job sources and its large number of processing, transmission and storage resources.
Grid development involves the efficient management of heterogeneous, geographically distributed, and dynamically available resources. In this environment, the resource broker (or scheduler) becomes one of the most critical components of the grid middleware, since it has the responsibility of selecting resources and scheduling jobs in such a way that the user/application requirements are met, in terms of overall execution time (performance) and cost of the resources utilized. A grid resource broker is required to perform the four tasks which are *Resource discovery, Resource selection, Job scheduling, Job monitoring and migration.*

3.1 PROPOSED SOLUTION

In this research work we will basically try to increase the efficiency of grid scheduler. Genetic Algorithms based scheduling algorithm named "Dynamic Online Scheduling" is proposed for better resource optimization and task scheduling. The scheduling process in this algorithm is addressed in two layers namely pre scheduling and post scheduling. The newly coming problems from outside grid boundary are scheduled in the first layer that is pre scheduling. In post scheduling the load balancing of the already submitted tasks is done, that is if certain resource is found overloaded with work while some other resources are free then some of the jobs of the overloaded machine is automatically shifted to the free machines while keeping in mind the robustness, reliability and efficiency of the job as well as the time cost, communication cost and resource cost will also be considered.

The proposed solution is a grid 3 tier infrastructure to enable applications to leverage the idle computing power from commodity computers as shown in figure. 3-1. First tier is the logical representation for user and task (submitted by user) management. Second tier consists of resource management and GA Load Balancer framework.

Key features of this new middleware are; support for a wide range of parallel applications, use of advance GA based techniques on architectural design and development, and node usage pattern collection, analysis and prediction. It has also a great potential for lowering the level of waste of computational resources in today's computing infrastructure. Third tier consists of task resource mapping, migration and execution.

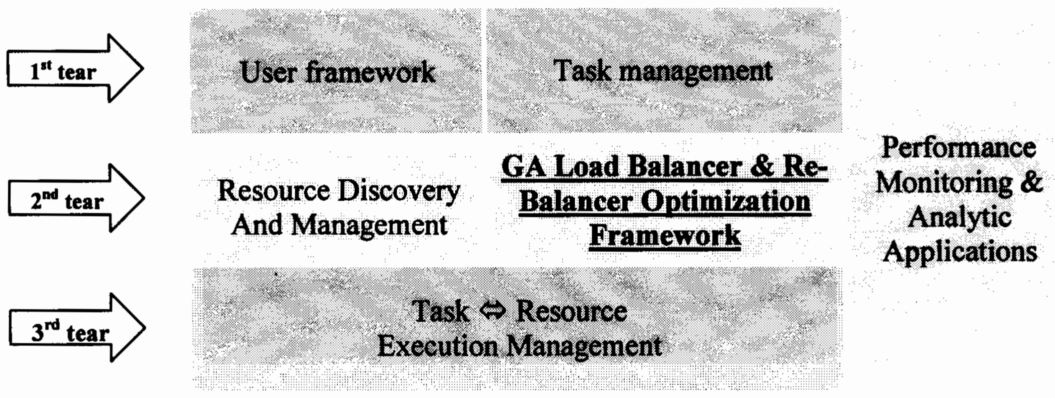

1st tear
2nd tear
3rd tear
User framework
Task management
Resource Discovery And Management
GA Load Balancer & Re-Balancer Optimization Framework
Performance Monitoring & Analytic Applications
Task ⇔ Resource Execution Management

Fig. 3-1 Grid Infrastructure

As show in the figure 3-1 Performance Monitoring and Analytic Applications are present in all three above mentioned tears. This is because it is playing its role in all three layers.

In this architecture tasks that are submitted from outside boundary, will be buffered inside the GA Load Balancer. Before scheduling the tasks, the balancer dynamically gets a list of available resources from the global directory entity. In order to get best results for parallel assignment of tasks we will use some GA based technique. The search space for genetic algorithm increases exponentially with the arrival of new tasks. The model of the system is learned in terms of best individual calculated by genetic algorithm. Time series analysis will be performed in a machine learning environment, and processing is done in two phases: the learning phase and the operational phase. In learning phase previous results will be gathered. In processing phase new threshold values will be assigned to resources.

3.2 FUNCTIONAL MODULES

Functional modules of the system are described below:

3.2.1 RESOURCE COLLECTOR

Resource collector manages all the resources and generates the log of each resource containing its utilization information at any particular time. Following pseudo code will explain its working.

1. Starts the resource collector.
2. Collect the utilization information from all resources. The utilization information which is monitored in this project consists of resource's status (online, offline), resource's load, total memory, current percentage utilization of memory and number of processors.
3. Display all the above mentioned information on GUI screen.
4. Continuously update this information after specified interval of time.

As mentioned above openmosix view is used as resource management tool. Openmosix provides us one solution for all above mention four steps. Openmosix View is not being used as a cluster management tool rather as a resource monitoring tool which could gather information about any kind of heterogeneous resources making a grid environment.

3.2.2 RESOURCE ANALYZER

Resource analyzer tool developed in this project analyzes all the data of each resource which is collected by resource collector in the log files. Following pseudo code will explain its working.

1. When resource analyzer starts it collects the current situation of all the resources.
2. Check resource's status for every resource then collect the timed history, which is generated by resource collector, of all the online resources.
3. Based on the time history set the threshold for each online resource. This threshold is updated after each one hour. This threshold tells us how many tasks could be assigned to that resource at any give time.

3.2.3 GA LOAD BALANCER/RE-BALANCER

This is the second important module of our grid. Our genetic algorithm is working here. The features of GA Load Balancer are as follows:

- **a.** Read the parsed task log file with respect to the parameters required by GA. In short it collects the offline information of resources.
- **b.** Extract the History of each task that matches any task in the submitted task queue according to the resource it is assigned to.
- **c.** Apply Genetic algorithm and get the results which can be in the form of optimized task schedule assigned to resources.
- **d.** Perform task mapping, migration and execution according to the schedule provide in above step and provide output results.
- **e.** In case of node failure, resume backup to get the tasks information and redistribute the tasks.
- **f.** Maintain the task and resource history for future use.

3.2.4 TASK COLLECTOR

Task collector tool developed in this project is the main GUI which interacts with user. It works according to the following pseudo code.

1. User submits the task by providing task input file, output file, type and size.
2. It calculates the task submission time and put it in the task queue.
3. On receiving run command it schedules it to the appropriate resource and starts its execution.
4. During the whole task execution period it monitors the resource to which this task is assigned.

5. After task is finished it calculates its total execution time and enters the task data i.e task type, size and submission time etc as well as the resource utilization during the execution of task into the log file.

Task scheduling tools works in the back end of task collector tool. Whenever the task comes it's the scheduler's job to assign it to the appropriate node for execution according to some criteria. In this project this criteria is genetic algorithm based. GA based scheduler and analyzer is explained in the following section.

CHAPTER 4

SYSTEM DESIGN

4. SYSTEM DESIGN

System design is the specification or construction of a technical, computer-based solution for the business requirements identified in the system analysis. It is the evaluation of alternative solutions and the specification of a detailed computer-based solution. The design phase is the first step towards moving from problem domain to the solution domain. System design develops the architectural detail required to build a system or product. In this phase we have designed software that will be used to verify the efficiency of proposed enhanced schema technique.

Design is actually a multi-step process that focuses on four distinct attributes of a program: data structure, software architecture, interface representation and procedural detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins.

Design of system is always considered as the main goal to achieve because it is the base of any system. It is necessary to have a solid and strong base to build the whole architecture, so, the base should be well built to hold the system. No matter how well the system is coded, if it does not have satisfying infrastructure, it will not be able to meet the requirements of the customer. So, main emphasis should be given to the design of any system.

4.1 OBJECT ORIENTED DESIGN METHOD

Object-Oriented design translates the Object Oriented Analysis (OOA) model of the real world into an implementation-specific model that can be realized in software. Object-oriented design transforms the analysis model, created using object-oriented analysis method, into a design model that serves as a blueprint for software construction. For the development of the system under consideration the same technique is used.

Object-oriented design (OOD) is concerned with developing an object-oriented model of a software system to implement the identified requirements.

Object Oriented Design builds on the products developed during Object-Oriented Analysis (OOA) by refining candidate objects into classes, defining message protocols for all objects, defining data structures and procedures, and mapping these into an object-oriented programming language (OOPL).

4.1.1 CLASS DIAGRAM

Class diagrams are the backbone of almost every object-oriented method including UML. They describe the static structure of a system. It can also be said that class diagrams identify the class structure of a system, including the properties and methods of each class. Also depicted are the various relationships that can exist between classes, such as an inheritance relationship. The Class diagram is one of the most widely used diagrams from the UML specification.

Another purpose of class diagrams is to specify the class relationships and the attributes and behaviors associated with each class. Class diagrams are remarkable at illustrating inheritance and composite relationships. A class diagram consists of one major component and that is the various classes, along with these are the various relationships shown between the classes such as aggregation, association, composition, dependency, and generalization. Refer to figure 4.1 which represents the class diagram of the software that will show the processing of the queries and their time differences.

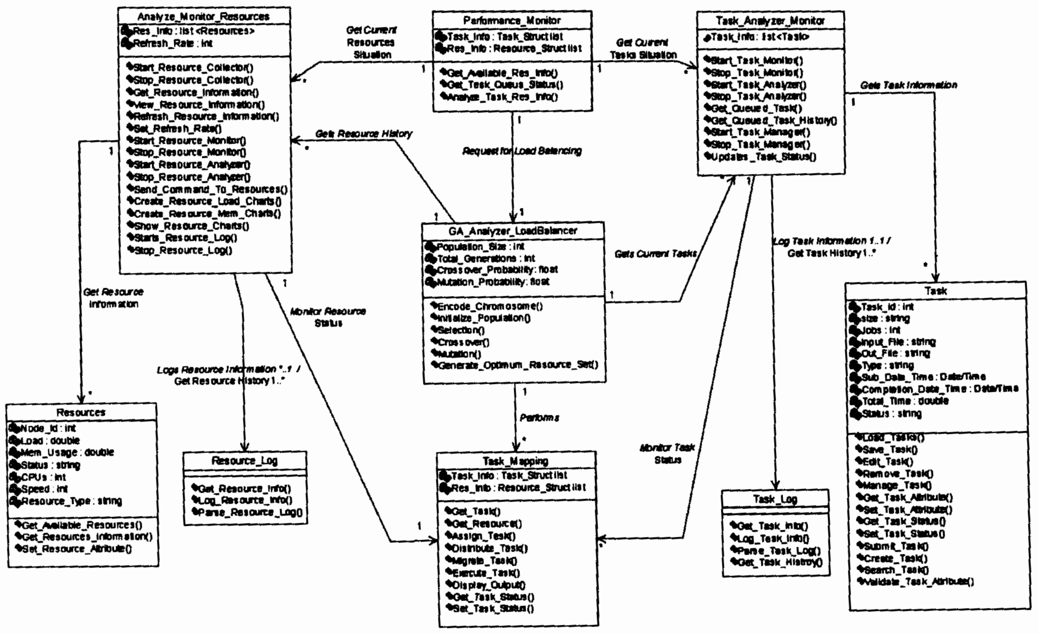


Fig. 4-1 Class Diagram

T-1502

4.1.2 SEQUENCE DIAGRAM

Sequence diagrams are temporal representation of interaction of objects. The functionality of use cases described in previous chapter is explained using following sequence diagrams described in figures 4-2 to 4-13.
Figure 4.2 explain the sequence diagram of resource collector use case. This use case is for the management of resources of grid. It then starts all the other resource handling modules of the grid like resource analyzer and resource monitor.

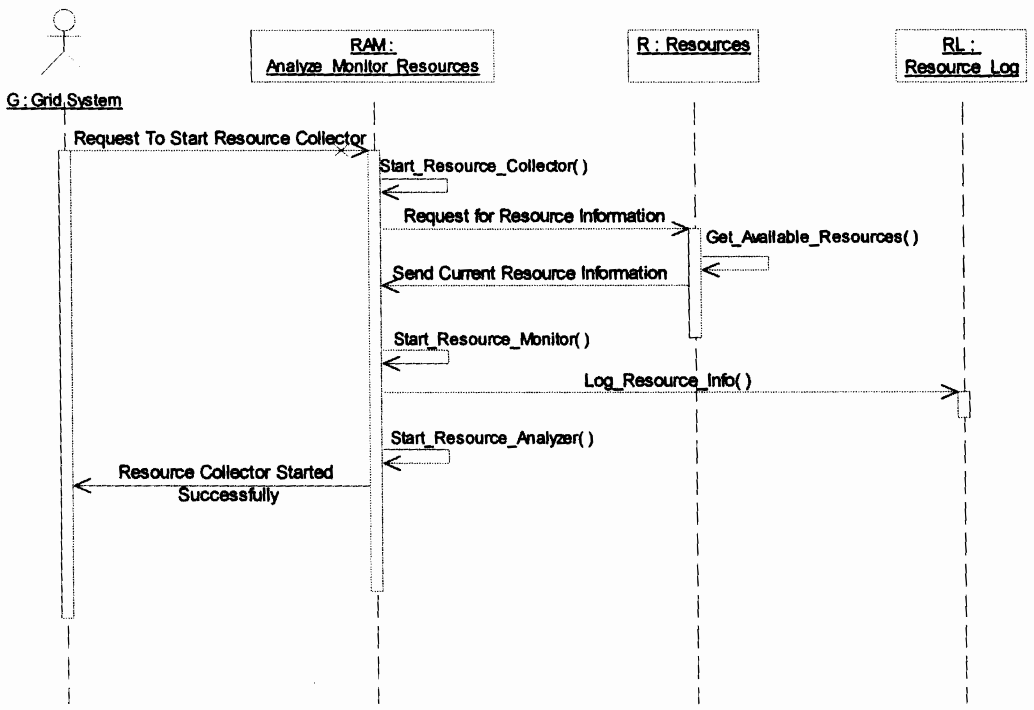

G : Grid System
RAM : Analyze Monitor Resources
R : Resources
RL : Resource Log
Request To Start Resource Collector
Start_Resource_Collector( )
Request for Resource Information
Get_Available_Resources( )
Send Current Resource Information
Start_Resource_Monitor( )
Log_Resource_Info( )
Start_Resource_Analyzer( )
Resource Collector Started Successfully

Fig. 4-2 Sequence diagram of Collecting Resource Information Use Case

Sequence of resource monitor use case is explained in figure 4-3. This use case continuously monitors the resources, updates its information after specified interval of time and logs the information.

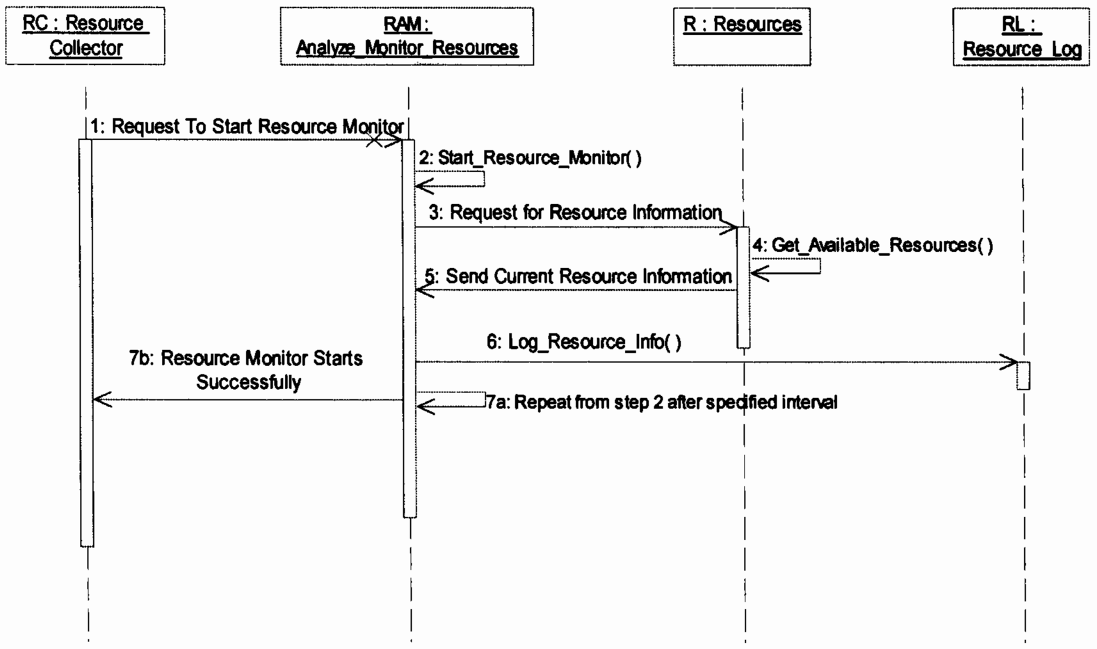

RC : Resource Collector
RAM: Analyze Monitor Resources
R : Resources
RL : Resource Log
1: Request To Start Resource Monitor
2: Start_Resource_Monitor( )
3: Request for Resource Information
4: Get_Available_Resources( )
5: Send Current Resource Information
6: Log_Resource_Info( )
7b: Resource Monitor Starts Successfully
7a: Repeat from step 2 after specified interval

Fig. 4-3 Sequence diagram of Monitoring Resource Information Use Case

Sequence of resource analyzer use case is explained in figure 4-4. This use case retrieves the resource information form resource log and creates different kinds of charts to show the resource performance.

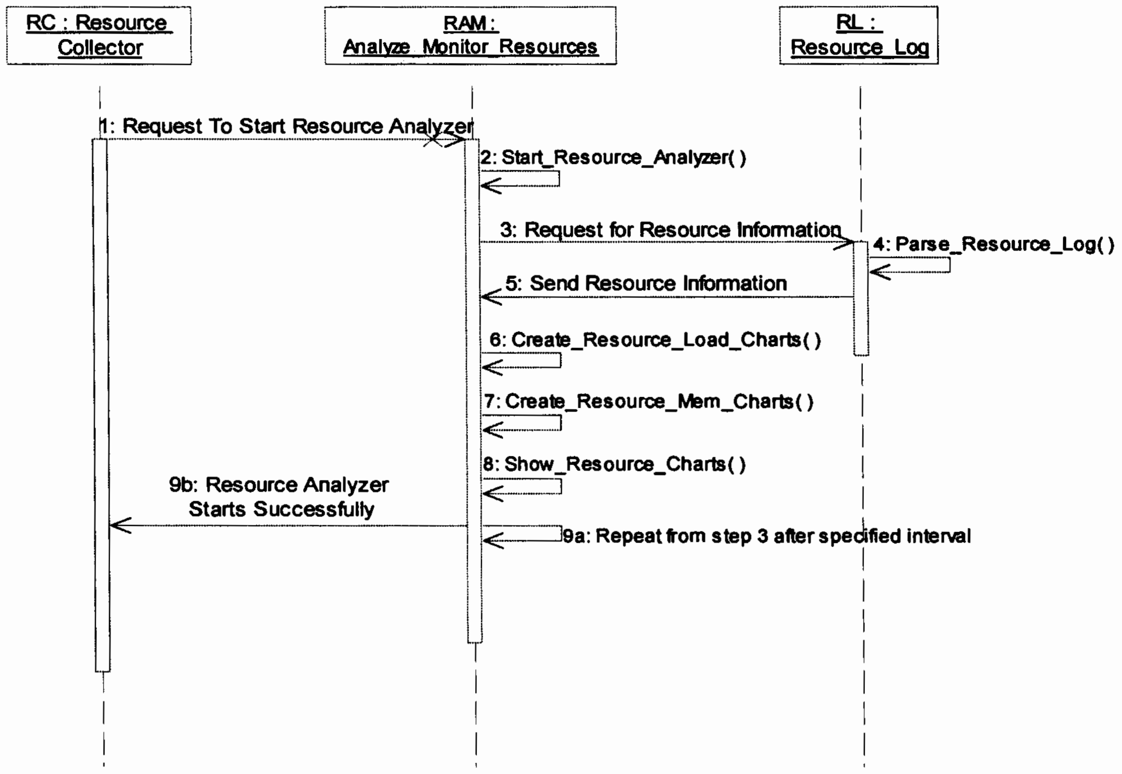

RC : Resource Collector
RAM : Analyze Monitor Resources
RL : Resource Log
1: Request To Start Resource Analyzer
2: Start_Resource_Analyzer( )
3: Request for Resource Information
4: Parse_Resource_Log( )
5: Send Resource Information
6: Create_Resource_Load_Charts( )
7: Create_Resource_Mem_Charts( )
8: Show_Resource_Charts( )
9b: Resource Analyzer Starts Successfully
9a: Repeat from step 3 after specified interval

Fig. 4-4 Sequence diagram of Analyzing Resource Information Use Case

Figure 4-5 describes the sequence of task collector. Task collector use case handles the user or client side of grid that is it receives task for execution on grid. This is the main client front end module and manages other sub task regarding modules like task manager, task monitor and task analyzer.

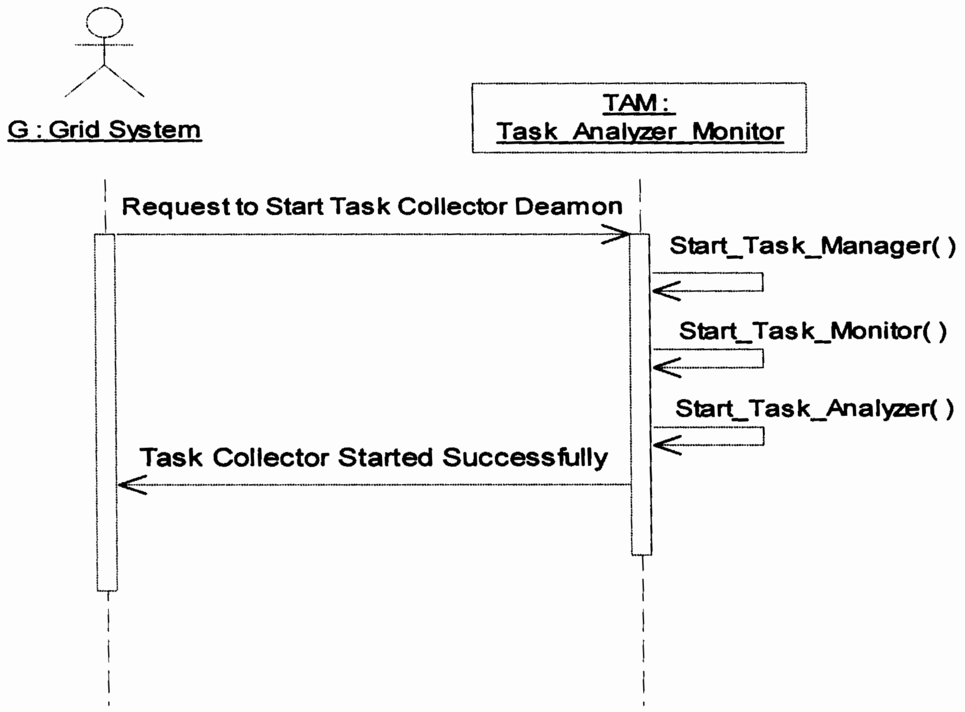


Fig. 4-5 Sequence diagram of Collecting Task Information Use Case

The sequence of Load Task use case is explained in figure 4-6. This use case handle requests for task submission from client.

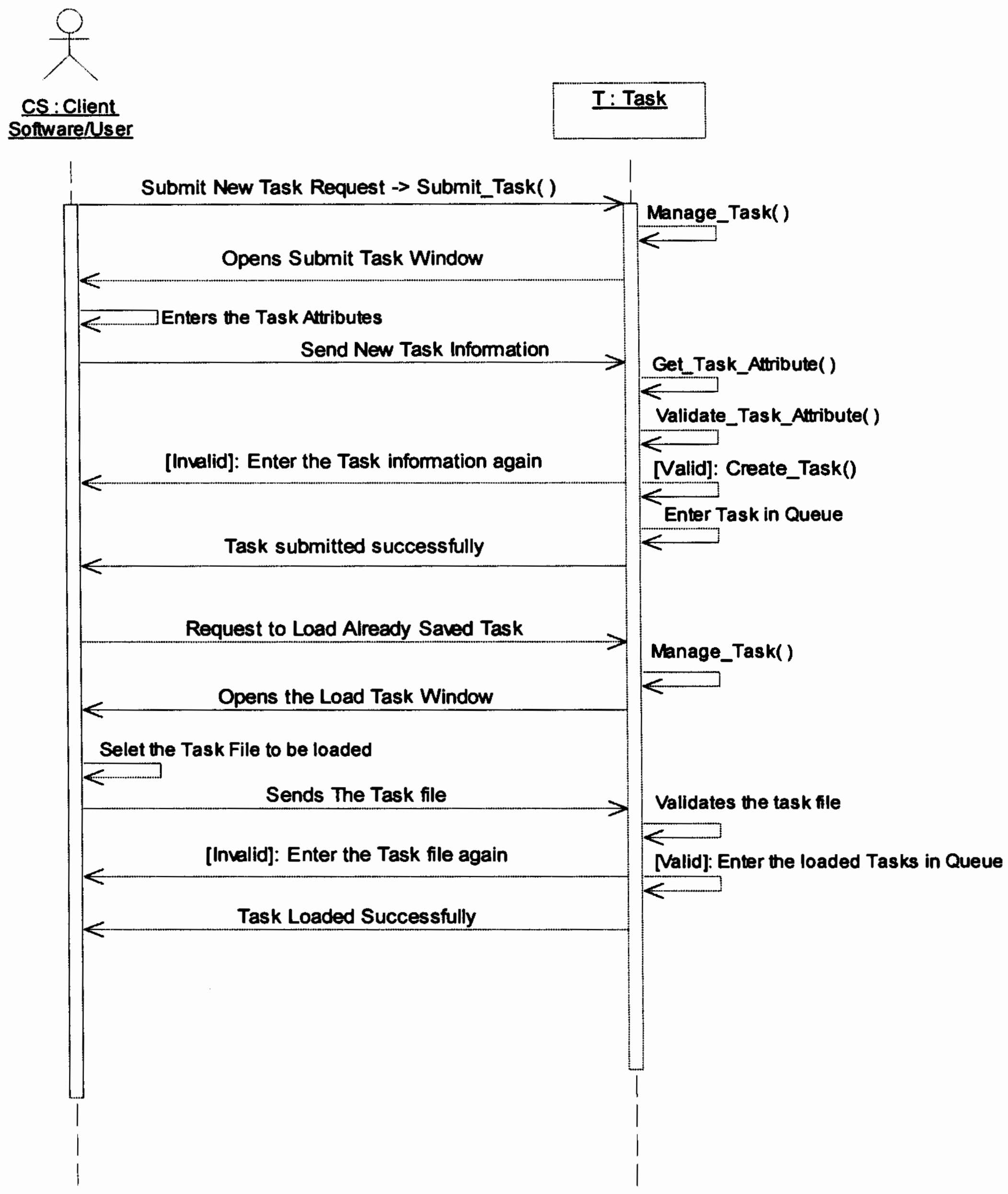

CS : Client Software/User
T : Task
Submit New Task Request -> Submit_Task( )
Manage_Task( )
Opens Submit Task Window
Enters the Task Attributes
Send New Task Information
Get_Task_Attribute( )
Validate_Task_Attribute( )
[Invalid]: Enter the Task information again
[Valid]: Create_Task()
Enter Task in Queue
Task submitted successfully
Request to Load Already Saved Task
Manage_Task( )
Opens the Load Task Window
Selet the Task File to be loaded
Sends The Task file
Validates the task file
[Invalid]: Enter the Task file again
[Valid]: Enter the loaded Tasks in Queue
Task Loaded Successfully

Fig. 4-6 Sequence diagram of Load Task Use Case

The sequence of edit task use case is described in figure 4-7. This use case handle the edit task attributes request from user. It takes new attributes, validates them and enters them in the task queue.

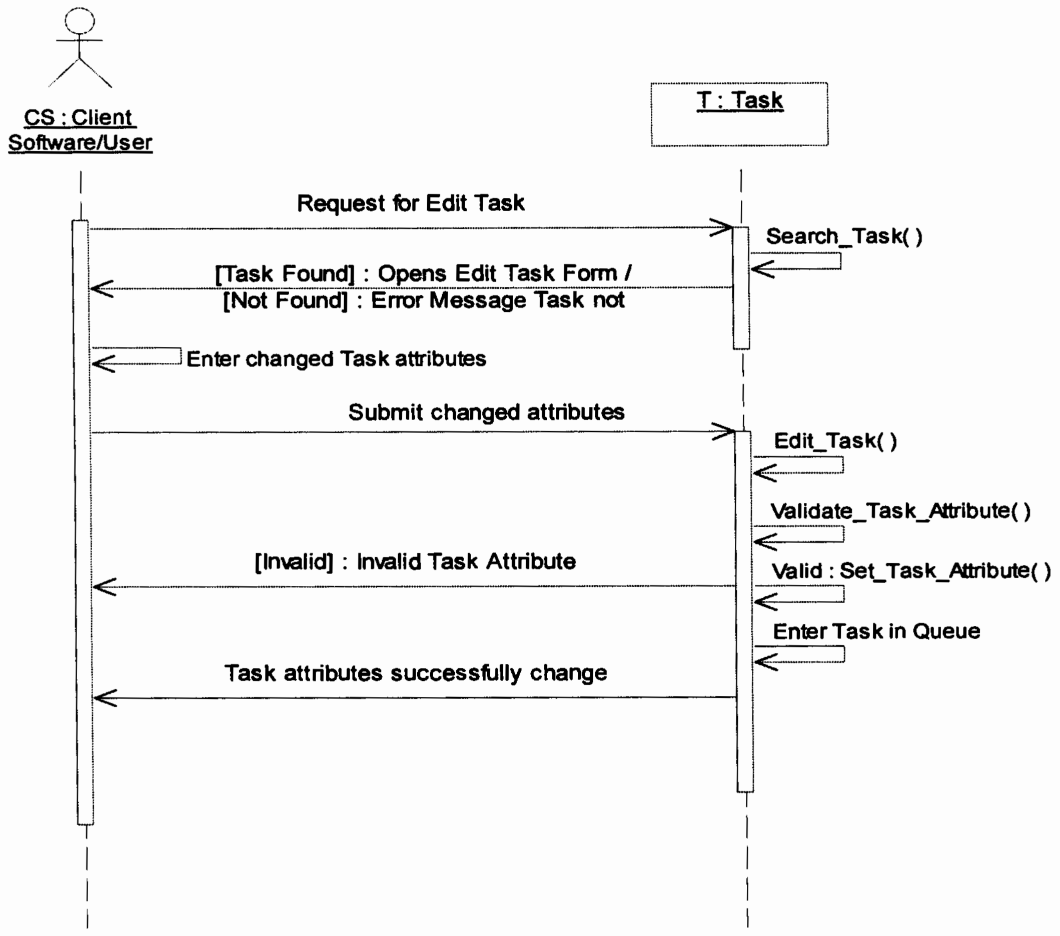

CS : Client Software/User
T : Task
Request for Edit Task
Search_Task( )
[Task Found] : Opens Edit Task Form / [Not Found] : Error Message Task not
Enter changed Task attributes
Submit changed attributes
Edit_Task( )
Validate_Task_Attribute( )
[Invalid] : Invalid Task Attribute
Valid : Set_Task_Attribute( )
Enter Task in Queue
Task attributes successfully change

Fig. 4-7 Sequence diagram of Edit Task Use Case

The sequence of save task use case is described in figure 4-8. This use case saves all the tasks currently in the task queue either finished or queued.

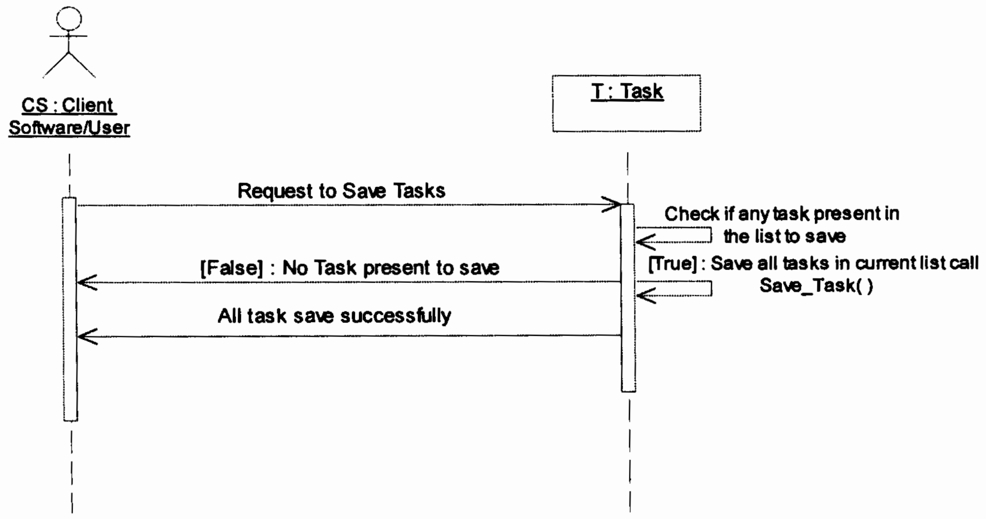

CS : Client Software/User
T : Task
Request to Save Tasks
Check if any task present in the list to save
[False] : No Task present to save
[True] : Save all tasks in current list call Save_Task( )
All task save successfully

Fig. 4-8 Sequence diagram of Save Task Use Case

The sequence of remove task use case is described in figure 4-9. This use case removes the specified task from the task queue.

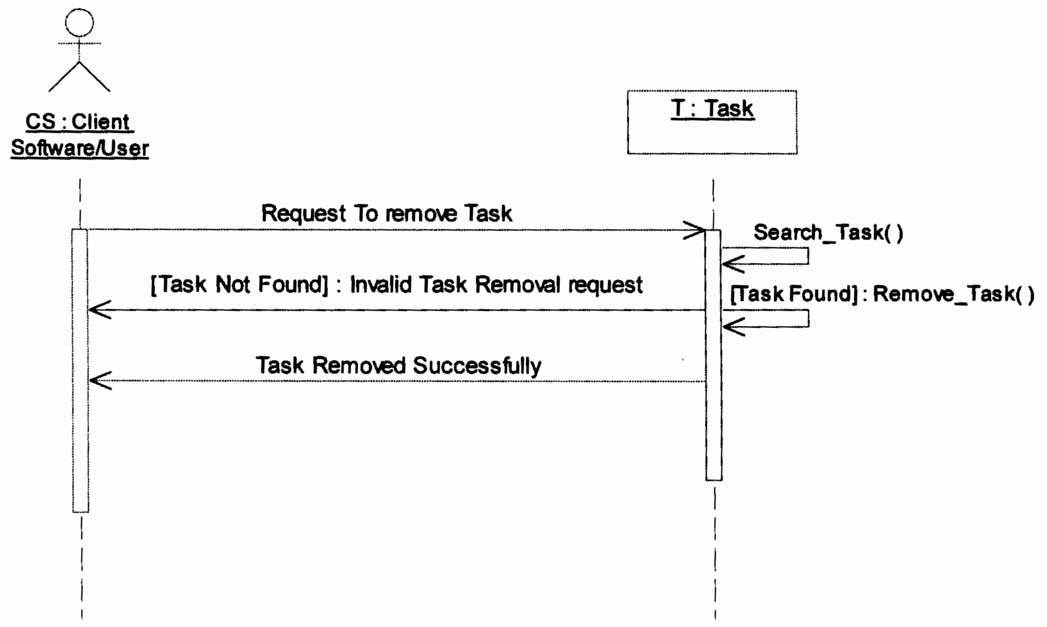

CS : Client Software/User
T : Task
Request To remove Task
Search_Task( )
[Task Not Found] : Invalid Task Removal request
[Task Found] : Remove_Task( )
Task Removed Successfully

Fig. 4-9 Sequence diagram of Remove Task Use Case

The sequence of task monitoring use case is described in figure 4-10. This use case is one of the main modules of task collector. This use case continuously monitors the tasks and their status (executing, running, scheduled etc).

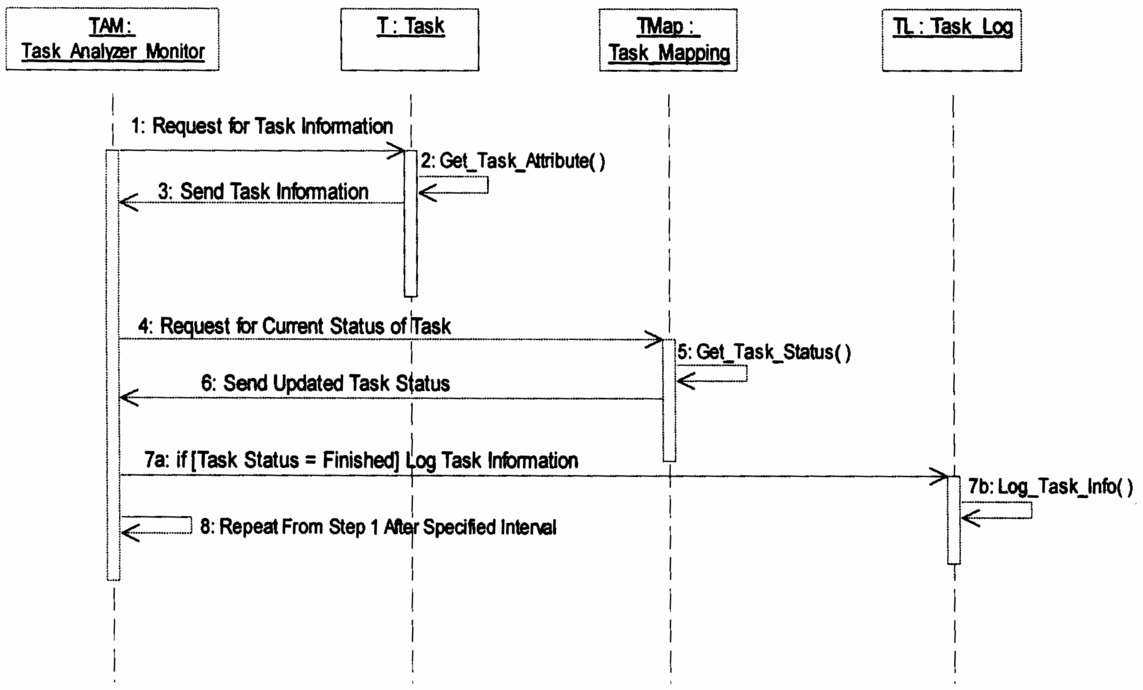

TAM : Task Analyzer Monitor
T : Task
TMap : Task Mapping
TL : Task Log
1: Request for Task Information
2: Get_Task_Attribute( )
3: Send Task Information
4: Request for Current Status of Task
5: Get_Task_Status( )
6: Send Updated Task Status
7a: if [Task Status = Finished] Log Task Information
7b: Log_Task_Info( )
8: Repeat From Step 1 After Specified Interval

Fig. 4-10 Sequence diagram of Task Monitoring Use Case

The sequence of task analyzing use case is described in figure 4-11. This use case is also one of the main modules of task collector. This use case analyzes the current task on the basis of its attributes and task history and gave this information to GA analyzer which predicts the task execution time on the basis of this information.

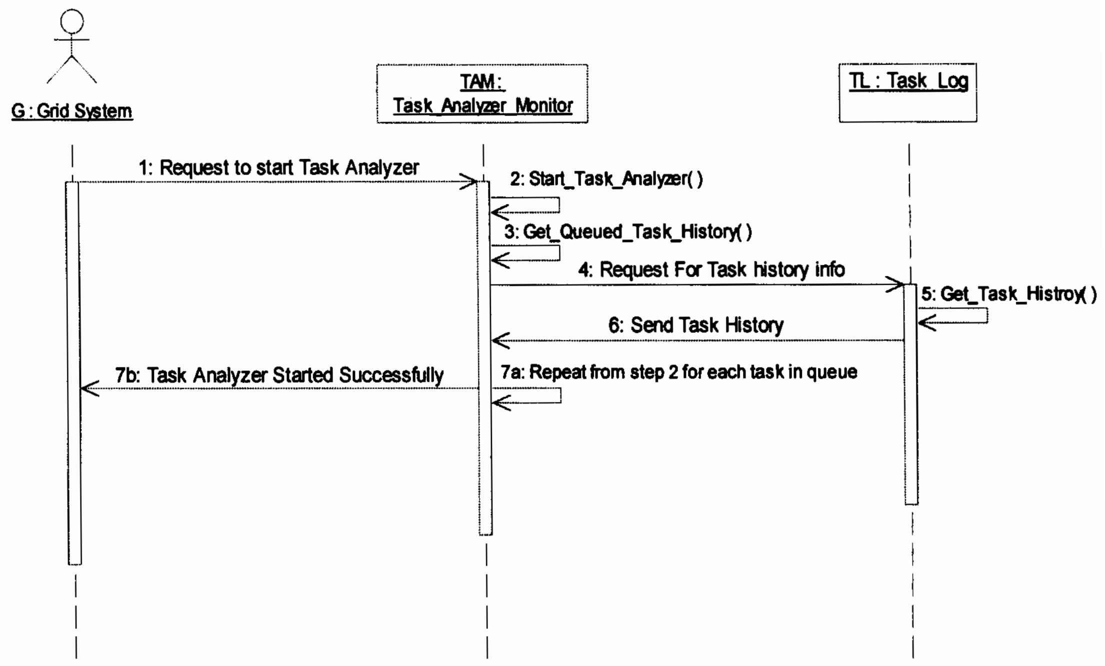

G : Grid System
TAM : Task Analyzer Monitor
TL : Task Log
1: Request to start Task Analyzer
2: Start_Task_Analyzer( )
3: Get_Queued_Task_History( )
4: Request For Task history info
5: Get_Task_Histroy( )
6: Send Task History
7a: Repeat from step 2 for each task in queue
7b: Task Analyzer Started Successfully

Fig. 4-11 Sequence diagram of Task Analyzing Use Case

Figure 4-12 describes the sequence of Performance Monitor use case. This is one of the major modules of our grid system. This use case constantly monitors the resources and task queues and whenever finds the resources over loaded palaces the request for rescheduling to GA analyzer and load balancer.

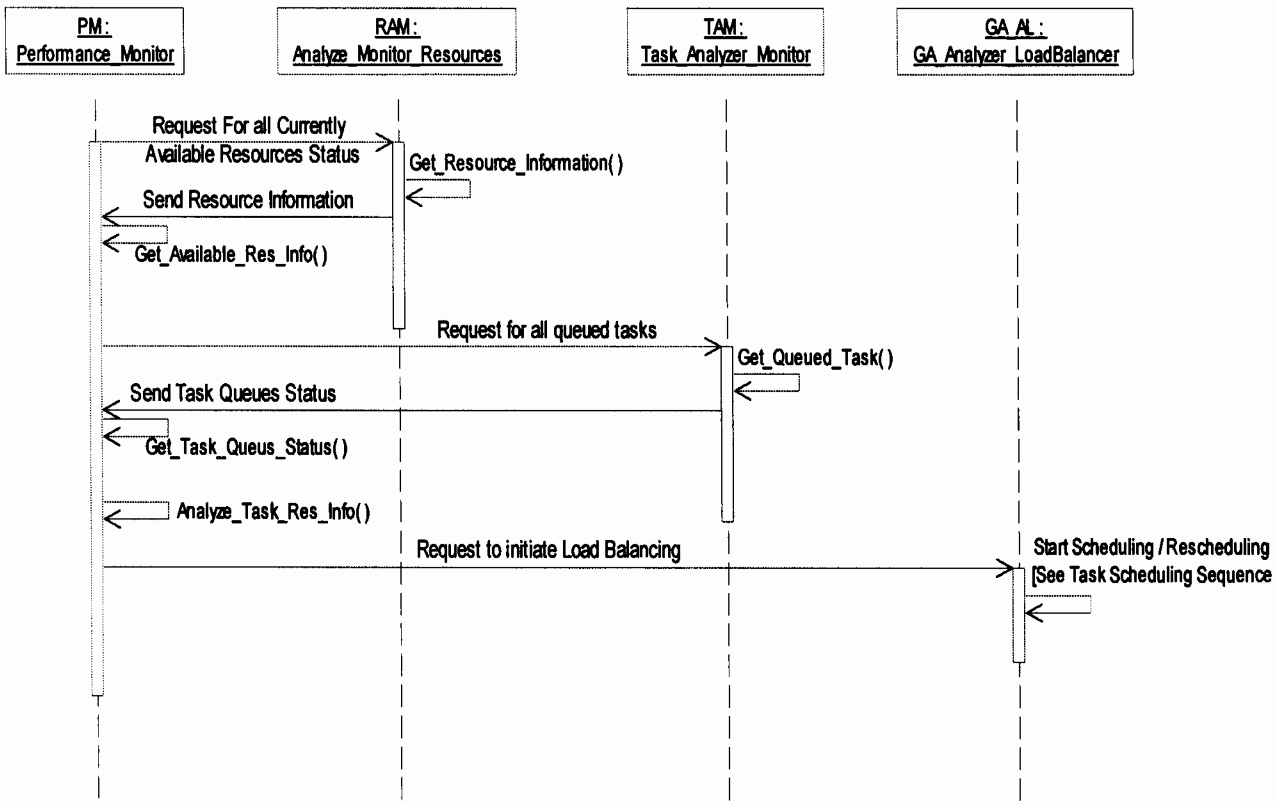

PM:
Performance Monitor
RAM:
Analyze Monitor Resources
TAM:
Task Analyzer Monitor
GA AL:
GA Analyzer LoadBalancer
Request For all Currently Available Resources Status
Get_Resource_Information( )
Send Resource Information
Get_Available_Res_Info( )
Request for all queued tasks
Get_Queued_Task( )
Send Task Queues Status
Get_Task_Queus_Status( )
Analyze_Task_Res_Info( )
Request to initiate Load Balancing
Start Scheduling / Rescheduling
[See Task Scheduling Sequence

Fig. 4-12 Sequence diagram of Performance Monitoring Use Case

Figure 4-13 describes the sequence of GA based Analyzer and Load Balancer use case. This is also one of the major and actual modules of our grid system. This use case tries to balance the load on the grid resources by producing the optimized scheduler for task allocation to resources.

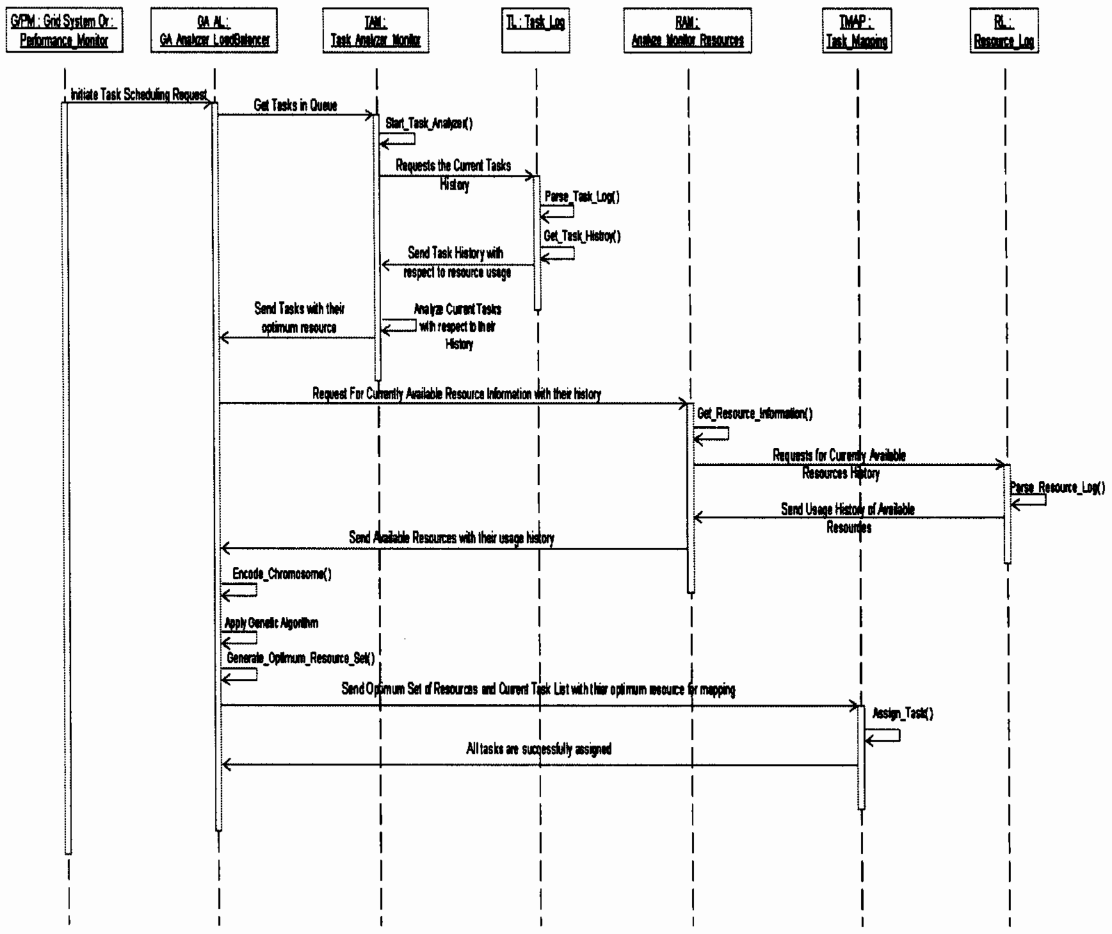

G/PM : Grid System Or : Performance_Monitor
GA AL : GA Analyzer LoadBalancer
TAM : Task Analyzer Monitor
TL : Task_Log
RAM : Analyze Monitor Resources
TMAP : Task_Mapping
RL : Resource_Log
Initiate Task Scheduling Request
Get Tasks in Queue
Start_Task_Analyzer()
Requests the Current Tasks History
Parse_Task_Log()
Get_Task_Histroy()
Send Task History with respect to resource usage
Send Tasks with their optimum resource
Analyze Current Tasks with respect to their History
Request For Currently Available Resource Information with their history
Get_Resource_Information()
Requests for Currently Available Resources History
Parse_Resource_Log()
Send Usage History of Available Resources
Send Available Resources with their usage history
Encode_Chromosome()
Apply Genetic Algorithm
Generate_Optimum_Resource_Set()
Send Optimum Set of Resources and Current Task List with their optimum resource for mapping
Assign_Task()
All tasks are successfully assigned

Fig. 4-13 Sequence diagram of GA-Base Analyzing Load Balancing and Task Mapping Use Case

4.1.3 SYSTEM ARCHITECTURE DIAGRAM

In figure 4-14 architecture of the system is illustrated in detail. System architecture provides the overall working and flow the system.

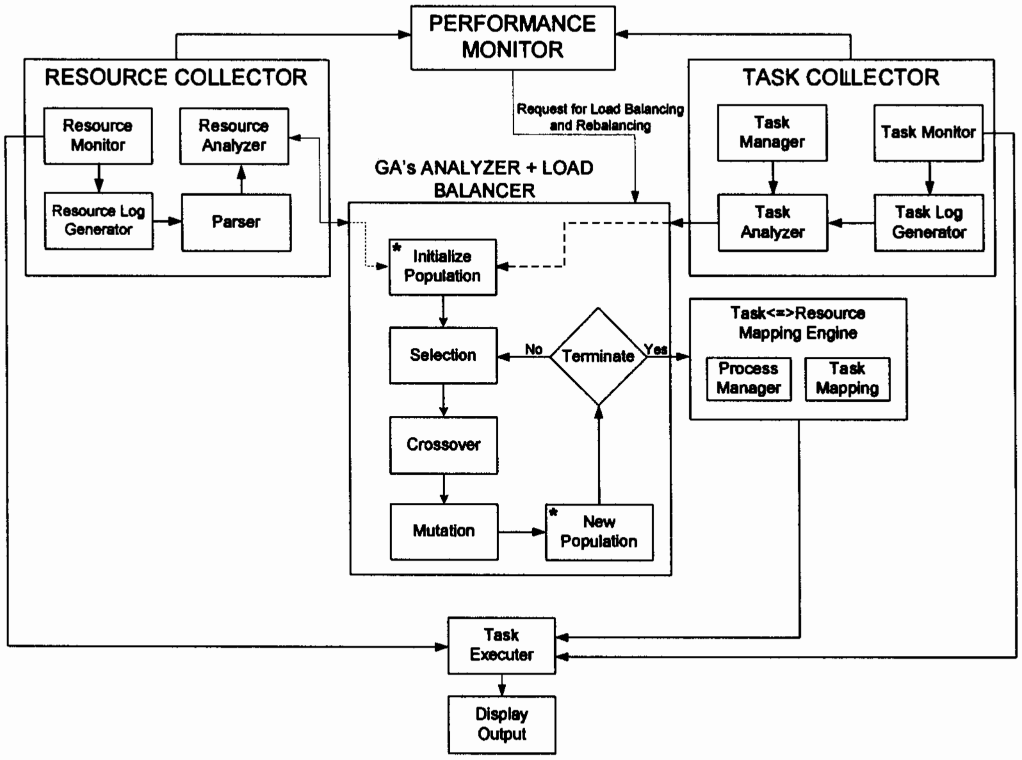


Fig. 4-14 System Architecture Diagram

CHAPTER 5

IMPLEMENTATION

5. IMPLEMENTATION

Software Development is the phase in which we transfer the proposed system into an executable software. It is the carrying out, execution, or practice of a plan, a method, or any design for doing something. Implementation is the action that must follow any preliminary thinking in order for something to actually happen.

In this phase, the final design is translated into a machine-readable form. This phase of implementation is the final phase in the software development life cycle. This phase is also critical because the designer has to look for each and every possibility that can occur during actual implementation. In case designer fails to cover all the possibilities, then it can result into disastrous situation, and can fail the whole system.

This Phase can be divided into 2 main stages:

- Tools
- Generating Code

5.1 TOOLS

Tools play a very important role in development of any system. These tools can be divided into different categories. They are as follows:

5.1.1 LANGUAGES

The major programming language used in this entire research work is C/C++. The reason for using C is that it gives easy access to resources even at very down level plus other resource monitoring and managing tools used in this research work are in C language so C is definitely required in order to modify them according to research requirement.

a) C/C++

5.1.2 EDITORS

The Editors are used to Design the interface and edit their contents. Some of the Editors used for this project are as follows:

1. KWrite
2. gedit
3. QT Designer

5.1.3 OFFICE TOOLS

Some of the office tools used in the research work is as follows:

a) Adobe Acrobat 7.0 Professional
b) Rational Rose 2002

c) Visio 2003
d) Gimp
e) MS Office XP/2006

5.1.4 GRID RESOURCE MANAGEMENT TOOL

This tool is used for task and resource monitoring and management.

a) OpenMosix View

5.2 GENERATING CODE

5.2.1 RESOURCE COLLECTOR METHODS

Resource collector manages all the resources and generates the log of each resource containing its utilization information at any particular time. Following pseudo code will explain its working.

1. Starts the resource collector.
2. Collect the utilization information from all resources. The utilization information which is monitored in this project consists of resource's status (online, offline), resource's load, total memory, current percentage utilization of memory and number of processors.
3. Display all the above mentioned information on GUI screen.
4. Continuously update this information after specified interval of time.

As mentioned above openmosix view is used as resource management tool. Openmosix provides us one solution for all above mention four steps. Openmosix View is not being used as a cluster management tool rather as a resource monitoring tool which could gather information about any kind of heterogeneous resources making a grid environment.

5.2.2 RESOURCE ANALYZER METHODS

Resource analyzer tool developed in this project analyzes all the data of each resource which is collected by resource collector in the log files. Following pseudo code will explain its working.

1. When resource analyzer starts it collects the current situation of all the resources.
2. Check resource's status for every resource then collect the timed history, which is generated by resource collector, of all the online resources.
3. Based on the time history set the threshold for each online resource. This threshold is updated after each one hour. This threshold tells us how many tasks could be assigned to that resource at any give time.

5.2.3 TASK COLLECTOR METHODS

Task collector tool developed in this project is the main GUI which interacts with user. It works according to the following pseudo code.

1. User submits the task by providing task input file, output file, type and size.
2. It calculates the task submission time and put it in the task queue.
3. On receiving run command it schedules it to the appropriate resource and starts its execution.
4. During the whole task execution period it monitors the resource to which this task is assigned.
5. After task is finished it calculates its total execution time and enters the task data i.e task type, size and submission time etc as well as the resource utilization during the execution of task into the log file.

Task scheduling tools works in the back end of task collector tool. Whenever the task comes it's the scheduler's job to assign it to the appropriate node for execution according to some criteria. In this project this criteria is genetic algorithm based. GA based scheduler and analyzer is explained in the following section.

5.2.4 GA BASED ANALYZER AND LOAD BALANCER METHODS

CHROMOSOME REPRESENTATION

We assume that the Tasks and Resources are arranged in an ascending order according to the Task attributes (that is size, type, submission time) and Resource usage (that is least loaded comes first). Figure 4 depicts the chromosome representation which is used in our current strategy. Basically the each chromosome in the population contains permutation of tasks and their fitness is calculated according to the resources they were assigned to. Task T_1 is allocated to resource R1, T_2 to R2 and T_3 to R3 and so on. When T_1 is completed, resource R1 is empty and task T_N is allocated. This procedure goes on until all the tasks are allocated.

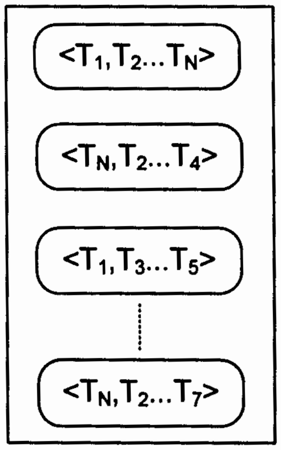


Fig. 3: Chromosome Representation

GA APPROACH FOR TASK SCHEDULING AND LOAD BALANCING

1. Get Task List from Task Collector of length TNT where TNT is the total number of tasks to be scheduled.
2. Get Resource List from Resource Collector of length TNAR where TNAR is the total number of available resources, if no resource available then wait until resources becomes available.
3. At t =0; generate an initial population with *P* chromosomes *Popi(t),* where *P* represents the permutations of tasks and is calculated as:

$$P = {}_{TNT}P_{TNT}$$

4. For each chromosome (i=1 to P), first allocate the jobs to the available resources based on the FCFS bases. Then calculate the predicted time for each task according to the resource's current parameters it is assigned to from the task history log. For example if one parameter of resource say load is used then the task time prediction formula will be follows

$$\frac{PT_{T,R}}{PL_R} = \frac{HT_{T,R}}{HL_R}$$

$$PT_{T,R} = PL_R \frac{HT_{T,R}}{HL_R}$$

Where:
$PT_{T,R}$ = Predicted Completion Time of task T on resource R
PL_R = Present Load on resource R for 1st task, and Predicted Load for next assigning tasks (Each task will increase some load on the resource which will be calculated as predicted present load for next task)
$HT_{T,R}$ = History Completion Time for task T on resource R (taken from Task Log History).
HL_R = History Load on resource R when task T had completion time $HT_{T,R}$.

Now the predicted time for each task in the chromosome can be easily calculated using above formula.

5. Fitness value for each chromosome is calculated which will tell us the make-span of schedule. Fitness value is calculated using following formula:

$$F = \frac{1}{Max(\sum_{i=1}^{TNT} PT_{Ti,R})}$$

6. The make-span of the schedule is calculated using following formula:

$$\acute{\varnothing} = Max(\sum_{i=1}^{TNT} PT_{Ti,R})$$

Where $\sum_{i=1}^{TNT} PT_{Ti,R}$ represents total number of tasks assigned to resource R.

7. Apply Crossover operator on the population according to the probability selected, Crospop(t+1) = recombined chromosomes of the population *$NewPop_i(t+1)$*;

8. Apply mutation operator on the population according to the probability selected, *MutPop(t+1)* = mutated population *CrosPop(t+1)*.

9. Evaluate the fitness of each chromosome in new population and check if the specified fitness value is achieved or not. If not starts the GAs loop again.

10. Send the specified schedule generated by GAs to Mapping Engine which will assign the tasks to the resources and in case of post scheduling Mapping Engine will migrate tasks from overloaded resources to relatively less loaded resources to which they are assigned to.

CHAPTER 6

RESULTS AND DISCUSSION

6. RESULTS AND DISCUSSION

First of all grid platform is setup on 8 machines out of which four Intel Pentium IV machines, one Intel Pentium II machine, one Intel Pentium III machine and two Intel Celeron Pentium IV machines. The software used for resource management purpose is OpenMosix which keeps up to date information about resources and keeps the log of that information which is latter used by the proposed genetic algorithm based scheduler. Then the Task Collector tool developed in this project is used for collecting task information and at the end tasks collected from task collector tool are mapped according to the genetic algorithm based scheduler to resources from resource management tool.

6.1 EXPERIMENTAL SETUP

For the performance evaluation purpose we first run bunch of tasks on limited resources (2-8 machines) without any GAs involvement and calculate the execution cost that is the total time it takes to complete all the assigns tasks on all machines. In absence of GA based scheduler OpenMosix's dynamic scheduler becomes active and performs the scheduling work. After that we replace the OpenMosix scheduler with GA based scheduler and again calculate the execution cost We calculate the execution cost for different sizes of schedules by slowly increasing the number of resources (5, 10, 15, 20, 25 and 30 number of machines) both with and without GAs and the results shows that GAs outperforms the OpenMosix scheduler in just 500 generations. The statistical results obtained are listed in the following tables. The schedule sizes used were from 10 to 100 taking an increment of 10 in each step.

6.2 EXPERIMENTAL RESULTS

As shown in table 6-1 comparison of make span time generated by GA based scheduler and OpenMosix scheduler is done for 15 and 30 number of processors and clearly the GAs outperforms the OpenMosix's scheduler.

Table 6-1 Comparison of Make span time generated by GAs and OpenMosix on different sets of schedule size and number of processors

Schedule Size	No of Processors 15		30	
	GAs	OpenMosix	GAs	OpenMosix
10	18.9173	36.501	8.536	25.5
20	53.4246	57.6	23.3635	45
30	96.3507	119.5	40.2314	77
40	154.848	174.5	64.9934	93.6
50	173.7653	211.001	73.5294	119.1
60	192.6826	247.502	82.0654	144.6
70	211.5999	284.003	90.6014	170.1
80	230.5172	320.504	99.1374	195.6
90	249.4345	357.005	107.6734	221.1
100	268.3518	393.506	116.2094	246.6

The detailed statistical results of proposed GA based technique are displayed in table 6-2 and of OpenMosix scheduler's results in table 6-3. Both tables show the results for different sets of schedule sizes with respect to different sets of processors. Optimal make span time can easily be seen in table 6-2 generated by GA for any set of schedule size and number of resources, which is much less than the make span generated by OpenMosix in table 6-3 for the same set of schedule size and number resources.

Table 6-2 Makespan time in seconds generated by GAs on different sets of schedule sizes and number of processors

GENETIC ALGORITHM BASED DYNAMIC ONLINE SCHEDULER RESULTS						
Schedule Size	No of Processors					
	5	10	15	20	25	30
10	31.8824	21.8824	18.9173	15.6805	10.2116	8.536
20	81.1515	66.78075	53.4246	44.7497	33.8824	23.3635
30	119.7331	111.0423	96.3507	90.9585	69.9869	40.2314
40	183.417	167.426	154.848	122.2548	115.5809	64.9934
50	215.2994	189.3084	173.7653	137.9353	125.7925	73.5294
60	247.1818	211.1908	192.6826	153.6158	136.0041	82.0654
70	279.0642	233.0732	211.5999	169.2963	146.2157	90.6014
80	310.9466	254.9556	230.5172	184.9768	156.4273	99.1374
90	342.829	276.838	249.4345	200.6573	166.6389	107.6734
100	374.7114	298.7204	268.3518	216.3378	176.8505	116.2094

Table 6-3 Makespan time in seconds generated by OpenMosix on different sets of schedule size and number of processors

OPENMOSIX'S DYNAMIC SCHEDULER RESULTS						
Schedule Size	No of Processors					
	5	10	15	20	25	30
10	59	43.8012	36.501	30	27.5	25.5
20	121	94.5	57.6	50	48	45
30	251.5	141	119.5	92.4	82.5	77
40	310.5	232.5	174.5	139	102.5	93.6
50	369.5	276.3012	211.001	169	130	119.1
60	428.5	320.1024	247.502	199	157.5	144.6
70	487.5	363.9036	284.003	229	185	170.1
80	546.5	407.7048	320.504	259	212.5	195.6
90	605.5	451.506	357.005	289	240	221.1
100	664.5	495.3072	393.506	319	267.5	246.6

The difference of statistical results for tables 6-3 and 6-2 can easily tells the overall performance gain of the system due to GA based technique and are shown in table 6-4.

Table 6-4 Performance gain of Genetic algorithms based scheduler over OpenMosix scheduler

PERFORMANCE GAIN OF GA's OVER OPENMOSIX						
Schedule Size	**No of Processors**					
	5	**10**	**15**	**20**	**25**	**30**
10	54.04%	49.96%	51.83%	52.27%	37.13%	33.47%
20	67.07%	70.67%	92.75%	89.50%	70.59%	51.92%
30	47.61%	78.75%	80.63%	98.44%	84.83%	26.27%
40	59.07%	72.01%	88.74%	87.95%	112.76%	69.44%
50	58.27%	68.52%	82.35%	81.62%	96.76%	61.74%
60	57.69%	65.98%	77.85%	77.19%	86.35%	56.75%
70	57.24%	64.05%	74.51%	73.93%	79.04%	53.26%
80	56.90%	62.53%	71.92%	71.42%	73.61%	50.68%
90	56.62%	61.31%	69.87%	69.43%	69.43%	48.70%
100	56.39%	60.31%	68.20%	67.82%	66.11%	47.12%

The graphical view of GA based technique (table 6-2) and OpenMosix (table 6-3) scheduler is shown in figures 6-1 and 6-2 respectively. In figure 6-1 schedule size versus make span is plotted for different number of resources for both OpenMosix and GA based scheduler.

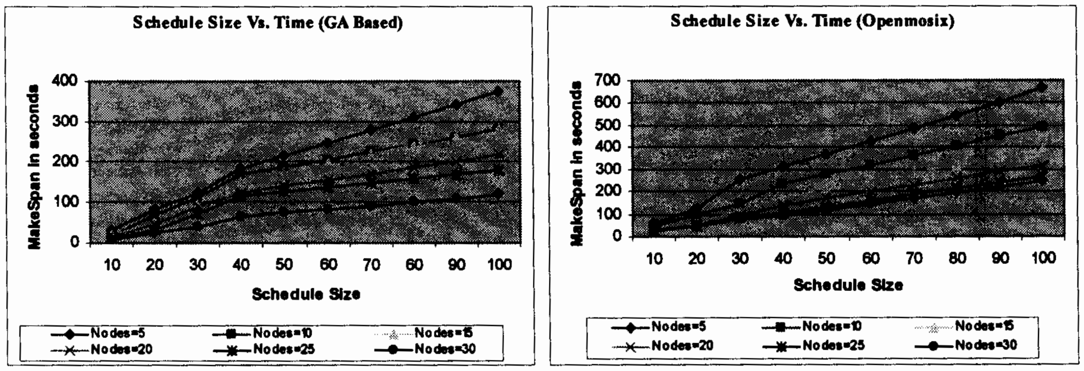


Fig. 6-1: Comparison of Genetic Algorithm Based Dynamic Online Scheduler vs. OpenMosix Scheduler (Schedule Size vs. Time)

In figure 6-2 Resources verses Make span is plotted for different lengths of schedules for both GA based and OpenMosix's scheduler.

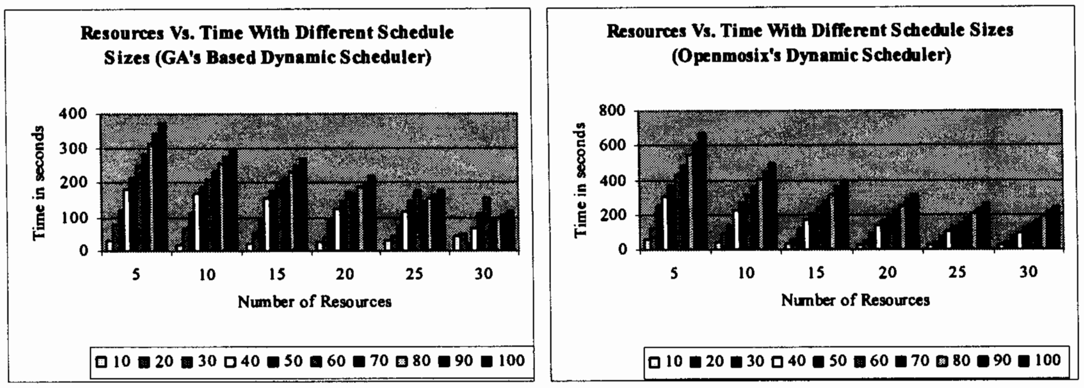


Fig. 6-2: Genetic Algorithm Based Dynamic Online Scheduler vs. OpenMosix Scheduler (Number of Resources vs. Time)

Comparing the performance of two schedules, the percentage gain in GA Based scheduler is shown in figure 6-3 for different lengths of schedules and different number of resources.

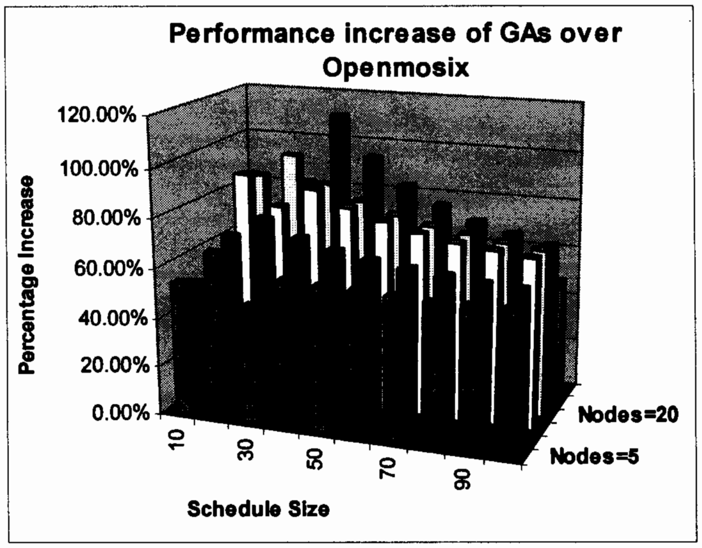
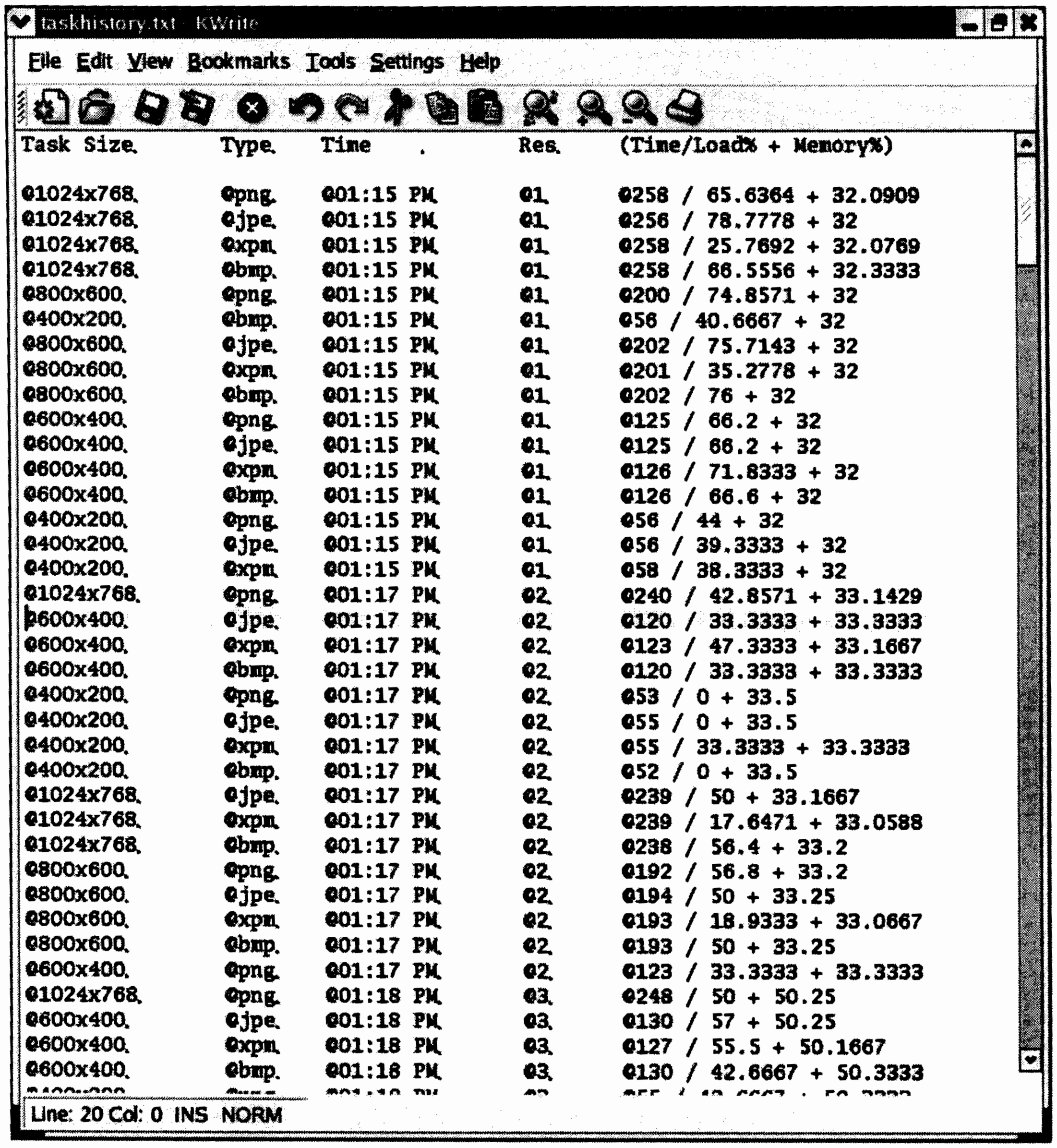


Fig. 6-3: Performance gain using genetic algorithms based dynamic scheduler over OpenMosix's dynamic scheduler

6.3 SAMPLE OUTPUT

Two types of log files are generated and used by genetic algorithms for scheduling purposes. After the task is finished the information in the following form is stored in the 1st log file. The preview of 1st log file is shown in figure 6-4.

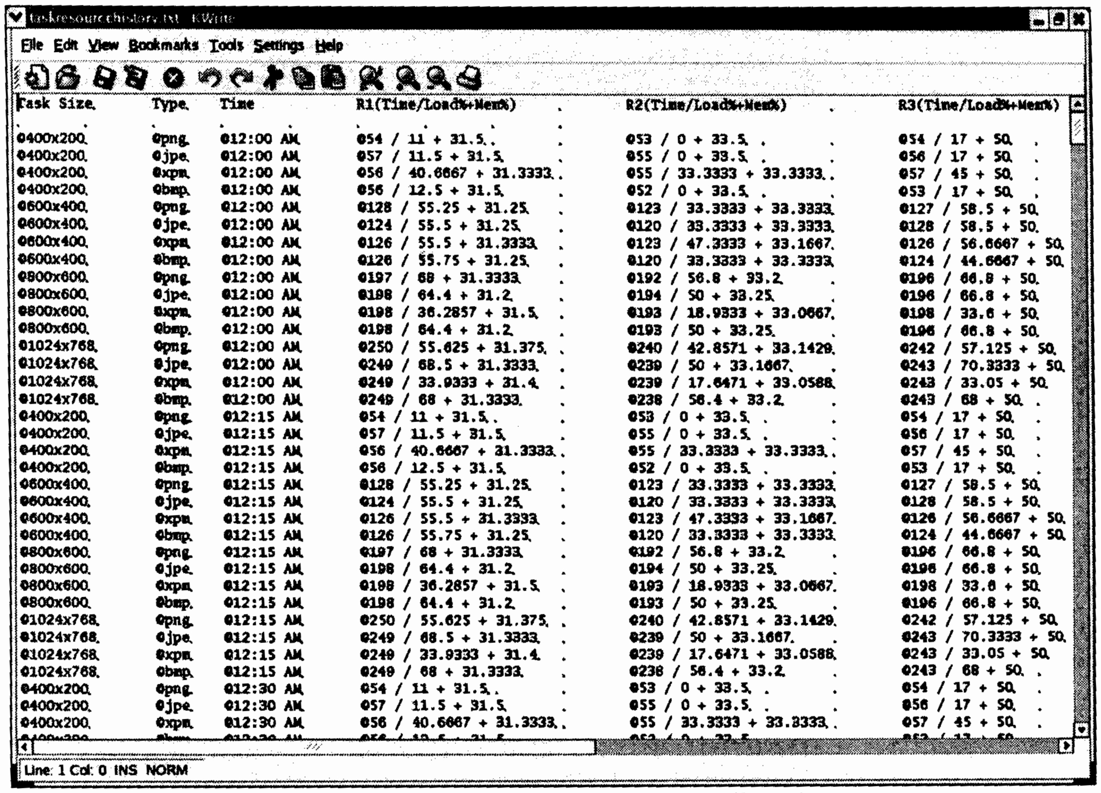


```
Task Size.      Type.   Time     .     Res.   (Time/Load% + Memory%)

@1024x768.      @png.   @01:15 PM.     @1.    @258 / 65.6364 + 32.0909
@1024x768.      @jpe.   @01:15 PM.     @1.    @256 / 78.7778 + 32
@1024x768.      @xpm.   @01:15 PM.     @1.    @258 / 25.7692 + 32.0769
@1024x768.      @bmp.   @01:15 PM.     @1.    @258 / 66.5556 + 32.3333
@800x600.       @png.   @01:15 PM.     @1.    @200 / 74.8571 + 32
@400x200.       @bmp.   @01:15 PM.     @1.    @56 / 40.6667 + 32
@800x600.       @jpe.   @01:15 PM.     @1.    @202 / 75.7143 + 32
@800x600.       @xpm.   @01:15 PM.     @1.    @201 / 35.2778 + 32
@800x600.       @bmp.   @01:15 PM.     @1.    @202 / 76 + 32
@600x400.       @png.   @01:15 PM.     @1.    @125 / 66.2 + 32
@600x400.       @jpe.   @01:15 PM.     @1.    @125 / 66.2 + 32
@600x400.       @xpm.   @01:15 PM.     @1.    @126 / 71.8333 + 32
@600x400.       @bmp.   @01:15 PM.     @1.    @126 / 66.6 + 32
@400x200.       @png.   @01:15 PM.     @1.    @56 / 44 + 32
@400x200.       @jpe.   @01:15 PM.     @1.    @56 / 39.3333 + 32
@400x200.       @xpm.   @01:15 PM.     @1.    @58 / 38.3333 + 32
@1024x768.      @png.   @01:17 PM.     @2.    @240 / 42.8571 + 33.1429
|600x400.       @jpe.   @01:17 PM.     @2.    @120 / 33.3333 + 33.3333
@600x400.       @xpm.   @01:17 PM.     @2.    @123 / 47.3333 + 33.1667
@600x400.       @bmp.   @01:17 PM.     @2.    @120 / 33.3333 + 33.3333
@400x200.       @png.   @01:17 PM.     @2.    @53 / 0 + 33.5
@400x200.       @jpe.   @01:17 PM.     @2.    @55 / 0 + 33.5
@400x200.       @xpm.   @01:17 PM.     @2.    @55 / 33.3333 + 33.3333
@400x200.       @bmp.   @01:17 PM.     @2.    @52 / 0 + 33.5
@1024x768.      @jpe.   @01:17 PM.     @2.    @239 / 50 + 33.1667
@1024x768.      @xpm.   @01:17 PM.     @2.    @239 / 17.6471 + 33.0588
@1024x768.      @bmp.   @01:17 PM.     @2.    @238 / 56.4 + 33.2
@800x600.       @png.   @01:17 PM.     @2.    @192 / 56.8 + 33.2
@800x600.       @jpe.   @01:17 PM.     @2.    @194 / 50 + 33.25
@800x600.       @xpm.   @01:17 PM.     @2.    @193 / 18.9333 + 33.0667
@800x600.       @bmp.   @01:17 PM.     @2.    @193 / 50 + 33.25
@600x400.       @png.   @01:17 PM.     @2.    @123 / 33.3333 + 33.3333
@1024x768.      @png.   @01:18 PM.     @3.    @248 / 50 + 50.25
@600x400.       @jpe.   @01:18 PM.     @3.    @130 / 57 + 50.25
@600x400.       @xpm.   @01:18 PM.     @3.    @127 / 55.5 + 50.1667
@600x400.       @bmp.   @01:18 PM.     @3.    @130 / 42.6667 + 50.3333
```

Fig. 6-4: Task History File – 1

After each twenty four hours the data from this log file is entered into the 2nd log file which is used for genetic base AI learning. The preview of 2nd log file is shown in figure 6-5.


taskresourcehistory.txt - KWrite
File Edit View Bookmarks Tools Settings Help
Line: 1 Col: 0 INS NORM

```
Task Size.      Type.   Time          R1(Time/Load%+Mem%)          .   R2(Time/Load%+Mem%)            .   R3(Time/Load%+Mem%)
.          .    .       .        .    .       .        .           .
●400x200.       ●png.   ●12:00 AM.    ●54 / 11 + 31.5..            .   ●53 / 0 + 33.5. .              .   ●54 / 17 + 50.  .
●400x200.       ●jpe.   ●12:00 AM.    ●57 / 11.5 + 31.5.           .   ●55 / 0 + 33.5. .              .   ●56 / 17 + 50.  .
●400x200.       ●xpm.   ●12:00 AM.    ●56 / 40.6667 + 31.3333.     .   ●55 / 33.3333 + 33.3333..      .   ●57 / 45 + 50.  .
●400x200.       ●bmp.   ●12:00 AM.    ●56 / 12.5 + 31.5.           .   ●52 / 0 + 33.5. .              .   ●53 / 17 + 50.  .
●600x400.       ●png.   ●12:00 AM.    ●128 / 55.25 + 31.25.        .   ●123 / 33.3333 + 33.3333.          ●127 / 58.5 + 50.
●600x400.       ●jpe.   ●12:00 AM.    ●124 / 55.5 + 31.25.         .   ●120 / 33.3333 + 33.3333.          ●128 / 58.5 + 50.
●600x400.       ●xpm.   ●12:00 AM.    ●126 / 55.5 + 31.3333.       .   ●123 / 47.3333 + 33.1667.          ●126 / 56.6667 + 50.
●600x400.       ●bmp.   ●12:00 AM.    ●126 / 55.75 + 31.25.        .   ●120 / 33.3333 + 33.3333.          ●124 / 44.6667 + 50.
●800x600.       ●png.   ●12:00 AM.    ●197 / 68 + 31.3333.         .   ●192 / 56.8 + 33.2.            .   ●196 / 66.8 + 50.
●800x600.       ●jpe.   ●12:00 AM.    ●198 / 64.4 + 31.2.          .   ●194 / 50 + 33.25.             .   ●196 / 66.8 + 50.
●800x600.       ●xpm.   ●12:00 AM.    ●198 / 36.2857 + 31.5.       .   ●193 / 18.9333 + 33.0667.          ●198 / 33.6 + 50.
●800x600.       ●bmp.   ●12:00 AM.    ●198 / 64.4 + 31.2.          .   ●193 / 50 + 33.25.             .   ●196 / 66.8 + 50.
●1024x768.      ●png.   ●12:00 AM.    ●250 / 55.625 + 31.375.      .   ●240 / 42.8571 + 33.1429.          ●242 / 57.125 + 50.
●1024x768.      ●jpe.   ●12:00 AM.    ●249 / 68.5 + 31.3333.       .   ●239 / 50 + 33.1667.           .   ●243 / 70.3333 + 50.
●1024x768.      ●xpm.   ●12:00 AM.    ●249 / 33.9333 + 31.4.       .   ●239 / 17.6471 + 33.0588.          ●243 / 33.05 + 50.
●1024x768.      ●bmp.   ●12:00 AM.    ●249 / 68 + 31.3333.         .   ●238 / 58.4 + 33.2.            .   ●243 / 68 + 50. .
●400x200.       ●png.   ●12:15 AM.    ●54 / 11 + 31.5..            .   ●53 / 0 + 33.5. .              .   ●54 / 17 + 50.  .
●400x200.       ●jpe.   ●12:15 AM.    ●57 / 11.5 + 31.5.           .   ●55 / 0 + 33.5. .              .   ●56 / 17 + 50.  .
●400x200.       ●xpm.   ●12:15 AM.    ●56 / 40.6667 + 31.3333.     .   ●55 / 33.3333 + 33.3333..      .   ●57 / 45 + 50.  .
●400x200.       ●bmp.   ●12:15 AM.    ●56 / 12.5 + 31.5.           .   ●52 / 0 + 33.5. .              .   ●53 / 17 + 50.  .
●600x400.       ●png.   ●12:15 AM.    ●128 / 55.25 + 31.25.        .   ●123 / 33.3333 + 33.3333.          ●127 / 58.5 + 50.
●600x400.       ●jpe.   ●12:15 AM.    ●124 / 55.5 + 31.25.         .   ●120 / 33.3333 + 33.3333.          ●128 / 58.5 + 50.
●600x400.       ●xpm.   ●12:15 AM.    ●126 / 55.5 + 31.3333.       .   ●123 / 47.3333 + 33.1667.          ●126 / 56.6667 + 50.
●600x400.       ●bmp.   ●12:15 AM.    ●126 / 55.75 + 31.25.        .   ●120 / 33.3333 + 33.3333.          ●124 / 44.6667 + 50.
●800x600.       ●png.   ●12:15 AM.    ●197 / 68 + 31.3333.         .   ●192 / 56.8 + 33.2.            .   ●196 / 66.8 + 50.
●800x600.       ●jpe.   ●12:15 AM.    ●198 / 64.4 + 31.2.          .   ●194 / 50 + 33.25.             .   ●196 / 66.8 + 50.
●800x600.       ●xpm.   ●12:15 AM.    ●198 / 36.2857 + 31.5.       .   ●193 / 18.9333 + 33.0667.          ●198 / 33.6 + 50.
●800x600.       ●bmp.   ●12:15 AM.    ●198 / 64.4 + 31.2.          .   ●193 / 50 + 33.25.             .   ●196 / 66.8 + 50.
●1024x768.      ●png.   ●12:15 AM.    ●250 / 55.625 + 31.375.      .   ●240 / 42.8571 + 33.1429.          ●242 / 57.125 + 50.
●1024x768.      ●jpe.   ●12:15 AM.    ●249 / 68.5 + 31.3333.       .   ●239 / 50 + 33.1667.           .   ●243 / 70.3333 + 50.
●1024x768.      ●xpm.   ●12:15 AM.    ●249 / 33.9333 + 31.4.       .   ●239 / 17.6471 + 33.0588.          ●243 / 33.05 + 50.
●1024x768.      ●bmp.   ●12:15 AM.    ●249 / 68 + 31.3333.         .   ●238 / 58.4 + 33.2.            .   ●243 / 68 + 50. .
●400x200.       ●png.   ●12:30 AM.    ●54 / 11 + 31.5..            .   ●53 / 0 + 33.5. .              .   ●54 / 17 + 50.  .
●400x200.       ●jpe.   ●12:30 AM.    ●57 / 11.5 + 31.5.           .   ●55 / 0 + 33.5. .              .   ●56 / 17 + 50.  .
●400x200.       ●xpm.   ●12:30 AM.    ●56 / 40.6667 + 31.3333.     .   ●55 / 33.3333 + 33.3333..      .   ●57 / 45 + 50.  .
```

Fig. 6-5: Task Resource History File – 2

6.4 CONCLUSION AND FUTURE ENHANCEMENTS

A new Genetic Load and Time Prediction Technique is proposed for better resource optimization and task scheduling. The scheduling process in this algorithm makes use of historical data in order to predict the resource load and task execution time. This load and time prediction is then used by genetic algorithms to figure out which resource is most suitable for which task. The scheduling process is addressed in two layers namely pre scheduling and post scheduling. The newly coming problems from outside grid boundary are scheduled in the first layer that is pre scheduling. In post scheduling the load balancing of the already submitted tasks is done, that is if certain resource is found overloaded with work while some other resources are free then some of the jobs of the overloaded machine is automatically shifted to the free machines while keeping in mind the robustness, reliability and efficiency of the job as well as the time cost, communication cost and resource cost will also be considered.

6.4.1 CONCLUSION

In this research work we attempt to address the problem of load balancing in grid computing using one of the popular heuristics namely GAs. Genetic Algorithm predicts the execution time for each task with respect to resource, it is assigned to. We have tested this algorithm on 30 machines heterogeneous grid environment simulation with different schedule sizes and the results shows that the proposed strategy outperforms one of the most popular dynamic scheduler that is OpenMosix scheduler.

6.4.2 FUTURE ENHANCEMENTS

Future work will examine the application of proposed GA based algorithm as parallel genetic algorithms and dynamic distributed algorithms proposed in [31]. In both cases number and size of populations must be carefully determined. Even though significant progress has been made in modeling the infrastructure of grid computing but the close review clearly indicates that not much progress is made in formulating and efficient, globally optimized, grid-scheduling algorithm for allocating jobs [32]. Therefore we are planning to investigate this research area in depth using GAs as well as other heuristic algorithms.

APPENDIX A

BIBLIOGRAPHY AND REFERENCES

A – BIBLIOGRAPHY AND REFERENCES

[1] http://www.nature.com/nature/webmatters/grid/grid.html

[2] Surendra Reddy ; "Grid Computing: Crossing the Chasm"; 24th September 2004 retrieved from World Wide Web; http://sciencecareers.sciencemag.org/carrer_development/previous_issues/articles/grid_computing_crossing_the_chasm/(parent)/12093

[3] http://www.en.wikipedia.org/wiki/Grid_computing

[4] Matt Haynos: "Perspectives on Grid: Grid Computing—next generation distributed computing"; 27th Jan 2004; retrieved from World Wide Web; http://www-128.ibm.com/developerworks/grid/library/gr_heritage/#N1009E.

[5] http://www.cronos.be/expertise/GridComputing/

[6] M. Wieczorek, R. Prodan and T. Fahringer; "Scheduling of Scientific Workflows in the ASKALON Grid Environment"; ACM SIGMOD Record, Vol. 34 (3); ACM Press; New York, USA; September 2005.

[7] D.P. Spooner, S.A. Jarvis, J.Cao, S. Saini and G.R.Nudd; "Local Grid Scheduling Techniques using Performance Prediction"; IEE Proceedings Computers and Digital. Techniques, Vol. 150(2); Institution of Electrical Engineers; Great Britain; April 2003.

[8] R. Buyya, D. Abramson and J. Giddy; "Nimrod/G: An Architecture for a Resource Management and Scheduling system in a Global Computational Grid"; Proc. of 4th Int. Conf. On High Performance Computing, Asia-Pacific Region; Beijing, China; 2000.

[9] H. Casanova and J.Dongara; "NetSolve: A Network Server for Solving Computational Science Problems"; Int. Journal of Supercomputing Applications and HPC, Vol 11(2); IEEE Computer Society; Washington DC, USA; 1996.

[10] F.Berman, R. Wolski, S.Figueira, J.Schopf and G.Shao; "Application-level Scheduling on Distributed Heterogeneous Networks"; Proc of Supercomputing; IEEE Computer Society; Washington DC, USA;1996.

[11] A. Takefusa, S.Matsuoka, H.Nakada, K.Aida and U.Nagashima; "Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms"; Proc. of Eighth IEEE Int. Symp. On High Performance Distributed Computing; IEEE Computer Society; Washington DC, USA; 1999.

[12] M.Litzkow; M.Livny and M.Mutka; "Condor – A Hunter of Idle Workstations"; Proc of 8th Int. Conf. On Distributed Computing Systems; San Jose, California; June 13-17, 1988.

[13] S.Zhou; “LSF: Load Sharing in Large-scale Heterogenous Distributed Systems”; Proc of 1992 Workshop on Cluster Computing; 1992.

[14] G. Aloisio, M. Cafaro, E. Blasi and I. Epicoco; “The Grid Resource Broker, a Ubiquitous Grid Computing Framework”; Scientific Programming, Special issue on Grid Computing Vol 10(2); IOS Press; Amsterdam, The Netherlands; 2002

[15] J. Cao, D. P. Spooner, S.A. Jarvis and G.R. Nudd; “Grid Load Balancing Using Intelligent Agents”; Future Generation Computer Systems, Vol 21(1); Elsevier Science Publishers B.V.; Amsterdam, The Netherlands; January 2005.

[16] A. Abraham, R. Buyya and B. Nath; “Nature’s Heuristics for Scheduling Jobs on Computational Grids”; The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000); Cochin, India; December 14-16, 2000.

[17] S. Sanyal and S. K. Das; “MaTCH: Mapping Data-Parallel Tasks on a Heterogeneous Computing Platforms Using the Cross-Entropy Heuristic”; Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium; IEEE Computer Society; Washington DC, USA; 2005.

[18] R. A. Moreno; “Job Scheduling and Resource Management Techniques in Dynamic Grid Environments”; In 1st European Across Grids Conference; Springer-Verlag; Heidelberg, Germany; February 2003.

[19] S. Wagner, M. Affenzeller; “Heuristiclab Grid – A Flexible And Extensible Environment For Parallel Heuristic Optimization”; Proceedings of the 15th International Conference on Systems Science, Vol. 1; Oficyna Wydawnicza Politechniki Wroclawskiej; 2004

[20] N. Navet, J. Migge; “Fine Tuning the Scheduling of Tasks through a Genetic Algorithm: Application to Posix1003.1b compliant Systems”; IEE Proceedings Software, Vol. 150(1); Feb 2003.

[21] M. Grajcar; “Conditional Scheduling for Embedded Systems Using Genetic List Scheduling”; Proceedings of the 13th international symposium on System synthesis; IEEE Computer Society; Washington, DC, USA; 2000.

[22] A. Y. Zomaya and Y.H. Teh; “Observations on Using Genetic Algorithms for Dynamic Load Balancing”; IEEE Transactions on Parallel and Distributed Systems, Vol. 12 (9); IEEE Press; September 2001

[23] X. Li, M. J. Garzaran and D. Padua; “Optimizing Sorting with Genetic Algorithms”; Proc. of the International Symposium on Code Generation and Optimization (CGO’05); IEEE Computer Society; Washington DC, USA; 2005.

[24] W. A. Greene; “Dynamic Load Balancing via a Genetic Algorithm”; Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01); IEEE Computer Society; Washington DC, USA; 2001.

[25] W. Yi, Q. Liu, Y. He; "Dynamic Distributed Genetic Algorithms"; Proceedings of the 2000 Congress on Evolutionary Computation CEC00; IEEE Press; California, USA; 2000.

[26] S. Song, Y. K. Kwok and K. Hwang; "Security-Driven heuristics and A FAST Genetic Algorithm for Trusted Grid Job Scheduling"; Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium; IEEE Computer Society; Washington DC, USA; 2005.

[27] S. Kim and J. B. Weissman; "A GA-based Approach for Scheduling Decomposable Data Grid Applications"; Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04) – Vol. 00; IEEE Computer Society; Washington DC, USA; 2004.

[28] S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam and J. Velazco; "Mapping of Subtasks with Multiple Versions in a Heterogeneous Ad Hoc Grid Environment"; Parallel Computing, Vol. 31(7); Elsevier Science Publishers B. V. Amsterdam, The Netherlands; July, 2005.

[29] Y. Tanimura, T. Hiroyasu, M. Miki and K. Aoi; "The System for Evolutionary Computing on the Computational Grid"; Proceedings of Parallel and Distributed Computing and Systems; ACTA Press; Cambridge, USA; 2002.

[30] T. Jing, M. H. Lim and Y. S. Ong; "Island Model Parallel Hybrid-GA for Large Scale Combinatorial Optimization"; The Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV2004), Special Session on Computational Intelligence on the Grid; Kunming, China; December 6-9, 2004.

[31] J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini and G. R. Nudd; "Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling"; Proceedings of the 17th International Symposium on Parallel and Distributed Processing; IEEE Computer Society; Washington DC, USA; 2003.

[32] Bart Jacob, Michael Brown, Kentaro Fukui and Nihar Trivedi; "Introduction to Grid Computing"; IBM Press 2005

[33] Luis Ferreira, Viktors Berstis, Jonathan Armstrong, Mike Kendzierski, Andreas Neukoetter, Masanobu Takagi, Richard Bing-Wo, Adeeb Amir, Ryo Murakawa, Olegario Hernandez, James Magowan, Norbert Bieberstein; "Introduction to Grid Computing with Globus"; IBM Press 2003

[34] David E. Goldberg; "Genetic Algorithms in Search, Optimization, and Machine Learning "; Pearson Education Inc. 1989

[35] Michael D. Vose, "The Simple Genetic Algorithm Foundations and Theory" Massachusetts Institute of Technology; 1999

[36] Mitchell Melanie; "An Introduction to Genetic Algorithms 5th Ed." Massachusetts Institute of Technology, 1999

[37] John R. Koza, "Genetic Programming: a paradigm for genetically Breeding populations of computer programs to Solve problems" Stanford University Computer Science Department technical report STAN-CS-90-1314. June 1990.

[38] George F. Luger, "Artificial Intelligence Structures and Strategies for Complex Problem Solving 4th Ed.", Åddison Wesley, 2001

[39] Neil Matthew and Richard Stones, "Beginning Linux Programming 3rd Ed." Wiley Publishing Inc. 2004

[40] Bill Ball and Hoyt Duff, "Red Hat Linux 9 Unleashed", Sams, May 8, 2003

[41] W. Richard Stevens, "UNIX Network Programming: Inter process Communications, Volume 2, 2nd Ed." Pearson Education Inc. 1995

[42] Craig Larman "Applying UML and Patterns". Prentice Hall, 1998.

APPENDIX B

PUBLICATIONS

Genetic Load and Time Prediction Technique for Dynamic Load Balancing in Grid Computing

Zahida Akhtar
zahida.akhtar@gmail.com
Department of Computer Science
Faculty of Applied Sciences
International Islamic University Islamabad
Pakistan

ABSTRACT

Gird computing is an emerging science in the field of distributed computing which involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations. Scheduling and load balancing techniques are critical issues in grid computing for achieving good performance. The goal of load balancing is to minimize the response and execution time of a program by trying to equally spread the load on processors and maximizing their utilization. It has been proven that finding optimal schedules for the load-balancing problem is an NP-complete problem, even when the communication cost is ignorable. Genetic algorithms are a probabilistic search approach, which are founded on the ideas of evolutionary processes. They are particularly applicable to problems that are large, non-linear and possibly discrete in nature; features that traditionally add to the degree of complexity of solution.

Present research aims to solve the grid load-balancing problem using Genetic Algorithms. A new Genetic Algorithm based task scheduling technique is introduced, which has been tested on a multi-node grid environment and the experimental results show that this new technique can lead to significant performance gain in various applications.

Keywords: Genetic Algorithms (GA), load balancing, grid computing, distributed computing, heterogeneous computing, evolutionary computing.

INTRODUCTION

Grid Computing, as defined by its founders, has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation (Foster et al. 2001 pp: 1).

Grid computing is a new technology that transforms a computer infrastructure into an integrated, pervasive virtual environment for dynamic collaboration and shared resources anywhere in the world providing users, especially in science, with unprecedented computing power, services and information (Reddy 2004). With the advance in technologies, the cost of computation resources required per operation is continuously decreasing. Grid computing is one of many factors which enable the effective use of wide spread computing resources thereby providing non-trivial services to users. According to the Department of Computer Science at the University of Warwick, with the emergence of grid environments featuring dynamic resources and varying user profiles, there is an increasing need to develop reliable tools that can effectively coordinate the requirements of an application with available computing resources. The ability to predict the behaviour of complex aggregated systems under dynamically changing workloads is particularly desirable, leading to effective resource usage and optimization of networked systems.

Scheduling and load balancing techniques are critical issues in grid computing for achieving optimum performance. The goal of load balancing is to minimize the response and execution time of a program by trying to equally spread the load on processors and maximizing their utilization. Present research aims to solve the grid load-balancing problem using Genetic Algorithms.

Genetic algorithms are based on a biological metaphor: They view learning as a competition among a population of evolving candidate problem solutions. A 'fitness' function evaluates each solution to decide whether it will contribute to the next generation of solutions. Then, through operations analogous to gene transfer in sexual reproduction, the algorithm creates a new population of candidate solutions (Melanie 1999), (Koza 1990), (Goldberg 1989), (Vose 1999), (Rennard 2000).Genetic Algorithms are nondeterministic stochastic search/optimization methods that utilize the theories of evolution and natural selection to solve a problem within a complex solution space. They are computer-based problem solving systems which use computational models of some of the known mechanisms in evolution as key elements in their design and implementation (SANDIKCI 2000).

The rest of the paper is structured as follows. Section 2 will provide some background work related to load balancing, grid computing and genetic algorithms. Section 3 provides the problem statement. Section 4 will describe our methodology on how genetic algorithms can be applied to the grid load balancing problem. Then results, conclusion and future work are discussed in section 4 and 5 respectively.

BASIC CONCEPT

Recent research studies indicate that genetic algorithms significantly enhance the performance of real time applications. In one such study done by Wu and Chau (2006), authors employ the hybrid genetic algorithm based artificial neural network model for flood prediction. The proposed model is tested against empirical linear regression model, conventional ANN model and a GA model, and results reveals that proposed hybrid GA-based ANN algorithm outperforms the conventional models. The limiting factor of this research study is that it requires additional modelling parameters and longer computation time.

In another study, Chau and Albermani (2003) develop the prototype system by coupling the blackboard architecture, an expert system shell VISUAL RULE STUDIO and genetic algorithm (GA). Chau and Albermani (2003) proposed the said system for the optimize design of liquid retaining structures which can act as a consultant to assist novice designers in the design of liquid retaining structures. Near-optimal solutions are claimed to be achieved after exploration of small portion of search space at extraordinarily converging speed.

Another example of genetic algorithm in a real time application is, Usage of parallel genetic algorithm for multiple criteria rainfall–runoff model calibration which is proposed by Cheng et al (2005). The method uses the fuzzy optimal model to evaluate multiple alternatives with multiple criteria where chromosomes are the alternatives, whilst the criteria are flood performance measures Cheng et al 2005). The proposed approach produces the similar results when compared with results obtained by using a two-stage calibration procedure but it significantly reduces the overall optimization time and improves the solution quality. The disadvantage of this research study is that it splits the whole procedure into two parts which makes it difficult to integrally grasp the best behaviours of model during the calibration procedure. In Continuation to study (Cheng et al 2005) Cheng et al (2006) proposed a new method to the multiple criteria parameter calibration problem, which combines GA with TOPSIS for Xinanjiang model. Cheng et al (2006) removes the disadvantage of their previous research study and integrates the two parts of Xinanjiang rainfall-runoff model calibration together thus simplifying the procedures of model calibration and validation. Comparison of results with two-step procedure shows that the proposed methodology gives similar results to the previous method, is also feasible and robust, but simpler and easier to apply in practice.

In another study, Chau (2004) proposed a two-stage dynamic model to assist construction planners to formulate the optimal strategy for establishing potential intermediate transfer centres for site-level facilities such as batch plants, lay-down yards, receiving warehouses, various workshops, etc. Under the proposed approach, the solution of the problem is split into two stages, namely, a lower-level stage and an upper-level stage. Standard linear programming method is used to solve former stage whereas the latter is solved by a genetic algorithm. The efficiency of the proposed algorithm is demonstrated through case examples.

In this research study the problem area has been described, which is mostly based on the description of Dong and Akl (2006). While different approaches have used GAs for solving load balancing problems yet the issues that remain to be addressed can be broadly categorized as the following.

a) The execution time for load balancing has not been considered or has not been quantitatively described.
b) Most of the algorithms are restricted to static load balancing and as such require prior knowledge of various parameters. While this approach may work in problems of equivalent nature but cannot be broadly applied to different applications.
c) A few dynamic load balancing algorithms that have been studied and are also mentioned in the literature review have not been implemented in loosely coupled systems such as grid computing.
d) To the best of the author's knowledge, no algorithm has been designed to prevent resubmission in case of load failure. The algorithms that incorporate fault tolerance use a simple strategy for restarting the task which in some cases requires extensive overheads.

Efficient execution in a distributed system can require, in the general case, mechanisms for the *discovery* of available resources, the *selection* of an application-appropriate subset of those resources, and the *mapping* of data or tasks onto selected resources.

Grid computing has become an increasingly popular solution to optimize resource allocation in highly charged IT environments. In one of the recent research studies done by Wieczorek et al (2005) three different algorithms (namely HEFT, Genetic and simple Myopic algorithm) are compared in terms of incremental versus full-graph scheduling for balanced versus unbalanced workflows. Without considering effect of the typical network scenarios Wieczorek et al (2005) declare HEFT as better algorithm. A multi-tiered framework based on, Globus providers, distribution brokers and local schedulers is used for grid work load management, out of which only lowest tier is primary focused by Spooner et al (2003). Spooner and his colleague use iterative heuristic algorithm and performance prediction techniques for performance based upon global and local scheduling. The future work of Spooner et al for examining the other two upper tiers is still on its way. Cao et al (2005) addresses grid load balancing issues using a combination of intelligent agents and multi-agent approaches. The experimental result of research study of Cao et al (2005) proves that the use of a distributed agent strategy can reduce the network overhead significantly and make the system scale well??? rather than using a centralized control, as well as achieving a reasonable good resource utilization and meeting application execution deadlines. Abraham et al (2000) addressed the hybridization of the three popular nature's heuristics namely Genetic Algorithms (GA), Simulated Annealing

(SA) and Tabu Search (TS) for dynamic job scheduling on large-scale distributed systems but didn't provide any experimental results for research evaluation. A novel mapping heuristic based on the cross-entropy (CE) method, for mapping a set of interacting tasks of a parallel application onto a heterogeneous computing platform, was proposed by Sanyal and Das (2005). According to their research studies, Cross Entropy methods are inherently slow and this slowness of the CE based methods in generating the appropriate mapping can decrease the performance gain for a large set of tasks. Moreno (2003) proposed new rescheduling policies for job migration under cost constraints after analysing the main tasks that the grid resource broker has to tackle (like resource discovery and selection, job scheduling, job monitoring and migration etc.) in detail. Wagner and Affenzeller (2004) present a new environment for parallel heuristic optimization based upon the already proposed Heuristic-Lab in.

A Formal model which allows multiple schedule optimizations and a new efficient heuristic approach based on genetic algorithms and list scheduling is presented by Grajcar (2000). In spite of the fact that the algorithm contains some programming inefficiencies, it still performs well in terms of running speed and result quality. Zomaya and Teh (2001) investigate how a genetic algorithm can be employed to solve the dynamic load balancing problem. The dynamic load-balancing algorithm is developed by Zomaya and Teh (2001) whereby optimal or near-optimal task allocations can "evolve" during the operation of the parallel computing system. A scheduling routine based upon a genetic algorithm is developed (Greene, 2001) which is claimed to be very effective and has relatively low cost. Two important aspects of this research study are: loads on the processors are well balanced and scheduling per se remains cheap in comparison to the actual productive work of the processors. Dynamic Distributed Genetic Algorithm is proposed by Yi et al (2000). According to the paper, dynamic distributed GA with directed migration has great potential to overcome premature convergence.

The contribution of Song et al (2005) is two-fold: first the Min-Min and Sufferage heuristics are enhanced under three risk modes driven by security concerns and secondly a new Space-Time Genetic Algorithm for trusted job scheduling is proposed. The results of Song et al (2005) shows that there is a need of more research study in order to over come the shortcoming of security driven Min-Min and Sufferage heuristics which are unstable when applied to different types of workloads. A novel GA-based approach is proposed by Kim and Weissman (2004) to address the problem of scheduling a divisible Data Grid application while considering communication and computation at the same time in wide area data intensive environment. According to Kim and Weissman (2004) the results from the experiments on GA-related parameters suggest that the initialization of population with chromosomes of good quality is critical to GA-based approach in terms of the quality of solution and the convergence rate. But the problem of multiple jobs competing for shared resources hasn't been overcome in this study. Five heuristics that have been designed, developed, and simulated using the HC environment, are presented by Shivle et al (2005). Application tasks are composed of communicating subtasks with data dependencies, and multiple versions were mapped using the heuristics described by Shivle et al (2005) and the results can be used in the development of ad hoc grids. The EVOLVE/G system, which is a Grid tool for developer of evolutionary computation, is proposed by Tanimura et al (2002). This system consists of an Agent and multiple Workers. Since the data can be exchanged between the Agent and Workers freely, any logical models of EC can be integrated. Jing et al (2004) describes a parallel hybrid-GA (PHGA) for combinatorial optimization using an island model running in a networked computing environment. Jing et al (2004) opens several issues for future research like extensive study on scalability of parallel GA in a distributed computing framework. Applicability of parallelizing the local search of a serial GA and other heuristics for local search can be explored to enhance the performance of the parallel GA. Cao et al (2003) developed a GA-based scheduler for fine-grained load balancing at the local level, which was then coupled with an agent-based mechanism that was applied to load balance at a higher level. Future enhancement to the system will include the integration with other grid toolkits (e.g. Globus MDS and NWS).

MATERIALS AND METHODS

In this research work an attempt has been made to increase the efficiency of grid scheduler. GAs based scheduling algorithm named "Dynamic Online Scheduling" is proposed for better resource optimization and task scheduling. The scheduling process in this algorithm is addressed in two layers namely pre-scheduling and post-scheduling. The newly coming problems from outside grid boundary are scheduled in the first layer which is pre-scheduling. In post-scheduling the load balancing of the already submitted tasks is done, that is if a certain resource is found overloaded with work while some other resources are free then some of the jobs of the overloaded machine are automatically shifted to the free machines while keeping in mind the robustness, reliability and efficiency of the job as well as the time cost, communication cost. Resource cost will also be considered.

Load and Time Prediction Technique:

The load on the resources and the execution time of the tasks both are interrelated and depend on each other. Every task has certain execution time and every task puts a certain amount of load on the machine it is executed on. We have developed a mechanism which on the basis of task attributes (e.g size, type etc) and resource attributes (e.g. memory, cpu cycles, load) tells the task execution time on that machine. This strategy gives us the advantage of dynamicity in the pool of heterogeneous resources as well as tasks, since both task and resource attributes are not assumed to be fixed. The load and time prediction strategy is based on historical data. Each resource is continuously monitored for its utilization and the data is entered into the log file. This log file helps us to set the threshold for our resources at any particular hour of day. The resource threshold is updated after each hour. At the same time an extensive amount tasks are executed on each resource and the resource utilization is calculated with each task and the log file logs that task's attributes and resource utilization. After that, we rearrange the task log file according to the task attributes and time of the day they were submitted on. Then we give this data to the genetic algorithms to learn which resource is best suited for which kind of task at any particular given time of the day.

So far time prediction seems enough for load balancing since each task will be assigned on the basis of threshold plus the current situation of resource. But that's not exactly what happens in the real world. When multiple tasks come the scheduler calculates the task execution time for each task with respect to the current situation of the available resources it is assigned to. Each task will definitely increase some amount of load on a resource. If multiple tasks are assigned to a resource then after one task starts running, the current situation of the resource will change but the scheduler predicts the task execution time according to the pervious resource situation that is before the execution of the previous task began. To cover up this flaw, we developed a Load Prediction strategy. This strategy is also based on history data but this data is required to be generated only once for each different set of attributes of task.

At this point it all seems static at compile time work while its not. Dynamicity comes when a new task arrives, GA scheduler become is activated and it retrieves the required task attributes, lists the currently available resources and predicts the execution time of the task for each resource on the basis of current resource parameters (e.g. memory, cpu cycles, load) or predicted parameters, task attributes and the history data from log file (that how much time the task of given attributes takes to execute with respect to the resource history parameters). The formula used by GAs to calculate the prediction time is explained in a later section.

Dynamic Online Scheduling: In dynamic online scheduling, scheduling is done on the basis of the current situation of the grid, which takes the current resources states / parameters from the resource collector and task list to be scheduled from the task collector and provides both lists to GA based scheduler as shown in figure 1. The Dynamic Online Scheduling procedure works in the following manner.

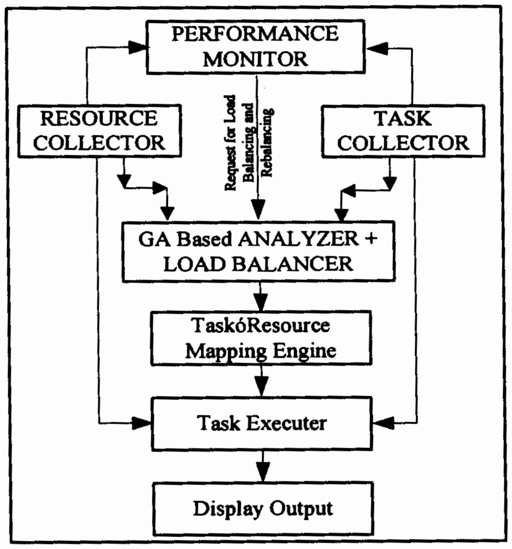

PERFORMANCE MONITOR
RESOURCE COLLECTOR
Request for Load Balancing and Rebalancing
TASK COLLECTOR
GA Based ANALYZER + LOAD BALANCER
Task6Resource Mapping Engine
Task Executer
Display Output

Figure 1: Dynamic Online Scheduling Overview

Pre-Scheduling

1. A list of available resources is generated on the basis of the current situation of resources as well as the history data about resources.
2. A list of tasks to be scheduled is generated from the task queue.
3. Both lists (Resource list and Task list) are provided to the GA based Analyzer and Load Balancer which will generate the optimized mapping of tasks to resources.
4. Mapping Engine assigns the tasks to resources.
5. Task Executer executes the task and displays the output.

Post-Scheduling

1. Post scheduling algorithm becomes active when the performance monitor views that certain resources are being over utilized while some others are being under utilized. The criteria for the measurement of the resource's overload (that is if whether a certain resource is overloaded or not) is its threshold limit.
2. Each resource has been assigned a certain threshold on the basis of history data.
3. If certain resources are overloaded then the performance monitor comes into action. It will place the request to Load Balancer for redistribution of the tasks.
4. Load Balancer will activate the Resource Collector to provide the list of overloaded resources and under utilized resources, and Task Collector to provide the Task list of overloaded resources.
5. Load Balancer will then generate the new mapping.

Block diagram for pre and post scheduling has provided the detailed internal architecture of system as shown in figure 2.

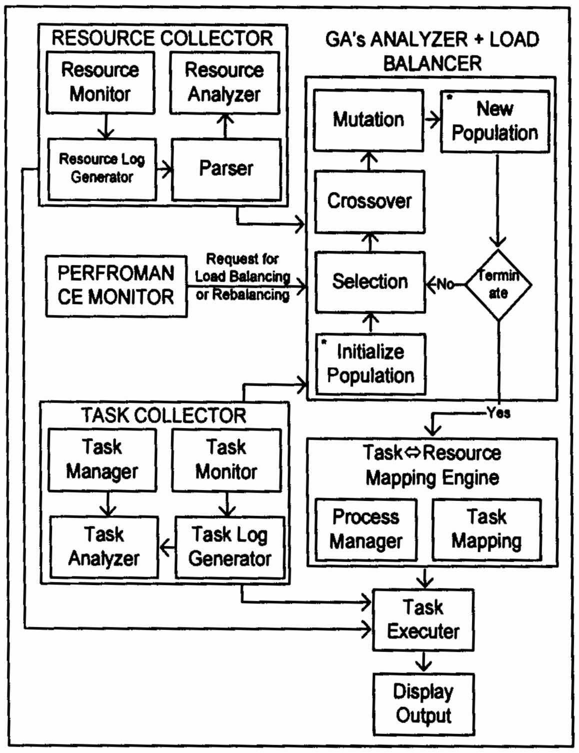


Figure 2: Detailed Internal Architecture of Online Scheduling

Genetic Algorithms for Scheduling

Chromosome Representation: For applying GAs directly or coupled with other meta-heuristics, problem (chromosome) representation is very important and it directly affects the performance of the proposed algorithm. The first decision a designer has to make is how to represent a solution in a chromosome [14]. We assume that the Tasks and Resources are arranged in an ascending order according to the Task attributes (that is size, type, submission time) and Resource usage (that is least loaded comes first). Figure 3 depicts the chromosome representation which is used in our current strategy. Basically each chromosome in the population contains permutation of tasks and their fitness is calculated according to the resources they were assigned to. Task T_1 is allocated to resource R1, T_2 to R2 and T_3 to R3 and so on. When T_1 is completed, resource R1 is empty and task T_N is allocated. This procedure goes on until all the tasks are allocated.

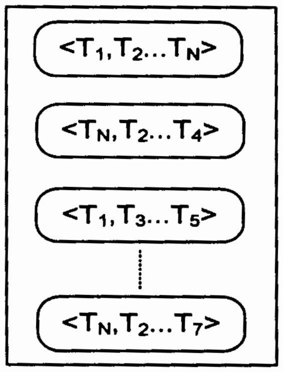


Figure 3: Chromosome Representation

GA Approach for Task Scheduling and Load Balancing

1. Get the Task List from the Task Collector of length TNT where TNT is the total number of tasks to be scheduled.
2. Get the Resource List from the Resource Collector of length TNAR where TNAR is the total number of available resources, if no resource is available then wait until resources become available.
3. At t =0; generate an initial population with *P* chromosomes *Popi(t)*, where *P* represents the permutations of tasks and is calculated as:

$$P = {}_{TNT}P_{TNT}$$

4. For each chromosome (i=1 to P), first allocate the jobs to the available resources based on the FCFS basis. Then calculate the predicted time for each task according to the current parameters of the resource it is assigned to from the task history log. For example if one parameter of a resource say load is used then the task time prediction formula will be as follows

$$\frac{PT_{T,R}}{PL_R} = \frac{HT_{T,R}}{HL_R}$$

$$PT_{T,R} = PL_R \frac{HT_{T,R}}{HL_R}$$

Where:
$PT_{T,R}$ = Predicted Completion Time of task T on resource R
PL_R = Present Load on resource R for 1st task, and Predicted Load for next assigning tasks (Each task will increase some load on the resource which will be calculated as predicted present load for next task)
$HT_{T,R}$ = History Completion Time for task T on resource R (taken from Task Log History).
HL_R = History Load on resource R when task T had completion time $HT_{T,R}$.

Now the predicted time for each task in the chromosome can be easily calculated using afore mentioned formula.

5. Fitness value for each chromosome is calculated which will tell us the make-span of the schedule. Fitness value is calculated using following formula:

$$F = \frac{1}{Max(\sum_{i=1}^{TNT} PT_{Ti,R})}$$

6. The make-span of the schedule is calculated using following formula:

$$\acute{\phi} = Max(\sum_{i=1}^{TNT} PT_{Ti,R})$$

Where $\sum_{i=1}^{TNT} PT_{Ti,R}$ represents the total number of tasks assigned to resource R.

7. Apply Crossover operator on the population according to the probability selected, Crospop(t+1) = recombined chromosomes of the population *NewPop*$_i$*(t+1)*;

8. Apply mutation operator on the population according to the probability selected, *MutPop(t+1)* = mutated population *CrosPop(t+1)*.

9. Evaluate the fitness of each chromosome in the new population and check if the specified fitness value is achieved or not. If not, start the GAs loop again.

10. Send the specified schedule generated by GAs to the Mapping Engine which will assign the tasks to the resources. In case of post scheduling the Mapping Engine will migrate the tasks from the overloaded resources to relatively less loaded resources.

EXPERIMENTAL RESULTS AND DISCUSSION

As a fundamental base, we have adopted OpenMosix as the resource monitoring and management tool. OpenMosix provides the update network weather service as well as logging the utilization information of each resource. A Task Management tool is developed which receives task from the users. GA scheduler works at the back end which takes tasks form task management tool and maps them to the available resources. (List of available resources is provided by resource management tool i.e OpenMosix) First we run bunches of tasks on limited resources (2-8 machines) without any involvement of GA scheduler and calculate the execution cost that is the total time it takes to complete all the assigned tasks on all machines, and then we perform the same test using GA based scheduler and again calculate the execution cost. We calculate the execution cost for different sizes of schedules by slowly increasing the number of resources (5, 10, 15, 20, 25 and 30 number of machines) both with and without GA and the results shows that performance of system increases when GA base scheduler is used. Figure 4 shows the comparison results of system with proposed GA scheduler and without GA scheduler. Schedule Size vs. Make span is plotted in figure 4.

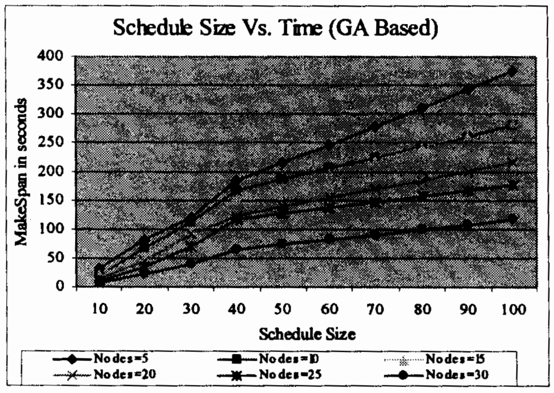


Figure 4-(a): System performance with GAs based scheduler (Schedule Size Vs. Time)

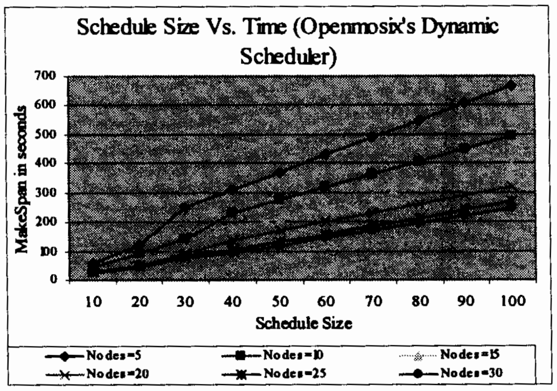


Figure 4-(b): System performance without GAs based scheduler (Schedule Size Vs. Time)

Table 1 and 2 displays the detailed information of execution time of different sizes of schedules with respect to different number of resources generated by GA scheduler and Openmosix scheduler respectively. Table 3 indicates the performance gain of GA scheduler over Openmosix scheduler for each set of schedules and resources.

Table 1: Makespan time in seconds generated by GA on different sizes of schedule and number of processors

GENETIC ALGORITHM BASED DYNAMIC ONLINE SCHEDULER RESULTS						
S.Size	No of Processors					
	5	10	15	20	25	30
10	31.8	21.8	18.9	15.6	10.2	8.5
20	81.1	66.7	53.4	44.7	33.8	23.3
30	119.7	111.0	96.3	90.9	69.9	40.2
40	183.4	167.4	154.8	122.2	115.5	64.9
50	215.2	189.3	173.7	137.9	125.7	73.5
60	247.1	211.1	192.6	153.6	136.0	82.0
70	279.0	233.0	211.5	169.2	146.2	90.6
80	310.9	254.9	230.5	184.9	156.4	99.1
90	342.8	276.8	249.4	200.6	166.6	107.6
100	374.7	298.7	268.3	216.3	176.8	116.2

Table 2: Make span time in seconds generated by openmosix on different sets of schedule size and number of processors

OPENMOSIX'S DYNAMIC SCHEDULER RESULTS						
S.Size	No of Processors					
	5	10	15	20	25	30
10	59	43.8	36.5	30	27.5	25.5
20	121	94.5	57.6	50	48	45
30	251.5	141	119.5	92.4	82.5	77
40	310.5	232.5	174.5	139	102.5	93.6
50	369.5	276.3	211.0	169	130	119.1
60	428.5	320.1	247.5	199	157.5	144.6
70	487.5	363.9	284.0	229	185	170.1
80	546.5	407.7	320.5	259	212.5	195.6
90	605.5	451.5	357.0	289	240	221.1
100	664.5	495.3	393.5	319	267.5	246.6

Table 3: Performance gain of Genetic algorithms based scheduler over openmosix scheduler

PERFORMANCE GAIN OF GA OVER OPENMOSIX					
S.Size	No of Processors				
	10	15	20	25	30
10	49.96%	51.83%	52.27%	37.13%	33.47%
20	70.67%	92.75%	89.50%	70.59%	51.92%
30	78.75%	80.63%	98.44%	84.83%	26.27%
40	72.01%	88.74%	87.95%	112.76%	69.44%
50	68.52%	82.35%	81.62%	96.76%	61.74%
60	65.98%	77.85%	77.19%	86.35%	56.75%
70	64.05%	74.51%	73.93%	79.04%	53.26%
80	62.53%	71.92%	71.42%	73.61%	50.68%
90	61.31%	69.87%	69.43%	69.43%	48.70%
100	60.31%	68.20%	67.82%	66.11%	47.12%

In order to visualize the performance of system more clearly root mean square root error and correlation coefficient is calculated from the results generated by Openmosix scheduler and GA scheduler. Figure 5 shows the correlation coefficient series, for different sizes of schedules and number of resources, of both GA scheduler and Openmosix Scheduler and it can be clearly seen that GA scheduler is more correlated when we increase the schedule size over time than Openmosix scheduler.

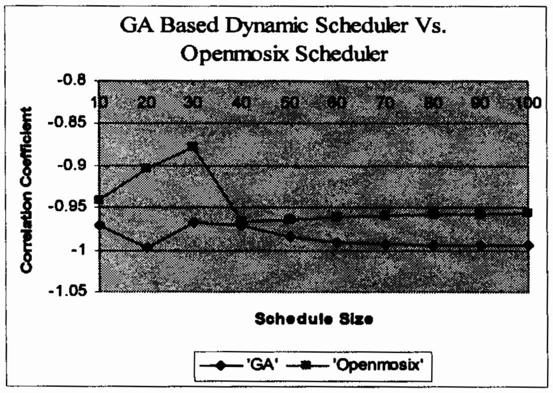


Figure 5: Correlation Coefficient of GA scheduler and Openmosix scheduler

Figure 6 displays the root mean square root error of Openmosix scheduler with respect to GA scheduler and the plot indicates that as the schedule size increases the error generated by Openmosix also increases.

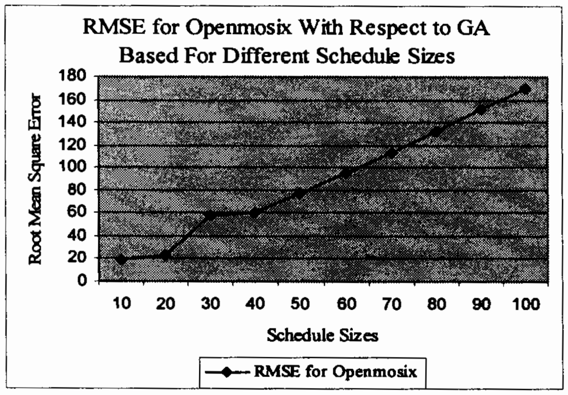


Figure 6: Root Mean Square Error for Openmosix scheduler

Finer results are obtained when GAs is executed for another 1000-1500 generations but by increasing the number of generations performance gain due to GAs starts compromising in terms of scheduling cost so in order to obtain the best optimal result number of generations for GAs are set to fixed for 500. After running for 1500 generations the results appear more or less the same. Table 4 shows the improvement in makespan ó (schedule cost) after each 100 generations for different sizes of schedules.

Table 4: Improvement in the Makespan (Schedule Cost) after each 100 generations for different schedule sizes

IMPROVEMENT IN THE MAKESPAN (SCHEDULE COST) AFTER EACH 100 GENERATIONS				
#. Of Generations	Schedule Size			
	70	80	90	100
100	427.1055	437.8263	484.9545	491.3856
200	323.7534	356.622	327.1157	374.9545
300	219.8443	234.5184	225.4005	280.4443
400	174.4709	128.2745	124.9915	191.7306
500	90.6014	99.1374	107.6734	116.2094
600	90.6014	98.1301	106.5564	115.5494
700	89.0151	98.1301	106.5564	115.5494
800	89.0151	97.1404	106.5564	115.5494
900	88.0151	97.1404	106.0112	115.0013
1000	88.0151	97.1404	105.5461	115.0013
1100	87.6014	96.0011	105.5461	114.5095
1200	87.6014	96.0011	105.5461	114.5095
1300	86.0121	96.0011	105.5461	114.5095
1400	86.0121	96.0011	105.0215	114.1421
1500	86.0121	96.0011	105.0215	114.1421
1600	85.0501	96.0011	105.0215	114.1421
1700	85.0501	96.0011	105.0215	114.1421
1800	85.0501	96.0011	105.0215	114.1421
1900	85.0501	96.0011	105.0215	114.1421
2000	85.0501	96.0011	105.0215	114.1421

CONCLUSION AND FUTURE WORK

In this research work we attempt to address the problem of load balancing in grid computing using one of the popular heuristics namely GAs. Genetic Algorithm predicts the execution time for each task with respect to the resource it is assigned to. The prediction time is based on the current attributes of task, current and historical parameters (like load, memory etc) of resources. We have tested this algorithm on a 30 machines heterogeneous grid environment with different schedule sizes and the results show that the proposed strategy can lead to the significant performance gain.

Future work will examine the application of proposed GA based algorithm as parallel genetic algorithms and dynamic distributed algorithms proposed by Yi et al (2000). In both cases number and size of populations must be carefully determined. Even though significant progress has been made in modelling the infrastructure of grid computing but a close review clearly indicates that not much progress is made in formulating the efficient and globally optimized, grid-scheduling algorithm for allocating jobs (Abraham et al. 2000). Therefore we are planning to investigate this research area in depth using GAs as well as other heuristic algorithms.

REFERENCES

A. Abraham, R. Buyya and B. Nath, 2000. Nature's Heuristics for Scheduling Jobs on Computational Grids. The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), December 14-16, 2000. Cochin, India.

A. Y. Zomaya and Y.H, 2001. The Observations on Using Genetic Algorithms for Dynamic Load Balancing. IEEE Transactions on Parallel and Distributed Systems, Vol. 12 (9), IEEE Press.

Burhaneddin SANDIKCI, 2000. Genetic Algorithms. *http://www.ie.bilkent.edu.tr/~lors/ie572/burhaneddin.pdf*

Cheng, C.T., Zhao, M.Y., Chau, K.W. and Wu, X.Y. 2006. Using genetic algorithm and TOPSIS for Xinanjiang model calibration with a single procedure. Journal of Hydrology, Vol. 316, No. 1-4, pp. 129-140.

Cheng, C.T., Wu, X.Y. and Chau, K.W. 2005. Multiple criteria rainfall-runoff model calibration using a parallel genetic algorithm in a cluster of computer. Hydrological Sciences Journal, Vol. 50, No. 6, 2005, pp. 1069-1087.

Chau, K.W. 2004. A two-stage dynamic model on allocation of construction facilities with genetic algorithm. Automation in Construction, Vol. 13, No. 4, 2004, pp. 481-490

Chau, K.W. and Albermani, F. 2003. Knowledge-based system on optimum design of liquid retaining structures with genetic algorithms. Journal of Structural Engineering, ASCE, Vol. 129, No. 10, 2003, pp. 1312-1321.

Devid E. Goldberg (1989). Genetic Algorithms in search optimization and Machine Learning. Pearson Education Pte. Ltd., Singapore.

D.P. Spooner, S.A. Jarvis, J.Cao, S. Saini and G.R.Nudd, 2003. Local Grid Scheduling Techniques using Performance Prediction. IEE Proceedings Computers and Digital. Techniques, Vol. 150(2). April 2003, Institution of Electrical Engineers, Great Britain.

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

http://www.dcs.warwick.ac.uk

http://cs.felk.cvut.cz/~xobitko/ga/about.html

Ian Foster, Carl Kesselman and Steven Tuecke, 2001. The Anatomy of the Grid Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, Volume 15, Issue 3, Sage Publications, Inc. Thousand Oaks, CA, USA.

John R. Koza (1990). Genetic programming: a paradigm for genetically Breeding populations of computer programs to Solve problems. Stanford University Computer Science Department technical report STAN-CS-90-1314.

Jean-Philippe Rennard, 2000. Genetic Algorithm Viewer: Demonstration of a Genetic Algorithm. http://www.rennard.org/alife

J. Cao, D. P. Spooner, S.A. Jarvis and G.R. Nudd, 2005. Grid Load Balancing Using Intelligent Agents. Future Generation Computer Systems, Vol 21(1). Elsevier Science Publishers B.V. Amsterdam, The Netherlands.

J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini and G. R. Nudd, 2003. Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling. Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IEEE Computer Society, 2003. Washington DC, USA.

Mitchell Melanie (1999). An Introduction to Genetic Algorithms. A Bradford Book, The MIT Press, Cambridge, Massachusetts, London, England.

Michael D. Vose (1999). The Simple Genetic Algorithm, Foundation and Theory. A Bradford Book, The MIT Press. Cambridge, Massachusetts. London, England.

M. Wieczorek, R. Prodan and T. Fahringer, 2005. Scheduling of Scientific Workflows in the ASKALON Grid Environment. ACM SIGMOD Record, Vol. 34 (3). ACM Press. New York, USA.

M. Grajcar, 2000. Conditional Scheduling for Embedded Systems Using Genetic List Scheduling. Proceedings of the 13th international symposium on System synthesis, 2000. Washington DC, USA. IEEE Computer Society.

R. A. Moreno, 2003. Job Scheduling and Resource Management Techniques in Dynamic Grid Environments. In 1st European Across Grids Conference, 2003. Heidelberg, Germany.

Surendra Reddy, 2004. Grid Computing: Crossing the Chasm, 24th September 2004. http://sciencecareers.sciencemag.org/carrer_development/previous_issues/articles/grid_computing_crossing_the_chasm/(parent)/12093

S. Sanyal and S. K. Das, 2005. MaTCH: Mapping Data-Parallel Tasks on a Heterogeneous Computing Platforms Using the Cross-Entropy Heuristic. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005. Washington DC, USA. IEEE Computer Society.

S. Wagner, M. Affenzeller, 2004. Heuristic lab Grid – A Flexible And Extensible Environment For Parallel Heuristic Optimization. Proceedings of the 15th International Conference on Systems Science, Vol. 1, 2004. Oficyna Wydawnicza Politechniki Wroclawskiej.

S. Song, Y. K. Kwok and K. Hwang, 2005. Security-Driven heuristics and A FAST Genetic Algorithm for Trusted Grid Job Scheduling. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005. Washington DC, USA. IEEE Computer Society.

infrastructure of grid computing but a close review clearly indicates that not much progress is made in formulating the efficient and globally optimized, grid-scheduling algorithm for allocating jobs (Abraham et al. 2000). Therefore we are planning to investigate this research area in depth using GAs as well as other heuristic algorithms.

REFERENCES

A. Abraham, R. Buyya and B. Nath, 2000. Nature's Heuristics for Scheduling Jobs on Computational Grids. The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), December 14-16, 2000. Cochin, India.

A. Y. Zomaya and Y.H, 2001. The Observations on Using Genetic Algorithms for Dynamic Load Balancing. IEEE Transactions on Parallel and Distributed Systems, Vol. 12 (9), IEEE Press.

Burhaneddin SANDIKCI, 2000. Genetic Algorithms. *http://www.ie.bilkent.edu.tr/~lors/ie572/burhaneddin.pdf*

Cheng, C.T., Zhao, M.Y., Chau, K.W. and Wu, X.Y. 2006. Using genetic algorithm and TOPSIS for Xinanjiang model calibration with a single procedure. Journal of Hydrology, Vol. 316, No. 1-4, pp. 129-140.

Cheng, C.T., Wu, X.Y. and Chau, K.W. 2005. Multiple criteria rainfall-runoff model calibration using a parallel genetic algorithm in a cluster of computer. Hydrological Sciences Journal, Vol. 50, No. 6, 2005, pp. 1069-1087.

Chau, K.W. 2004. A two-stage dynamic model on allocation of construction facilities with genetic algorithm. Automation in Construction, Vol. 13, No. 4, 2004, pp. 481-490

Chau, K.W. and Albermani, F. 2003. Knowledge-based system on optimum design of liquid retaining structures with genetic algorithms. Journal of Structural Engineering, ASCE, Vol. 129, No. 10, 2003, pp. 1312-1321.

Devid E. Goldberg (1989). Genetic Algorithms in search optimization and Machine Learning. Pearson Education Pte. Ltd., Singapore.

D.P. Spooner, S.A. Jarvis, J.Cao, S. Saini and G.R.Nudd, 2003. Local Grid Scheduling Techniques using Performance Prediction. IEE Proceedings Computers and Digital. Techniques, Vol. 150(2). April 2003, Institution of Electrical Engineers, Great Britain.

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

http://www.dcs.warwick.ac.uk

http://cs.felk.cvut.cz/~xobitko/ga/about.html

Ian Foster, Carl Kesselman and Steven Tuecke, 2001. The Anatomy of the Grid Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, Volume 15, Issue 3, Sage Publications, Inc. Thousand Oaks, CA, USA.

John R. Koza (1990). Genetic programming: a paradigm for genetically Breeding populations of computer programs to Solve problems. Stanford University Computer Science Department technical report STAN-CS-90-1314.

Jean-Philippe Rennard, 2000. Genetic Algorithm Viewer: Demonstration of a Genetic Algorithm. http://www.rennard.org/alife

J. Cao, D. P. Spooner, S.A. Jarvis and G.R. Nudd, 2005. Grid Load Balancing Using Intelligent Agents. Future Generation Computer Systems, Vol 21(1). Elsevier Science Publishers B.V. Amsterdam, The Netherlands.

J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini and G. R. Nudd, 2003. Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling. Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IEEE Computer Society, 2003. Washington DC, USA.

Mitchell Melanie (1999). An Introduction to Genetic Algorithms. A Bradford Book, The MIT Press, Cambridge, Massachusetts, London, England.

Michael D. Vose (1999). The Simple Genetic Algorithm, Foundation and Theory. A Bradford Book, The MIT Press. Cambridge, Massachusetts. London, England.

M. Wieczorek, R. Prodan and T. Fahringer, 2005. Scheduling of Scientific Workflows in the ASKALON Grid Environment. ACM SIGMOD Record, Vol. 34 (3). ACM Press. New York, USA.

M. Grajcar, 2000. Conditional Scheduling for Embedded Systems Using Genetic List Scheduling. Proceedings of the 13th international symposium on System synthesis, 2000. Washington DC, USA. IEEE Computer Society.

R. A. Moreno, 2003. Job Scheduling and Resource Management Techniques in Dynamic Grid Environments. In 1st European Across Grids Conference, 2003. Heidelberg, Germany.

Surendra Reddy, 2004. Grid Computing: Crossing the Chasm, 24th September 2004. http://sciencecareers.sciencemag.org/carrer_development/previous_issues/articles/grid_computing_crossing_the_chasm/(parent)/12093

S. Sanyal and S. K. Das, 2005. MaTCH: Mapping Data-Parallel Tasks on a Heterogeneous Computing Platforms Using the Cross-Entropy Heuristic. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005. Washington DC, USA. IEEE Computer Society.

S. Wagner, M. Affenzeller, 2004. Heuristic lab Grid – A Flexible And Extensible Environment For Parallel Heuristic Optimization. Proceedings of the 15th International Conference on Systems Science, Vol. 1, 2004. Oficyna Wydawnicza Politechniki Wroclawskiej.

S. Song, Y. K. Kwok and K. Hwang, 2005. Security-Driven heuristics and A FAST Genetic Algorithm for Trusted Grid Job Scheduling. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005. Washington DC, USA. IEEE Computer Society.

S. Kim and J. B. Weissman, 2004. A GA-based Approach for Scheduling Decomposable Data Grid Applications. Proceedings of the International Conference on Parallel Processing (ICPP'04) – Vol. 00, 2004. Washington DC, USA. IEEE Computer Society.

S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam and J. Velazco. Mapping of Subtasks with Multiple Versions in a Heterogeneous Ad Hoc Grid Environment, 2005. Parallel Computing, Vol. 31(7), Elsevier Science Publishers B. V. Amsterdam, The Netherlands.

T. Jing, M. H. Lim and Y. S. Ong, 2004. Island Model Parallel Hybrid-GA for Large Scale Combinatorial Optimization. The Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV2004), Special Session on Computational Intelligence on the Grid, December 6-9, 2004. Kunming, China.

W. A. Greene, 2001. Dynamic Load Balancing via a Genetic Algorithm. Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01), 2001. Washington DC, USA. IEEE Computer Society.

Wu, C.L. and Chau K.W. 2006. A flood forecasting neural network model with genetic algorithm. International Journal of Environment and Pollution, Vol. 28, No. 3-4, pp. 261-273

W. Yi, Q. Liu, Y. He, 2000. Dynamic Distributed Genetic Algorithms. Proceedings of the 2000 Congress on Evolutionary Computation CEC00, 2000. California, USA. IEEE Press.

Y. Tanimura, T. Hiroyasu, M. Miki and K. Aoi, 2002. The System for Evolutionary Computing on the Computational Grid. Proceedings of Parallel and Distributed Computing and Systems, 2002. Cambridge, USA. ACTA Press.

APPENDIX C

USER MANUAL

C – USER MANUAL

Following is the description of software that is being used to compare the results. The screens and their descriptions will be useful to understand this software.

.

C.1 TASK COLLECTOR SCREENS

Task collector is the main GUI which interacts with user to receive new tasks from user. Task collector screens are shown in figures 8-1 to 8-5.

C.1.1 TASK COLLECTOR MAIN SCREEN

Task Collector is executed to start receiving the tasks from user and the screen shown in Fig 8-1 is displayed for this purpose.

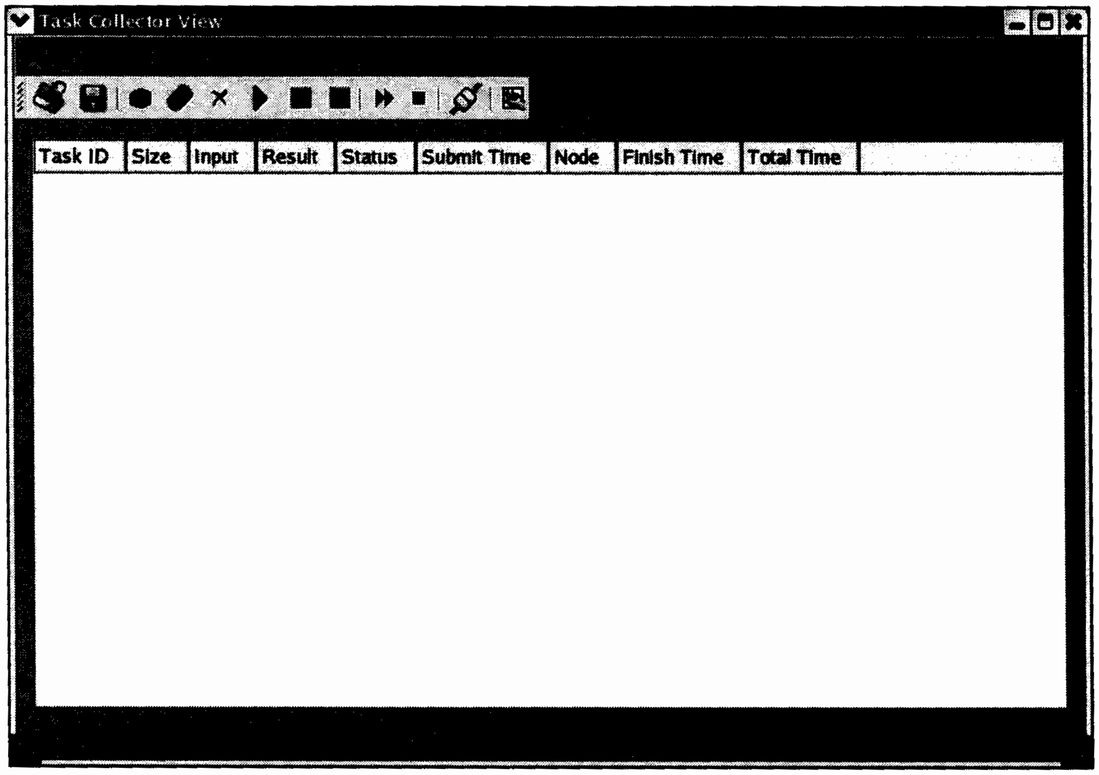
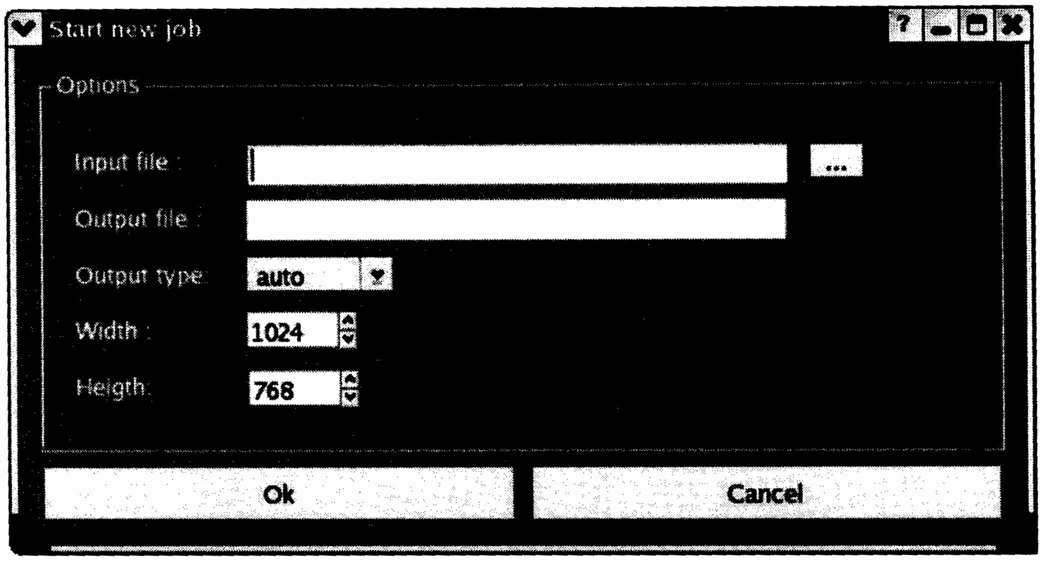


Fig C-1: Main Screen of Task Collector

C.1.2 SUBMIT TASK SCREEN

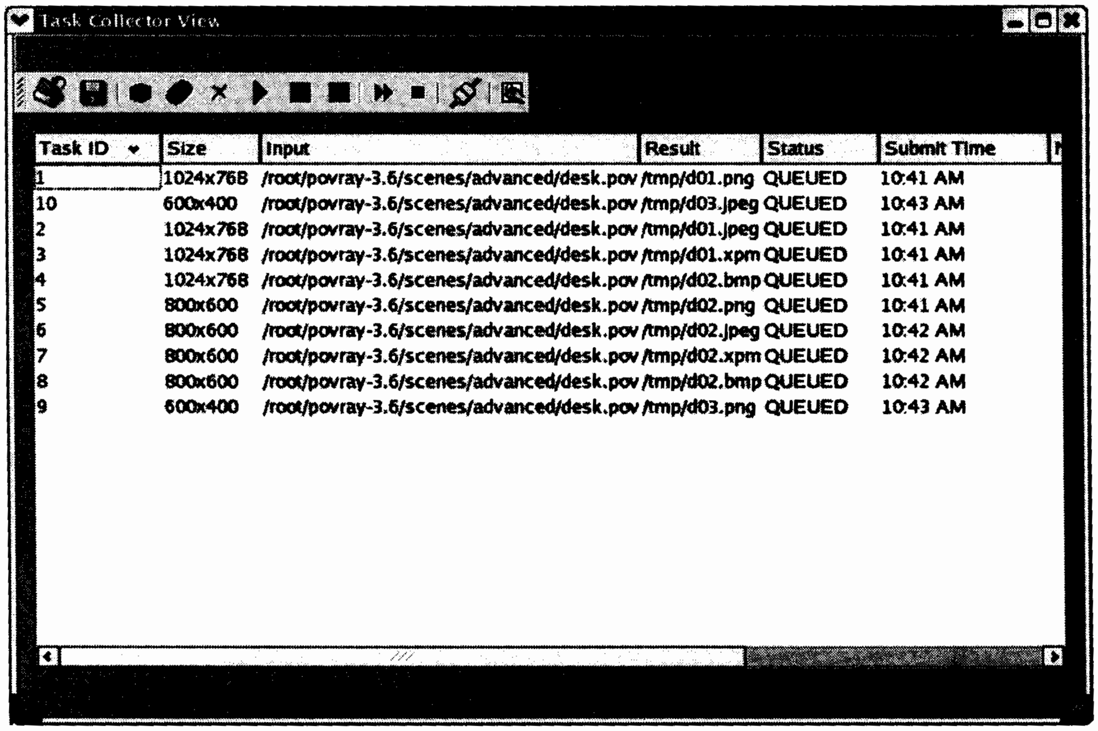
Task information will be received from the user in the following screen as displayed in figure 8-2.



Input file :	
Output file :	
Output type:	auto
Width :	1024
Heigth:	768

Fig C-2: Submit Task

After the task is submitted the main screen shown in figure 8-1 will display the information about the tasks as shown in the figure 8-3.



Task ID	Size	Input	Result	Status	Submit Time
1	1024x768	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d01.png	QUEUED	10:41 AM
10	600x400	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d03.jpeg	QUEUED	10:43 AM
2	1024x768	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d01.jpeg	QUEUED	10:41 AM
3	1024x768	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d01.xpm	QUEUED	10:41 AM
4	1024x768	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d02.bmp	QUEUED	10:41 AM
5	800x600	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d02.png	QUEUED	10:41 AM
6	800x600	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d02.jpeg	QUEUED	10:42 AM
7	800x600	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d02.xpm	QUEUED	10:42 AM
8	800x600	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d02.bmp	QUEUED	10:42 AM
9	600x400	/root/povray-3.6/scenes/advanced/desk.pov	/tmp/d03.png	QUEUED	10:43 AM

Fig C-3: After task submission

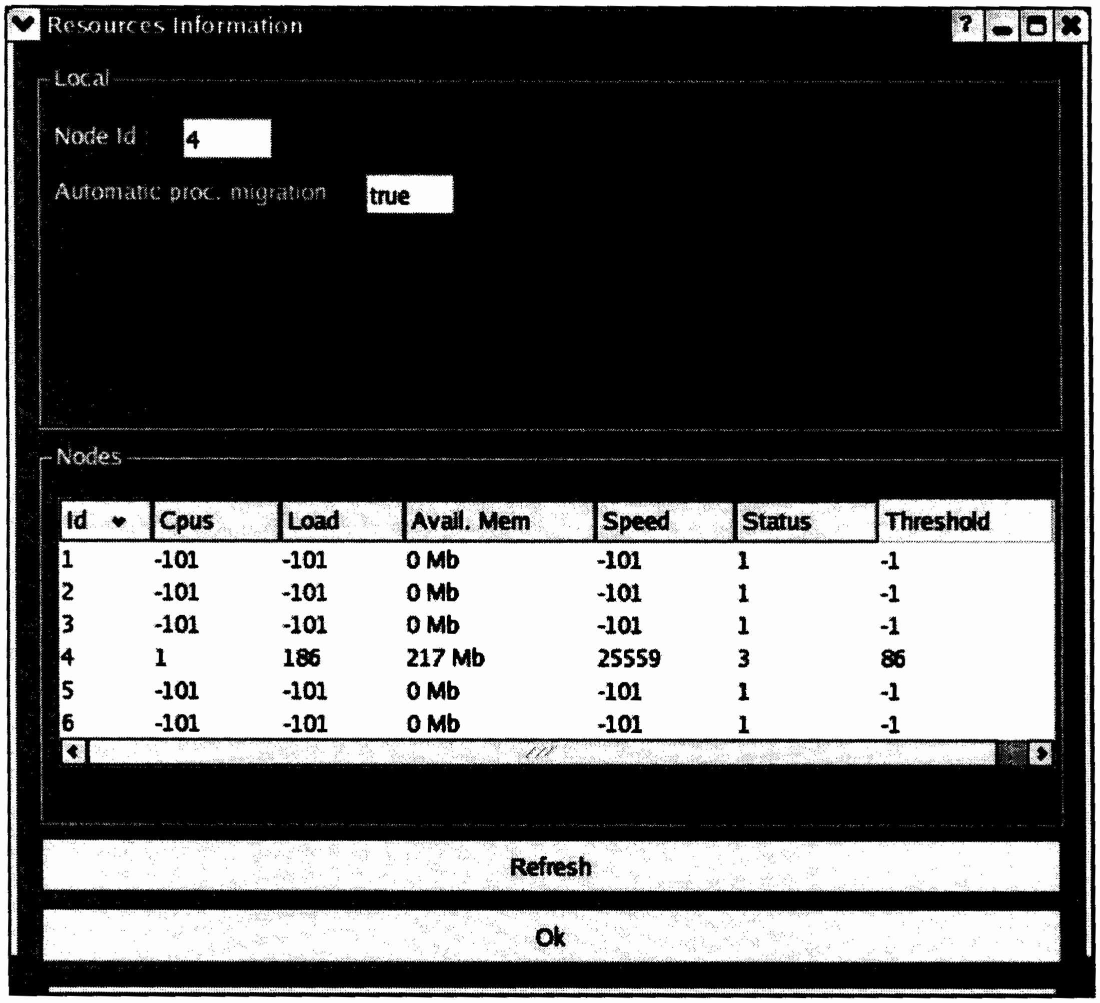
C.1.3 TASK OUTPUT SCREEN

When the task is finished the view output button is enabled and task output is displayed as shown in the figure 8-4 according to the size provided by the user at the time of task submission.

Fig C-4: Task Output Screen

C.1.4 RESOURCE INFORMATION SCREEN

For task collector usage resource information is also kept within the task collector module. Resource information e.g. its status, memory usage, load and threshold etc is displayed in the following screen shown in figure 8-5.



Node Id: 4

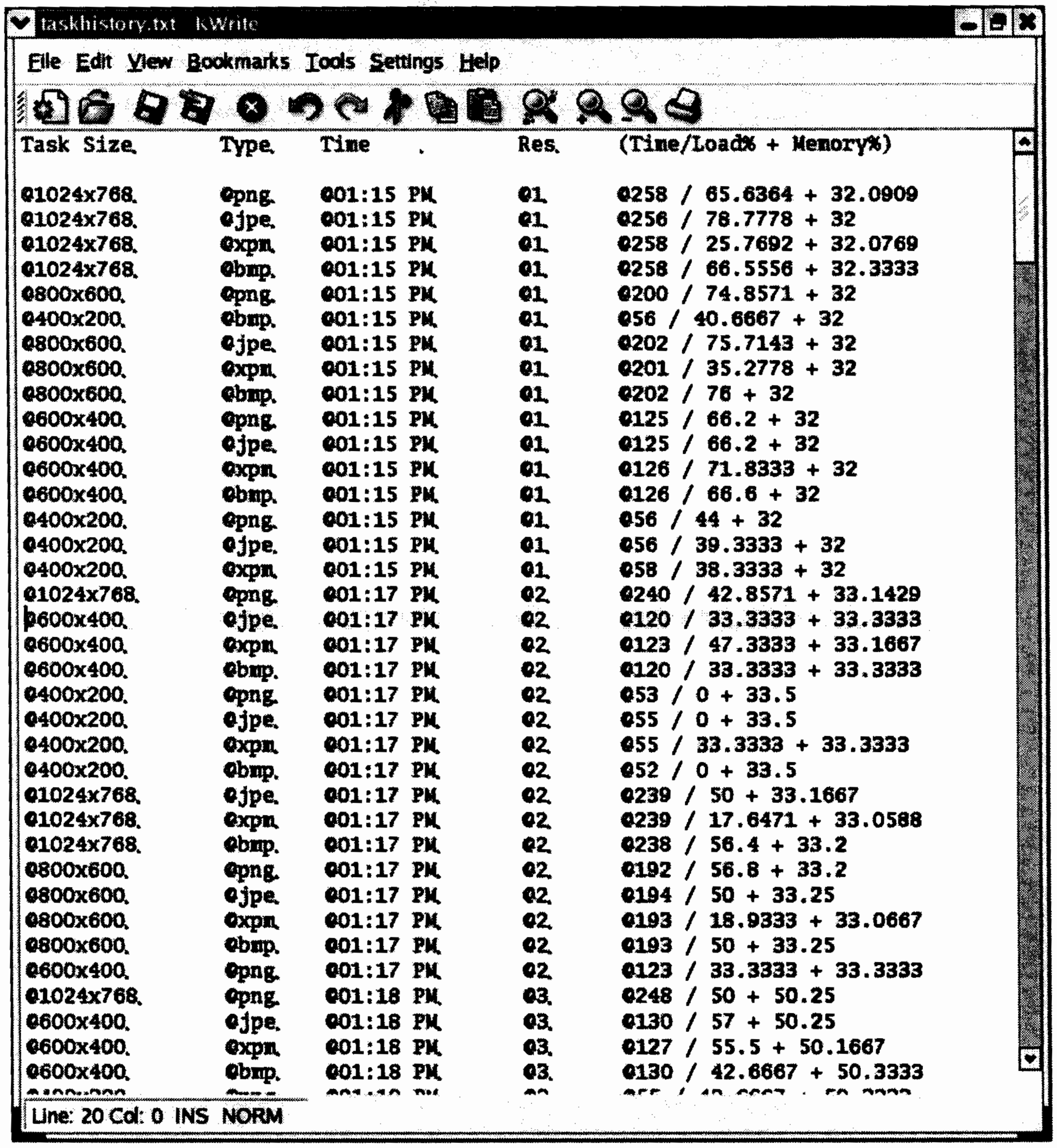
Automatic proc. migration: true

Id	Cpus	Load	Avail. Mem	Speed	Status	Threshold
1	-101	-101	0 Mb	-101	1	-1
2	-101	-101	0 Mb	-101	1	-1
3	-101	-101	0 Mb	-101	1	-1
4	1	186	217 Mb	25559	3	86
5	-101	-101	0 Mb	-101	1	-1
6	-101	-101	0 Mb	-101	1	-1

Fig C-5: Resource Information Screen

C.1.5 TASK LOG FILES

Two log files are generated and used by task collector, which are keeping the information of total task execution time with respect to resource it is assigned to. These log files are later on used by genetic algorithms to make scheduling decisions. The preview of two log files is shown in figures 8-6 and 8-7.

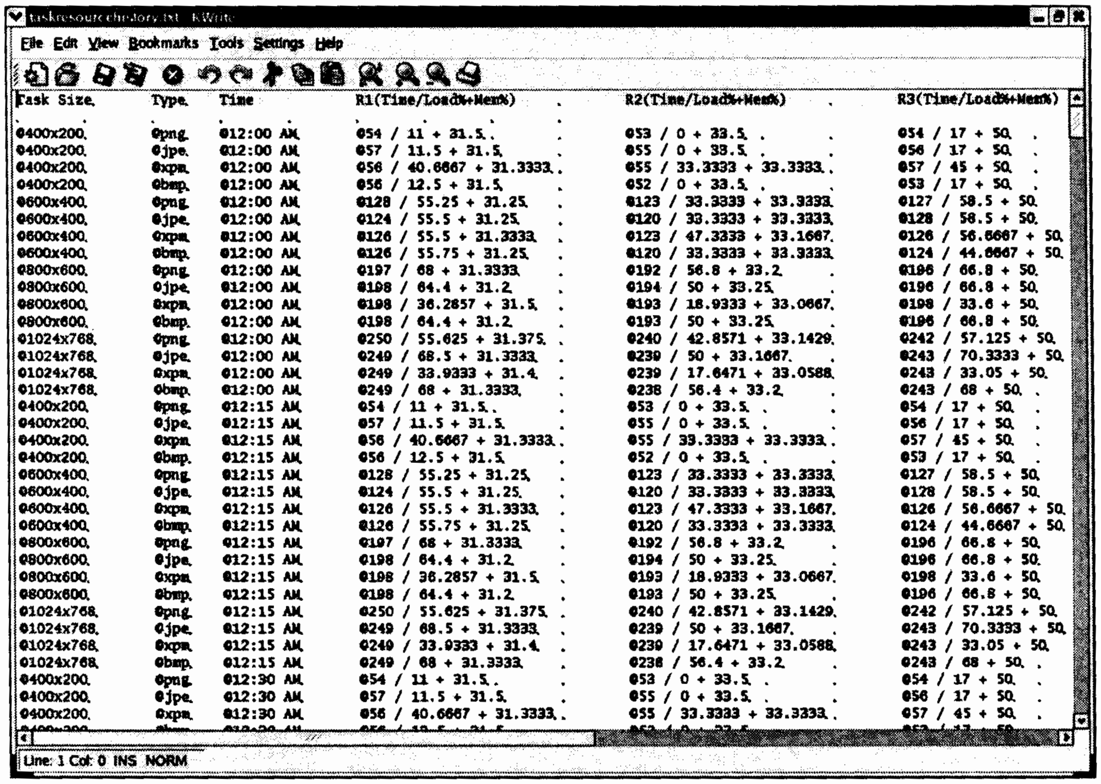


```
Task Size.     Type.   Time     .   Res.   (Time/Load% + Memory%)

@1024x768.     @png.   @01:15 PM.   @1.    @258 / 65.6364 + 32.0909
@1024x768.     @jpe.   @01:15 PM.   @1.    @256 / 78.7778 + 32
@1024x768.     @xpm.   @01:15 PM.   @1.    @258 / 25.7692 + 32.0769
@1024x768.     @bmp.   @01:15 PM.   @1.    @258 / 66.5556 + 32.3333
@800x600.      @png.   @01:15 PM.   @1.    @200 / 74.8571 + 32
@400x200.      @bmp.   @01:15 PM.   @1.    @56 / 40.6667 + 32
@800x600.      @jpe.   @01:15 PM.   @1.    @202 / 75.7143 + 32
@800x600.      @xpm.   @01:15 PM.   @1.    @201 / 35.2778 + 32
@800x600.      @bmp.   @01:15 PM.   @1.    @202 / 76 + 32
@600x400.      @png.   @01:15 PM.   @1.    @125 / 66.2 + 32
@600x400.      @jpe.   @01:15 PM.   @1.    @125 / 66.2 + 32
@600x400.      @xpm.   @01:15 PM.   @1.    @126 / 71.8333 + 32
@600x400.      @bmp.   @01:15 PM.   @1.    @126 / 66.6 + 32
@400x200.      @png.   @01:15 PM.   @1.    @56 / 44 + 32
@400x200.      @jpe.   @01:15 PM.   @1.    @56 / 39.3333 + 32
@400x200.      @xpm.   @01:15 PM.   @1.    @58 / 38.3333 + 32
@1024x768.     @png.   @01:17 PM.   @2.    @240 / 42.8571 + 33.1429
@600x400.      @jpe.   @01:17 PM.   @2.    @120 / 33.3333 + 33.3333
@600x400.      @xpm.   @01:17 PM.   @2.    @123 / 47.3333 + 33.1667
@600x400.      @bmp.   @01:17 PM.   @2.    @120 / 33.3333 + 33.3333
@400x200.      @png.   @01:17 PM.   @2.    @53 / 0 + 33.5
@400x200.      @jpe.   @01:17 PM.   @2.    @55 / 0 + 33.5
@400x200.      @xpm.   @01:17 PM.   @2.    @55 / 33.3333 + 33.3333
@400x200.      @bmp.   @01:17 PM.   @2.    @52 / 0 + 33.5
@1024x768.     @jpe.   @01:17 PM.   @2.    @239 / 50 + 33.1667
@1024x768.     @xpm.   @01:17 PM.   @2.    @239 / 17.6471 + 33.0588
@1024x768.     @bmp.   @01:17 PM.   @2.    @238 / 56.4 + 33.2
@800x600.      @png.   @01:17 PM.   @2.    @192 / 56.8 + 33.2
@800x600.      @jpe.   @01:17 PM.   @2.    @194 / 50 + 33.25
@800x600.      @xpm.   @01:17 PM.   @2.    @193 / 18.9333 + 33.0667
@800x600.      @bmp.   @01:17 PM.   @2.    @193 / 50 + 33.25
@600x400.      @png.   @01:17 PM.   @2.    @123 / 33.3333 + 33.3333
@1024x768.     @png.   @01:18 PM.   @3.    @248 / 50 + 50.25
@600x400.      @jpe.   @01:18 PM.   @3.    @130 / 57 + 50.25
@600x400.      @xpm.   @01:18 PM.   @3.    @127 / 55.5 + 50.1667
@600x400.      @bmp.   @01:18 PM.   @3.    @130 / 42.6667 + 50.3333
```

Fig. C-6: Task History File – 1

After each twenty four hours the data from log file 1 is entered into the 2nd log file which The preview of 2nd log file is shown in figure 8-7.


taskresourcehistory.txt - KWrite
File Edit View Bookmarks Tools Settings Help
Line: 1 Col: 0 INS NORM

```
Task Size.       Type.   Time          R1(Time/Load%+Mem%)          .    R2(Time/Load%+Mem%)            .   R3(Time/Load%+Mem%)
.     .          .       .     .       .      .      .              .
@400x200.        @png.   @12:00 AM.    @54 / 11 + 31.5. .           .    @53 / 0 + 33.5.  .             .   @54 / 17 + 50.  .
@400x200.        @jpe.   @12:00 AM.    @57 / 11.5 + 31.5.           .    @55 / 0 + 33.5.  .             .   @56 / 17 + 50.  .
@400x200.        @xpm.   @12:00 AM.    @56 / 40.6667 + 31.3333. .        @55 / 33.3333 + 33.3333. .         @57 / 45 + 50.  .
@400x200.        @bmp.   @12:00 AM.    @56 / 12.5 + 31.5.           .    @52 / 0 + 33.5.  .             .   @53 / 17 + 50.  .
@600x400.        @png.   @12:00 AM.    @128 / 55.25 + 31.25.        .    @123 / 33.3333 + 33.3333.          @127 / 58.5 + 50.
@600x400.        @jpe.   @12:00 AM.    @124 / 55.5 + 31.25.         .    @120 / 33.3333 + 33.3333.          @128 / 58.5 + 50.
@600x400.        @xpm.   @12:00 AM.    @126 / 55.5 + 31.3333.       .    @123 / 47.3333 + 33.1667.          @126 / 56.6667 + 50.
@600x400.        @bmp.   @12:00 AM.    @126 / 55.75 + 31.25.        .    @120 / 33.3333 + 33.3333.          @124 / 44.6667 + 50.
@800x600.        @png.   @12:00 AM.    @197 / 68 + 31.3333.         .    @192 / 56.8 + 33.2.            .   @196 / 66.8 + 50.
@800x600.        @jpe.   @12:00 AM.    @198 / 64.4 + 31.2.          .    @194 / 50 + 33.25.             .   @196 / 66.8 + 50.
@800x600.        @xpm.   @12:00 AM.    @198 / 36.2857 + 31.5.       .    @193 / 18.9333 + 33.0667.          @198 / 33.6 + 50.
@800x600.        @bmp.   @12:00 AM.    @198 / 64.4 + 31.2.          .    @193 / 50 + 33.25.             .   @196 / 66.8 + 50.
@1024x768.       @png.   @12:00 AM.    @250 / 55.625 + 31.375.      .    @240 / 42.8571 + 33.1429.          @242 / 57.125 + 50.
@1024x768.       @jpe.   @12:00 AM.    @249 / 68.5 + 31.3333.       .    @239 / 50 + 33.1667.           .   @243 / 70.3333 + 50.
@1024x768.       @xpm.   @12:00 AM.    @249 / 33.9333 + 31.4.       .    @239 / 17.6471 + 33.0588.          @243 / 33.05 + 50.
@1024x768.       @bmp.   @12:00 AM.    @249 / 68 + 31.3333.         .    @238 / 56.4 + 33.2.            .   @243 / 68 + 50. .
@400x200.        @png.   @12:15 AM.    @54 / 11 + 31.5. .           .    @53 / 0 + 33.5.  .             .   @54 / 17 + 50.  .
@400x200.        @jpe.   @12:15 AM.    @57 / 11.5 + 31.5.           .    @55 / 0 + 33.5.  .             .   @56 / 17 + 50.  .
@400x200.        @xpm.   @12:15 AM.    @56 / 40.6667 + 31.3333. .        @55 / 33.3333 + 33.3333. .         @57 / 45 + 50.  .
@400x200.        @bmp.   @12:15 AM.    @56 / 12.5 + 31.5.           .    @52 / 0 + 33.5.  .             .   @53 / 17 + 50.  .
@600x400.        @png.   @12:15 AM.    @128 / 55.25 + 31.25.        .    @123 / 33.3333 + 33.3333.          @127 / 58.5 + 50.
@600x400.        @jpe.   @12:15 AM.    @124 / 55.5 + 31.25.         .    @120 / 33.3333 + 33.3333.          @128 / 58.5 + 50.
@600x400.        @xpm.   @12:15 AM.    @126 / 55.5 + 31.3333.       .    @123 / 47.3333 + 33.1667.          @126 / 56.6667 + 50.
@600x400.        @bmp.   @12:15 AM.    @126 / 55.75 + 31.25.        .    @120 / 33.3333 + 33.3333.          @124 / 44.6667 + 50.
@800x600.        @png.   @12:15 AM.    @197 / 68 + 31.3333.         .    @192 / 56.8 + 33.2.            .   @196 / 66.8 + 50.
@800x600.        @jpe.   @12:15 AM.    @198 / 64.4 + 31.2.          .    @194 / 50 + 33.25.             .   @196 / 66.8 + 50.
@800x600.        @xpm.   @12:15 AM.    @198 / 36.2857 + 31.5.       .    @193 / 18.9333 + 33.0667.          @198 / 33.6 + 50.
@800x600.        @bmp.   @12:15 AM.    @198 / 64.4 + 31.2.          .    @193 / 50 + 33.25.             .   @196 / 66.8 + 50.
@1024x768.       @png.   @12:15 AM.    @250 / 55.625 + 31.375.      .    @240 / 42.8571 + 33.1429.          @242 / 57.125 + 50.
@1024x768.       @jpe.   @12:15 AM.    @249 / 68.5 + 31.3333.       .    @239 / 50 + 33.1667.           .   @243 / 70.3333 + 50.
@1024x768.       @xpm.   @12:15 AM.    @249 / 33.9333 + 31.4.       .    @239 / 17.6471 + 33.0588.          @243 / 33.05 + 50.
@1024x768.       @bmp.   @12:15 AM.    @249 / 68 + 31.3333.         .    @238 / 56.4 + 33.2.            .   @243 / 68 + 50. .
@400x200.        @png.   @12:30 AM.    @54 / 11 + 31.5. .           .    @53 / 0 + 33.5.  .             .   @54 / 17 + 50.  .
@400x200.        @jpe.   @12:30 AM.    @57 / 11.5 + 31.5.           .    @55 / 0 + 33.5.  .             .   @56 / 17 + 50.  .
@400x200.        @xpm.   @12:30 AM.    @56 / 40.6667 + 31.3333. .        @55 / 33.3333 + 33.3333. .         @57 / 45 + 50.  .
```

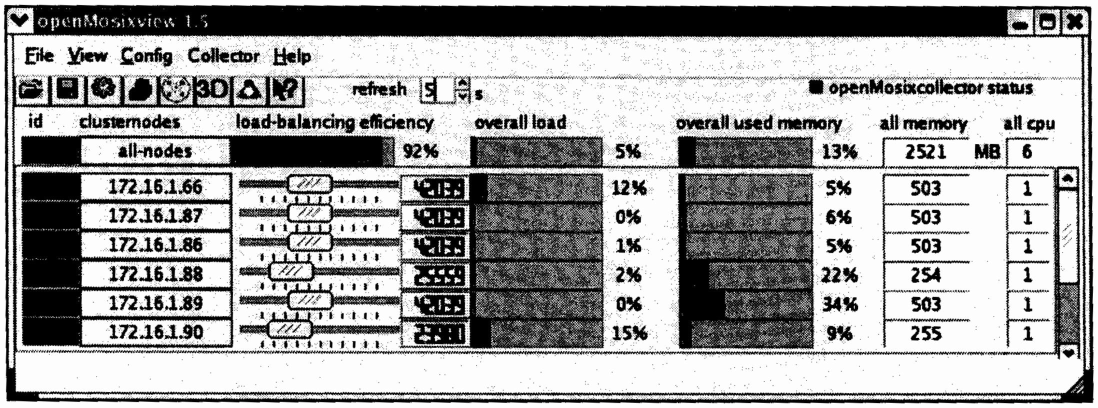
Fig. C-7: Task Resource History File – 2

C.2 RESOURCE COLLECTOR SCREENS

The second module of this project is to collect the resource information from all the resources in the platform and update it after certain intervals. The following are the screens shots of openmosix view which is used as the resource collector module in this project.

C.2.1 RESOURCE COLLECTOR MAIN SCREEN

The main screen of resource collector which shows the information about all the resources and their utilization is shown in figure 8-6.

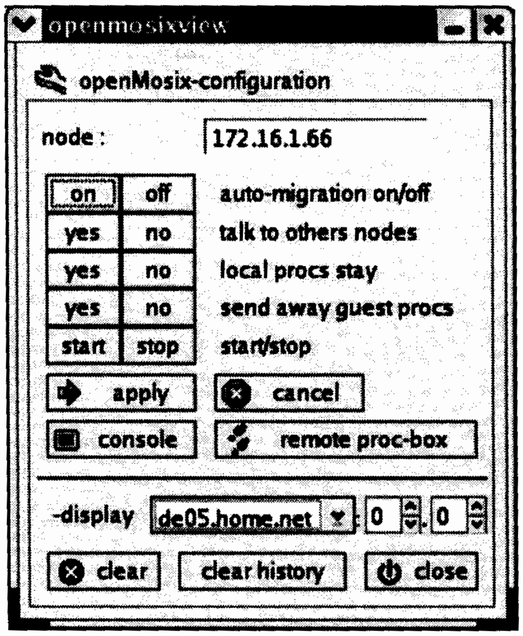


id	clusternodes	load-balancing efficiency	overall load	overall used memory	all memory		all cpu
	all-nodes	92%	5%	13%	2521	MB	6
	172.16.1.66	[illegible]	12%	5%	503		1
	172.16.1.87	[illegible]	0%	6%	503		1
	172.16.1.86	[illegible]	1%	5%	503		1
	172.16.1.88	[illegible]	2%	22%	254		1
	172.16.1.89	[illegible]	0%	34%	503		1
	172.16.1.90	[illegible]	15%	9%	255		1

Fig C-8: Resource Collector Main Screen

C.2.2 RESOURCE CONFIGURATION SCREEN

Using resource configuration screen you can configure the usage of your particular resource in the environment e.g. task migration decisions, connection with other resources and migration of local or guest processes. The following screen shown in figure 8-7 will be displayed for resource configuration.


openmosixview

openMosix-configuration

node : 172.16.1.66

on	off	auto-migration on/off
yes	no	talk to others nodes
yes	no	local procs stay
yes	no	send away guest procs
start	stop	start/stop

Fig C-9: Resource Configuration Screen

C.2.3 RESOURCE UTILIZATION GRAPHICAL VIEW

The graphical view of individual resource as well as the whole grid environment is displayed in the following screen shown in figures 8-7 and 8-8.

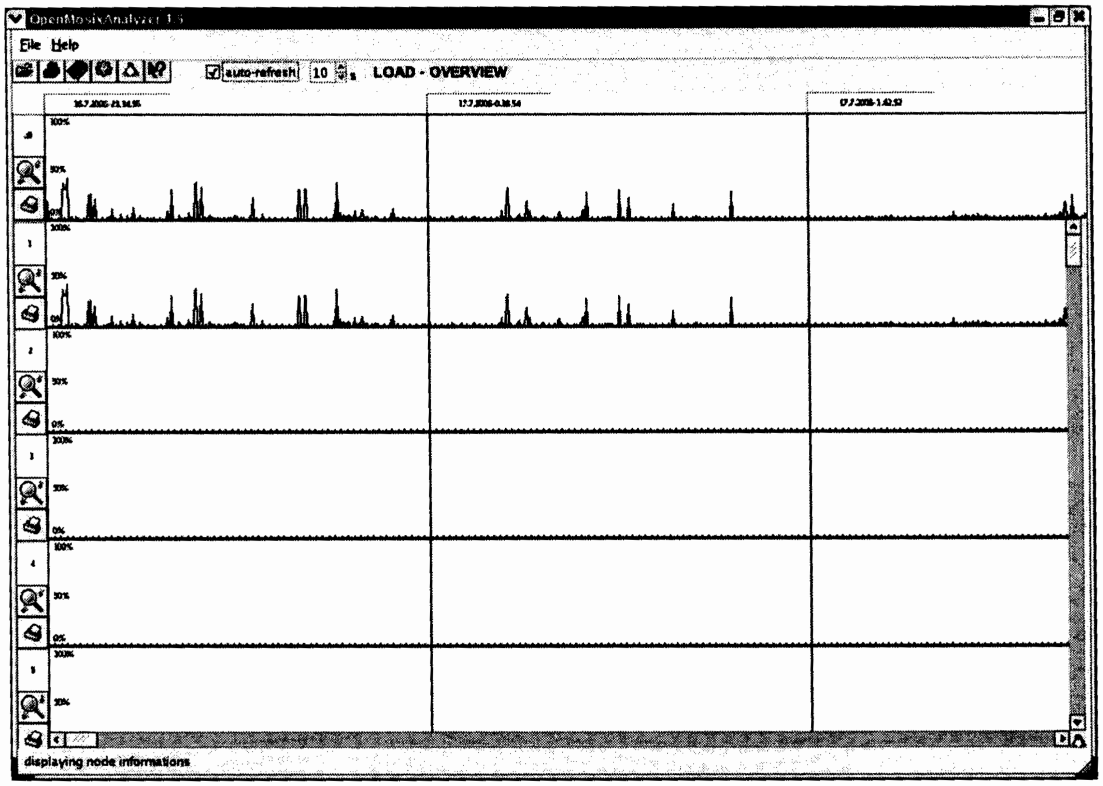
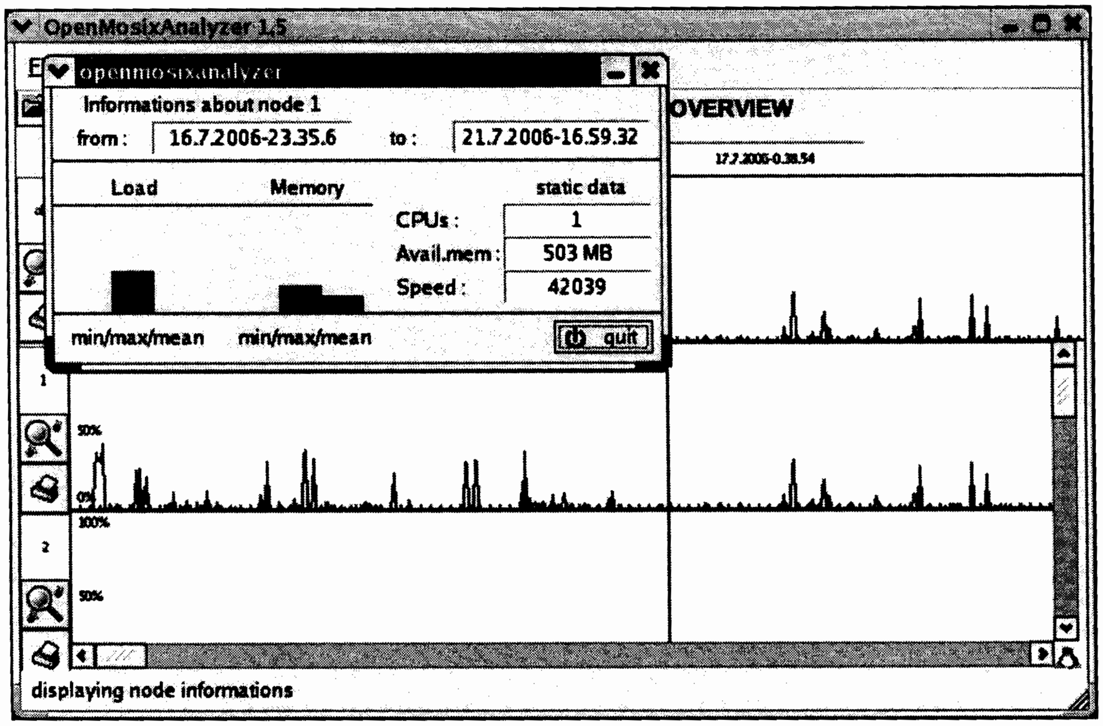


Fig C-10: Resource Utilization view of all resources



Informations about node 1
from : 16.7.2006-23.35.6 to : 21.7.2006-16.59.32

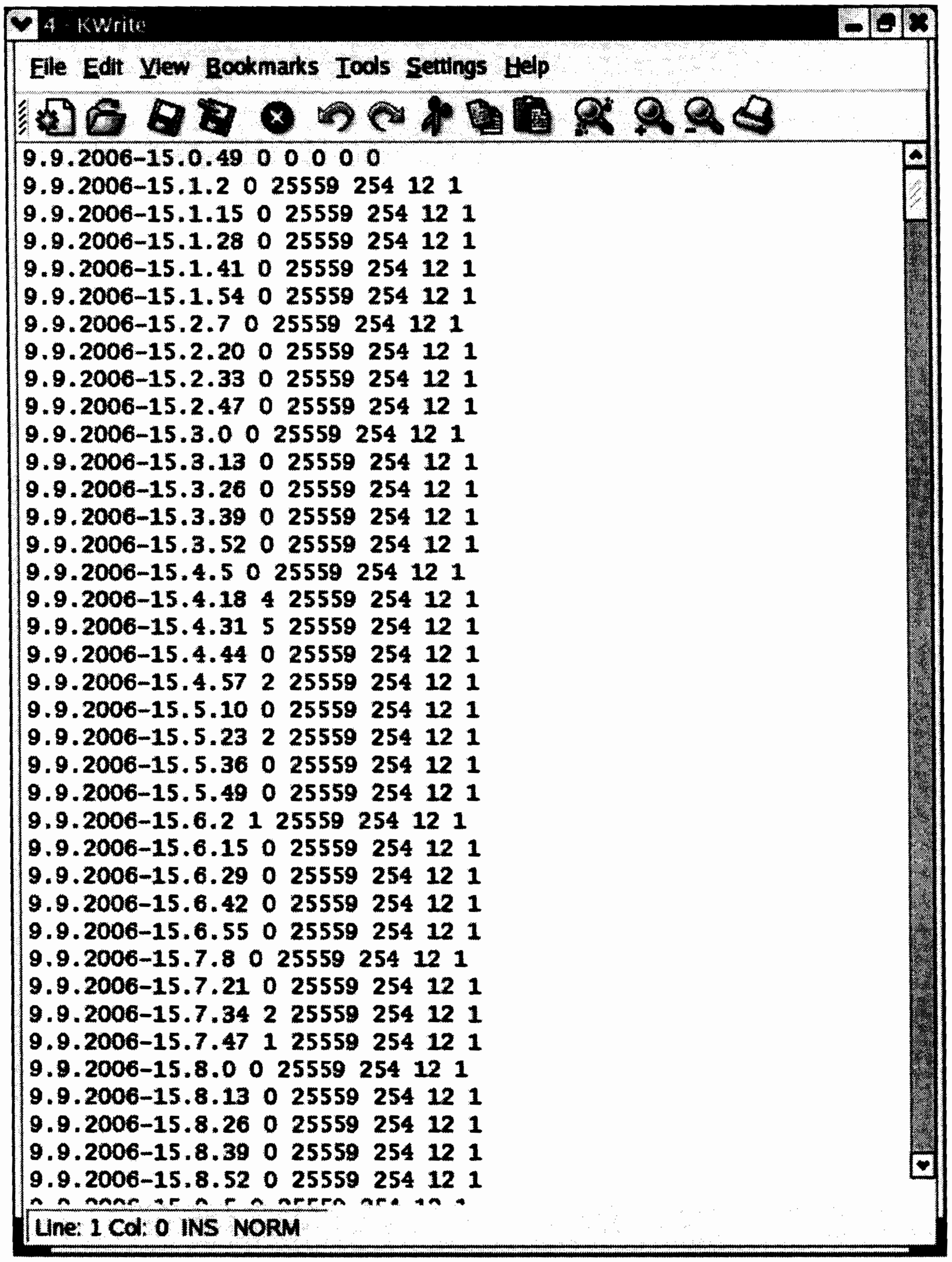
static data
CPUs : 1
Avail.mem : 503 MB
Speed : 42039

displaying node informations

Fig C-11: Resource Utilization view of individual resource

C.2.4 RESOURCE HISTORY

Resource history generated by resource collector is used to set up a threshold policy for a resource. The preview of the resource history file is shown in figure 8-12.



```
9.9.2006-15.0.49 0 0 0 0 0
9.9.2006-15.1.2 0 25559 254 12 1
9.9.2006-15.1.15 0 25559 254 12 1
9.9.2006-15.1.28 0 25559 254 12 1
9.9.2006-15.1.41 0 25559 254 12 1
9.9.2006-15.1.54 0 25559 254 12 1
9.9.2006-15.2.7 0 25559 254 12 1
9.9.2006-15.2.20 0 25559 254 12 1
9.9.2006-15.2.33 0 25559 254 12 1
9.9.2006-15.2.47 0 25559 254 12 1
9.9.2006-15.3.0 0 25559 254 12 1
9.9.2006-15.3.13 0 25559 254 12 1
9.9.2006-15.3.26 0 25559 254 12 1
9.9.2006-15.3.39 0 25559 254 12 1
9.9.2006-15.3.52 0 25559 254 12 1
9.9.2006-15.4.5 0 25559 254 12 1
9.9.2006-15.4.18 4 25559 254 12 1
9.9.2006-15.4.31 5 25559 254 12 1
9.9.2006-15.4.44 0 25559 254 12 1
9.9.2006-15.4.57 2 25559 254 12 1
9.9.2006-15.5.10 0 25559 254 12 1
9.9.2006-15.5.23 2 25559 254 12 1
9.9.2006-15.5.36 0 25559 254 12 1
9.9.2006-15.5.49 0 25559 254 12 1
9.9.2006-15.6.2 1 25559 254 12 1
9.9.2006-15.6.15 0 25559 254 12 1
9.9.2006-15.6.29 0 25559 254 12 1
9.9.2006-15.6.42 0 25559 254 12 1
9.9.2006-15.6.55 0 25559 254 12 1
9.9.2006-15.7.8 0 25559 254 12 1
9.9.2006-15.7.21 0 25559 254 12 1
9.9.2006-15.7.34 2 25559 254 12 1
9.9.2006-15.7.47 1 25559 254 12 1
9.9.2006-15.8.0 0 25559 254 12 1
9.9.2006-15.8.13 0 25559 254 12 1
9.9.2006-15.8.26 0 25559 254 12 1
9.9.2006-15.8.39 0 25559 254 12 1
9.9.2006-15.8.52 0 25559 254 12 1
```

Fig C-12: Resource History Files