

# NOVEL ANN BASED LOAD BALANCING TECHNIQUE FOR HETEROGENOUS ENVIRONMENT

T4465



## *Developed By*

Muniza Salim (Reg# 146-CS/MS/2003)  
Ammara Manzoor (Reg# 150-CS/MS/2003)

## *Supervised By*

Prof.Dr.Khalid Rashid



Department of Computer Science  
Faculty of Basic and Applied Sciences  
International Islamic University  
Islamabad  
2007

**In The Name Of**

**ALLAH**

**The Most Merciful  
The Most Beneficent**

**Department of Computer Science  
International Islamic University Islamabad**

Date: 2.6.2007

**Final Approval**

This is to certify that we have read the thesis entitled "Novel ANN based load balancing technique for Heterogeneous Environment" submitted by Muniza Salim, Reg #(146-MS/CS/03), Ammara Manzoor, Reg # (150-MS/CS/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 Sattar  
Ex. Director General,  
Pakistan Computer Bureau.



**Internal Examiner**

Ms Muneera Bano  
Lecturer,  
Dept of Computer Science,  
Faculty of Basic and Applied Sciences,  
International Islamic University, Islamabad.



**Supervisor**

Prof. Dr Khalid Rashid  
~~Dr. Rashid~~  
Faculty of Basic and Applied Sciences,  
International Islamic University, Islamabad.



**A DISSERTATION SUBMITTED TO  
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 DEGREE OF  
MS IN COMPUTER SCIENCE.**

Dedicated to our dearest and affectionate parents  
who encouraged us to work hard  
and without their moral support  
this work could never  
become a reality.

## **DECLARATION**

We hereby declare that this software and accompanied thesis neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this software and thesis on the basis of our own efforts and under sincere guidance of our teachers. No portion of the work presented in this report has been submitted in support of an application for another degree or qualification of this or any other university or institute of learning.

Muniza Salim  
146-CS/MS/03

Ammara Manzoor  
150-CS/MS/03

## ACKNOWLEDGEMENT

We bestow all praises, acclamations and appreciation to Almighty Allah the most merciful and compassionate, the most gracious and beneficent, whose bounteous blessing enabled us to perceive and pursue higher ideals of life. All praises and respects are for the Holy Prophet Mohammad (Peace be upon him) who enabled us to recognize the creator.

Without the support and prayers of our families we could have never completed this project work. They always helped us in our difficult times and boosted our morals. We would like to convey them special thanks for their best wishes, encouragement and support in not only this project but throughout our lives, without which we would have not been able to achieve anything worthy.

We gratefully acknowledge the Supervision of Prof.Dr.Khalid Rashid Dean, Faculty of Basic and Applied Sciences who was very kind and helpful throughout the project.

We would also like to thank Dr. M.Sikander Hayat Khiyal Head, Faculty of Basic and Applied Sciences and all faculty members for their cooperation and healthy suggestions through out our academic period at Islamic University, Islamabad.

Muniza Salim  
146-CS/MS/03

Ammara Manzoor  
150-CS/MS/03

## **PROJECT IN BRIEF**

- PROJECT TITLE** : Novel ANN based Load balancing in Heterogeneous Environment
- OBJECTIVE** : To develop a dynamic and adaptive ANN based load balancing technique for the heterogeneous systems.
- UNDERTAKEN BY** : Muniza Salim Reg # 146-CS/MS/03  
Ammara Manzoor Reg # 150-CS/MS/03
- SUPERVISED BY** : Prof. Dr. Khalid Rashid  
Dean, Faculty of Basic and Applied sciences,  
International Islamic University, Islamabad.
- STARTED ON** : SEPT 2004
- COMPLETED ON** : APRIL 2007
- TOOLS** : OpenMosix  
Povray 3.0  
Qt Designer 3.1  
Rational Rose 2000  
Visio 2002
- OPERATING SYSTEM USED** : Linux Red Hat 9
- SYSTEM USED** : Pentium II, Pentium III, Pentium IV



## **Abstract**

Present study aims to solve load balancing decisions using ANN in heterogeneous environment. Grid computing is an emerging computing paradigm and is distinguished from distributed computing by its efficient and optimal utilization of heterogeneous, loosely coupled resources tied to work load management. However, complexity incurred in efficient management of heterogeneous, geographically distributed and dynamically available resources has become one of the most challenging issues in grid computing. A lot of parameters have to be taken into consideration to efficiently utilize the grid resources. Since ANN are best at identifying patterns or trends in data, their ability to learn by examples makes them very flexible and powerful. Experimental results suggest that once trained, ANN outperforms other heuristic approaches for large tasks. However for small tasks, ANN suffers from extensive overheads.

## **Abbreviations**

<b>DSS</b>	Distributed Self Scheduling
<b>EJB</b>	Enterprise Java beans
<b>KNN</b>	Kohonen Neural Network
<b>NOW</b>	Network of Workstations
<b>OGSA</b>	Open Grid Services Architecture
<b>OGSI</b>	Open Grid Services Infrastructure
<b>POV</b>	Persistence of vision
<b>SASH</b>	Self Adjusting Scheduling for Heterogeneous Systems
<b>SOA</b>	Service Oriented Architecture
<b>SOM</b>	Self-Organizing Map
<b>WTA</b>	Winner-take-all

**TABLE OF CONTENTS**

<b>CH.NO</b>	<b>CONTENTS</b>	<b>PAGE NO</b>
<b>1.</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 Grid Computing	2
	1.1.1 Grid Architecture	3
	1.1.2 Open Grid Services Architecture	4
	1.2 Load Balancing	5
	1.2.1 Static Load Balancing	6
	1.2.2 Dynamic Load Balancing	6
	1.3 Neural Network	6
	1.3.1 Biological Neural Systems	7
	1.3.2 Artificial Neural Networks	7
	1.3.2.1 Artificial Neuron	8
	1.3.2.2 McCulloch Pitts Neuron	8
	1.4 Self-Organizing Map (Kohonen)	9
<b>2.</b>	<b>LITERATURE SURVEY</b>	<b>14</b>
	2.1 Problem Definition	15
<b>3.</b>	<b>PROPOSED SOLUTION</b>	<b>16</b>
	3.1 Methodology	17
	3.2 Functional Modules	18
	3.2.1 Resource Collector	18
	3.2.2 Resource Monitor	19
	3.2.3 Resource Analyzer	19
	3.2.4 KNN Load balancer	19
	3.2.5 Task Collector	19
	3.2.6 Task Manager	20
	3.2.7 Task Monitor	20
	3.2.8 Performance Monitor	20
<b>4.</b>	<b>SYSTEM DESIGN</b>	<b>21</b>
	4.1 Object-Oriented Analysis and Design	21
	4.1.1 Class Diagram	21
	4.1.2 Sequence Diagram	22
	4.1.2.1 Collect Resource Information	22
	4.1.2.2 Analyze and Monitor Resource Information	24
	4.1.2.3 Load balancing and Re-balancing	25
	4.1.2.4 Performance Monitor	26
	4.1.2.5 Task Scheduling	27
	4.1.2.6 Task Log Generation	28

	4.2 System Architecture	29
5.	<b>SYSTEM DEVELOPMENT</b>	31
	5.1 Tools	31
	5.1.1 Languages	31
	5.1.2 Editors	31
	5.1.3 Office Tools	31
	5.1.4 Benchmarking tools	31
	5.1.5 Grid Management tools	32
	5.2 Pseudo code	32
	5.2.1 Resource Collector	32
	5.2.2 Resource Evaluator	32
	5.2.3 KNN Load Balancer	33
	5.2.4 Task collector	33
6.	<b>RESULTS</b>	34
	6.1 Experimental Setup	34
	6.2 Description of Experiments	34
	6.3 Experimental Results	34
	6.4 Conclusion	38
	<b>REFERENCES AND BIBLIOGRAPHY</b>	39
	<b>APPENDIX-A</b>	A-1
	<b>APPENDIX-B</b>	B-1

**CHAPTER 1**  
**INTRODUCTION**

# 1. INTRODUCTION

Distributed Computing has been the holygrail of software industry for the last 3 decades, solving problems in domains of business applications, scientific computations and large scale collaborative systems to name a few. The motivation for parallel and distributed systems comes from the applications as well as from hardware limitation. Complex applications require complex machines; like scientific applications, solving NP complete problems or analyzing massive amount of data. Some applications are inherently distributed like web based applications, emails, news, electronic conferencing; multiplayer games etc. Besides, there are applications that are easier to build in components.

As computation, storage, and communication technologies steadily improve, increasingly large, complex, and resource-intensive applications are being developed both in research institutions and in industry. It is a common observation that computational resources are failing to meet the demand of those applications. The power of network, storage, and computing resources is projected to double every 9, 12, and 18 months, respectively [1]. Those three constants have important implications. Anticipating the trends in storage capacities (and price), application developers and users are planning increasingly large runs that will operate on and generate petabytes of data [1]. Although microprocessors are reaching impressive speeds, in the long run they are falling behind storage. As a result, it is becoming increasingly difficult to gather enough computational resources for running applications at a single location. Fortunately, improvements in wide-area networking make it possible to aggregate distributed resources in various collaborating institutions

On the other hand, due to the high cost of dedicated parallel and cluster machines, one is forced to look for alternatives to fulfill the requirement for high performance computing in modern scientific research. In the early days of parallel computing, the only users of parallel computing technology consisted of cutting edge researchers with multi-million dollar budgets. In the mid-90's, machines such as the Cray T3E [2] were the most powerful parallel machines available, with expensive custom built interconnections between hundreds or even thousands of tightly coupled processors in a single shared box [3]. For the ordinary researchers, these specialized parallel machines have always been too expensive to be considered as a research tool. Distributed computing emerged as a viable alternative to dedicated parallel computing. By harnessing the spare clock cycles of idle machines, it is possible to emulate the computing power offered by dedicated supercomputers. Vast advances and constantly declining costs of hardware technology has also encouraged utilizing them in collaboration and cooperation.

Over the period, different architecture of software systems has evolved, typically following the evolution of hardware systems from mainframes in the 70's, client/server systems in the 80's and early 90's, thin-clients in the late 90's and peer-to-peer distributed systems at the start of this decade. Client/Server is a network architecture which separates the client (often a graphical user interface) from the server. Each instance of the client software can send requests to a server or application server. Client/Server architecture is intended to provide a scalable architecture, whereby each computer or process on the network is either a client or a server. Server software generally, but not always, runs on powerful computers dedicated for exclusive use to running the business application. Client software on the other hand generally

runs on common PCs or workstations. Clients get all or most of their information and rely on the application server for things such as configuration files, stock quotes, business application programs, or to offload computer-intensive application tasks back to the server in order to keep the client computers (and client computer user) free to perform other tasks.

Another type of network architecture is known as a Peer-to-Peer architecture because each node or instance of the program is both a "client" and a "server" and each has equivalent responsibilities. Both client/server and peer-to-peer architectures are in wide use. An important goal in peer-to-peer networks is that all clients provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demand on the system increases, the total capacity of the system also increases. This is not true of client-server architecture with a fixed set of servers, in which adding more clients could mean slower data transfer for all users. 3-tier architecture 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 architecture 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. A computer cluster is a group of loosely coupled computers that work together closely so that in many respects it can be viewed as though it were a single computer. Clusters are commonly, but not always, connected through fast local area networks. Clusters are usually deployed to improve speed and/or reliability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or reliability. Clusters are categorized as follows: High Availability clusters (HA), Load Balancing Clusters, High Performance Clusters and Grid Computing.

## 1.1 Grid Computing

Grid computing can be defined in many ways. According to Buyya "A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements" [4]. According to Carl Kesselman "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [5].

Grid computing is an emerging computing model that provides the ability to perform higher throughput computing by taking advantage of many networked computers to model a virtual computer architecture that is able to distribute process execution across a parallel infrastructure. Grids use the resources of many separate computers connected by a network to solve large-scale computation problems. Grids provide the ability to perform computations on large data sets, by breaking them down into many smaller ones, or provide the ability to perform many more computations at once than would be possible on a single computer, by modeling a parallel division of labor between processes[6].It offers a model for solving massive computational problems by making use of the unused resources (CPU cycles and/or disk storage) of large numbers of disparate, often desktop, computers treated as a virtual cluster embedded in a distributed telecommunications infrastructure. Grid computing focus

on the ability to support computation across administrative domains sets it apart from traditional computer clusters or traditional distributed computing. Resource management in grid is complex due to various factors like site autonomy, resource heterogeneity, co-allocation of resources. In grid systems resources are added and removed dynamically. It varies in different systems like Condor [7], Nimrod-G [8], globus [9], legion [10] etc.

Grid computing is often confused with cluster computing. The key difference is that a cluster is a single set of nodes sitting in one location, while a Grid is composed of many clusters and other kinds of resources (e.g. networks, storage facilities). 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 [11].

### 1.1.1 Grid Architecture

Grid architecture has been developed for the establishment and management of cross-organizational resource sharing. It identifies the basic components of a grid system. The grid architecture defines the purpose and functions of its components, while indicating how these components interact with one another. The main focus of the architecture is on interoperability among resource providers and users in order to establish the sharing relationships [12]. This interoperability, in turn, necessitates common protocols at each layer of the architectural model, which leads to the definition of a grid protocol architecture as shown in Figure 1.1. This protocol architecture defines common mechanisms, interfaces, schema, and protocols at each layer, by which users and resources can negotiate, establish, manage, and share resources. Fig 1.1 shows the component layers of the grid architecture and the capabilities of each layer. The description of core features of each component layer is as follows.

- Fabric layer defines the interface to local resources, which may be shared. This includes computational resources, data storage, networks, catalogs, software modules, and other system resources.
- Connectivity layer defines the basic communication and authentication protocols required for grid-specific networking-service transactions.
- Resource layer uses the communication and security protocols (defined by the connectivity layer) to control secure negotiation, initiation, monitoring, accounting, and payment for the sharing of functions of individual resources. The resource layer calls the fabric layer functions to access and control local resources. This layer only handles individual resources, ignoring global states and atomic actions across the resource collection pool, which are the responsibility of the collective layer.
- Collective layer is responsible for all global resource management and interaction with collections of resources. This protocol layer implements a wide variety of sharing behaviors using a small number of resource-layer and connectivity-layer protocols.
- Application layer enables the use of resources in a grid environment through various collaboration and resource access protocols.



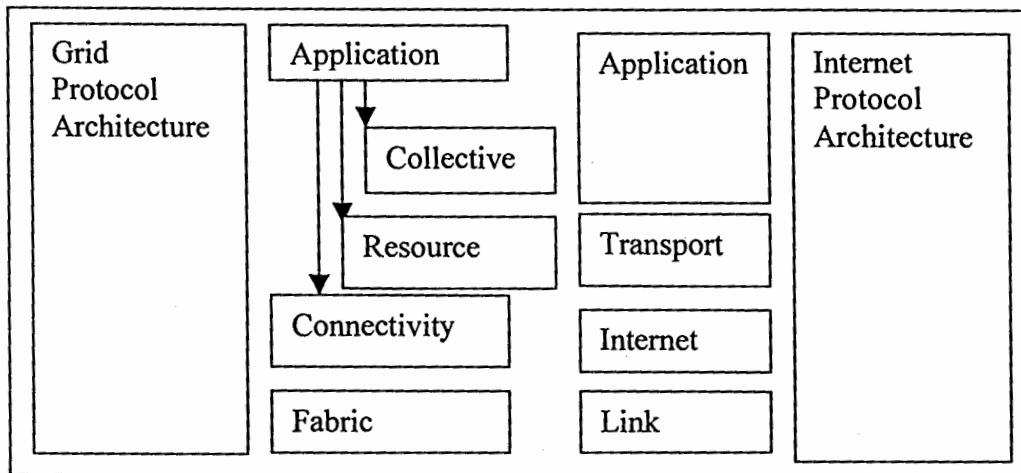


Fig 1.1 Grid Architecture

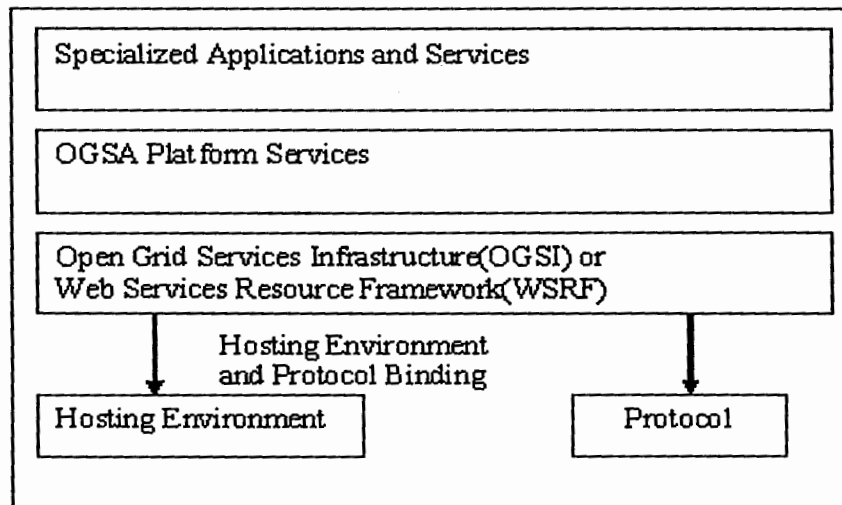


Fig 1.2 OGSA Platform Architecture

### 1.1.2 Open Grid Services Architecture (OGSA)

OGSA is a distributed interaction and computing architecture based around services, assuring interoperability on heterogeneous systems so that different types of resources can communicate and share information. OGSA has been described as a refinement of the emerging Web Services architecture, specifically designed to support Grid requirements [13]. OGSA is a result of the alignment of existing grid standards with emerging service-oriented architecture (SOAs) as well as with the Web. A service-oriented architecture (SOA) is a specific type of distributed system framework which maintains agents that act as

“software services,” performing well-defined operations. OGSA provides a uniform way to describe grid services and define a common pattern of behavior for these services. OGSA is a layered architecture, as shown in Figure 1.2, with clear separation of the functionalities at each layer. As seen in the figure, the core architecture layers are the Open Grid Services Infrastructure (OGSI) and OGSA platform services [12]. The platform services establish a set of standard services including policy, logging, service level management, and other networking services. High-level applications and services use these lower-layer platform core components to become a part of a resource-sharing grid.

## 1.2 Load Balancing

Load balancing includes 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. It is important that the overhead of executing the load balancing algorithm does not contribute significantly to the overall processing or communications load. The primary function of a load-balancing strategy is to recommend decisions that will improve performance. Performance here, mean to reduce the overall execution time or attain speed-up over local execution of tasks or improvement in some other parameter compared to a base strategy [14].

Distributed scheduling algorithms may be static, dynamic or preemptive. Static algorithms allocate processes to processors at run time while taking no account of current network load. Dynamic algorithms are more flexible, though more computationally expensive, and give some consideration to the network load before allocating the new process to a processor. Preemptive algorithms are more expensive and flexible still, and may migrate running processes from one host to another if deemed beneficial. Load balancing is an important technique to enhance the performance of distributed computing systems. The objective is to make workload as equal as possible in order to achieve good speedup. Load distribution seeks to improve the performance of a distributed system, usually in terms of response time or resource availability, by allocating workload amongst a set of cooperating hosts. It consists in taking benefit of the fact that, in the network, some machines are less loaded than others (or even totally inactive), by running some processes on a less loaded machine. Incoming requests should be evenly distributed among all the sites to achieve quick response and to enhance the system throughput. Thus, the system resources can get full utilization [15].

The workload of a site consists of the combined demands on its resources from all of the local processes. The absolute and relative utilizations of various resources at each site, and of various sites across the network, are highly dynamic quantities. The dynamic nature of load causes frequent imbalances: certain resources local to a site may be overloaded even as similar resources at a remote site are underutilized or idle. With increase in the speed of

individual processors, and with growth in the scale of typical systems, there are parallel increases in both the magnitude and the frequency of load imbalances.[16]

It has been proven that finding optimal schedules for the load-balancing problem is NP-complete problem, even when the communication cost is ignorable [16]. Because of the difficulty for reaching optimal schedules, most of the research efforts have been focused on finding sub-optimal solutions by some approximate and heuristic approaches.

### 1.2.1 Static Load Balancing

Static load balancing schemes predetermine the individual processor loads prior to runtime. Static load balancing lends itself best to applications that will be run on dedicated systems with predefined characteristics. The various tasks are usually partitioned accordingly at compile /link time. The assignment of tasks may be influenced by the problem size, number of processors available, relative performance of individual processors, etc. However, once the assignment is made, it is not changed at runtime. This is the simplest scheme of load balancing and is usually done by mapping tasks to processors.

### 1.2.2 Dynamic Load Balancing

Dynamic load balancing is a method of load balancing that is done dynamically at runtime. Tasks are spread evenly across the available processors and the workload is adjusted accordingly. Dynamic load balancing lends itself to most parallel applications. This type of balancing has the advantage of being customized to a particular system it is being run on at any given time. In a dynamically balanced system, if one processor has more computing power than second, the balancing algorithm can assign more of the workload to the first processor, fully taking advantage of the processor with the greater horsepower. Some methods may use past execution results to predict future patterns and adjust accordingly. Other methods may use real time performance characteristics extracted from the system at runtime to adjust the application accordingly. The type of algorithm used to balance the system specifies what balancing mechanisms are used.

One of the obvious disadvantages to dynamic load balancing is the added runtime support required to run the balancing algorithm. Typically, the algorithms monitor, exchange information among processes, calculate workloads and distribute, and in some cases redistribute, the workload [17]. Thus, an algorithm that perfectly balances the workload may have the disadvantage that it itself takes too many cycles to run and effectively takes cycles away from the application. However generally speaking, the added overhead of a properly chosen algorithm can be offset by the improved performance of the dynamically allocated and optimized application.

## 1.3 Neural Network

A neural network is a computational structure inspired by the study of biological neural processing. There are many different types of neural networks, from relatively simple to very complex, just as there are many theories on how biological neural processing works.

### 1.3.1 Biological Neural Systems

The brain is composed of approximately 100 billion ( $10^{11}$ ) neurons. From a computational point of view we also know that the fundamental processing unit of the brain is a neuron [18].

- A neuron consists of a cell body, or soma which contains a nucleus.
- Each neuron has a number of dendrites which receive connections from other neurons.
- Neurons also have an axon which goes out from the neuron and eventually splits into a number of strands to make a connection to other neurons.
- The point at which neurons join other neurons is called a synapse.
- A neuron may connect to as many as 100,000 other neurons.

A simplified view of a neuron is shown in the diagram below.

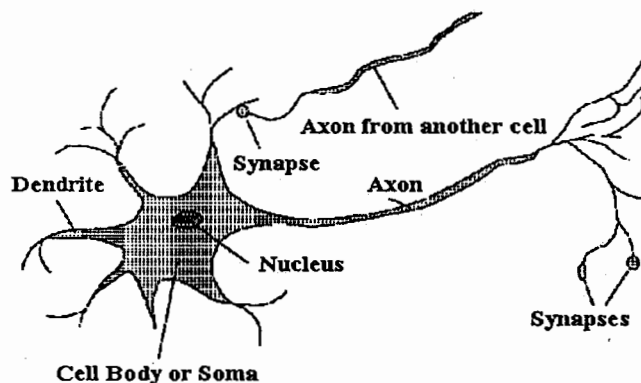


Fig 1.3 Structure of a biological neuron

Signals move from neuron to neuron via electrochemical reactions. The synapses release a chemical transmitter which enters the dendrite. This raises or lowers the electrical potential of the cell body. The soma sums the inputs it receives and once a threshold level is reached an electrical impulse is sent down the axon (often known as firing). These impulses eventually reach synapses and the cycle continues. Synapses which raise the potential within a cell body are called excitatory. Synapses which lower the potential are called inhibitory. It has been found that synapses exhibit plasticity.

### 1.3.2 Artificial Neural Networks

Artificial neural networks are computational paradigms based on mathematical models that unlike traditional computing have a structure and operation that resembles that of the mammal brain. Artificial neural networks or neural networks for short are also called connectionist systems, parallel distributed systems or adaptive systems, because they are composed by a series of interconnected processing elements that operate in parallel. Neural networks lack centralized control in the classical sense, since all the interconnected

processing elements change or “adapt” simultaneously with the flow of information and adaptive rules[18].

### 1.3.2.1 Artificial Neuron

Neuron is the basic building block of the artificial neural network. A neuron is the processing unit, which has more than one input and only one output. First each input  $x_i$  is weighted by a factor  $w_i$  and the whole sum of inputs is calculated  $\sum w_i x_i = a$ . Then an activation function  $f$  is applied to the result  $a$ . The neuronal output is taken to be  $f(a)$ .

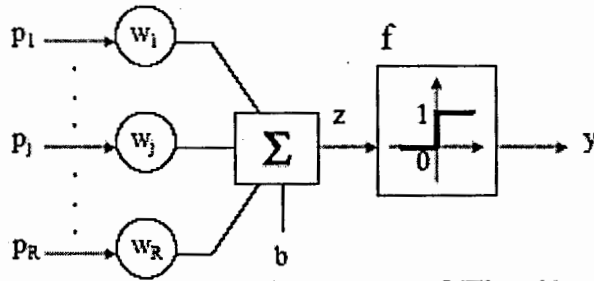


Fig 1.4 Basic Neuron

### 1.3.2.2 Mc Culloch Pitts neuron

McCulloch Pitts (1943) produced the first neural network, which was based on their artificial neuron [19]. Although this work was developed in the early forties, many of the principles can still be seen in the neural networks of today. Important features of McCulloch-Pitts network are as follows

- The activation of a neuron is binary. That is, the neuron either fires (activation of one) or does not fire (activation of zero).  
For the network shown in fig 1.5 the activation function for unit Y is

$$f(y_{in}) = 1, \text{ if } y_{in} \geq t \text{ else } 0$$

where  $y_{in}$  is the total input signal received,  $t$  is the threshold for Y.

- Neurons in a McCulloch-Pitts network are connected by directed, weighted paths.
- If the weight on a path is positive the path is excitatory, otherwise it is inhibitory.
- All excitatory connections into a particular neuron have the same weight, although different weighted connections can be input to different neurons.
- Each neuron has a fixed threshold. If the net input into the neuron is greater than the threshold, the neuron fires.
- The threshold is set such that any non-zero inhibitory input will prevent the neuron from firing.
- It takes one time step for a signal to pass over one connection.

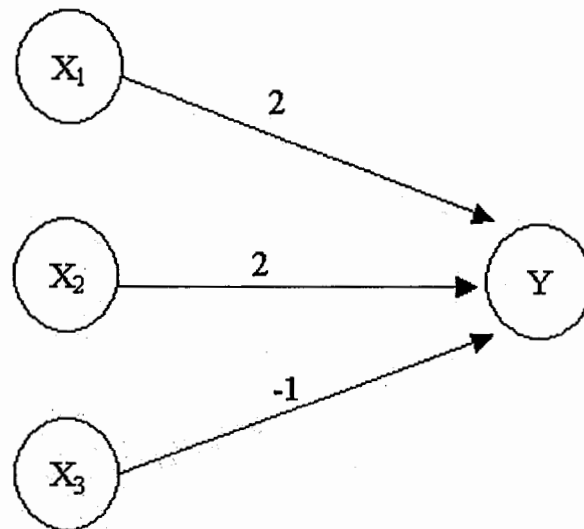


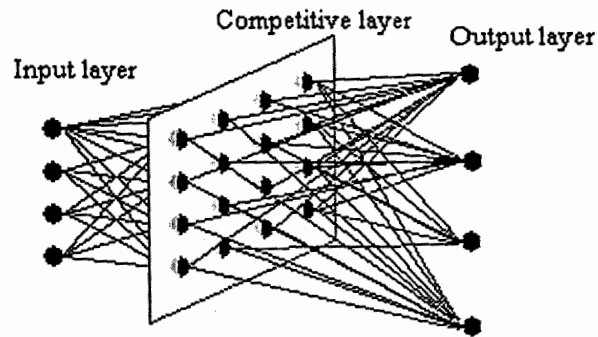
Fig 1.5 McCulloch Pitts Network

#### 1.4 Self-Organizing Map (Kohonen Maps)

A neural network with a capability to learn by itself is called a self-organizing system. The human beings are certainly capable of learning spontaneously, without the benefit of a tutor. In essence, self-organizing systems try to mimic the biologically reasonable systems. One type of a neural network with such capability is a competitive filter associative memory (also known as a Kohonen feature map). This is an example of a self-organizing map, SOM [20]. The unsupervised mode of training is also called self-organized learning because there is no external teacher to preclassify the training examples.

The self-organizing neural networks consist usually of three layers:

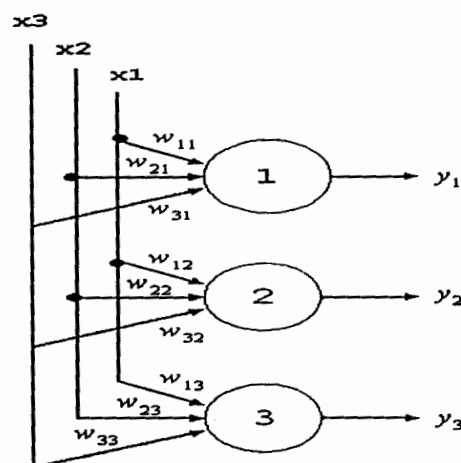
- 1) An input layer that receives the data.
- 2) A competitive layer of neurons that compete with each other to determine to which cluster the given input belongs.
- 3) An output layer which generates the result in a way suitable to the application. Unsupervised training of such networks is carried out with algorithms using competitive training rules. The competitive training rule implements a winner-takes all strategy: it chooses the neuron with the greatest total input as a winner and turns it on, while all other neurons are switched off.



**Fig 1.6 Schematic representation of self organizing maps**

Self-organizing maps are a special class of artificial neural networks based on competitive unsupervised learning. These are networks whose neurons are allocated into one (or two) dimensional lattice structure. During the competitive learning process the neurons are tuned selectively, that is the training data select the winning neurons. Since the locations of the neurons are ordered with respect to each other, the lattice may be considered a kind of a topographic map of the inputs. The locations of the neurons in the topographic map show the statistical features of the provided input data. The neurons in a self-organizing network transform the signals into a corresponding place-coded data distribution [20].

This is a loose simulation of the organization of the cells in the brain, which are assumed to form topologically ordered maps that react to common sensory input signals. Different sensor inputs are mapped into areas of the cerebral cortex in the brain in ordered way. That is why, our intention as computer engineers is to design such computational network devices that perform self-learning following the principle of topographic map formation in the brain. According to this principle the location of a neuron in the lattice reflects a particular feature from the input space.



**Fig 1.7 One-dimensional self-organizing map**

A one-dimensional lattice of fully connected neurons to all input data is shown in the figure 1.7. This is a feed-forward network that trains its synaptic weights adaptively after the arrival of each next input example. The training involves three main phases:

- **Competition:** The neurons in the Kohonen layer compute a certain function, and thus generate outputs that are compared for selection of a winner.
- **Cooperation:** The winner is taken as a basis for cooperation in the sense that it determines the topological neighborhood within which the example falls.
- **Adaptation:** The neurons are adjusted to reflect the information in the provided training example by updating their weights so that the neuron output changes correspondingly.



**CHAPTER 2**  
**LITERATURE SURVEY**

## 2. LITERATURE SURVEY

Foster and Kesselmen(2002) presented OGSA(Open Grid Services Architecture) which defines uniform exposed services semantics(the grid service) and standard mechanism for creating, naming and discovering transient grid service instances, provides transparency for service instances and supports integration with underlying native platform facilities [13].H.Nishikawa(1993) presents load balancing architecture that can deal with applications with heterogeneous tasks [21].However it lacks in several areas e.g. proposed architecture has to be evaluated using more complex applications and larger systems.Additionally Interface of the current system is at low level and user has to specify large no of functions.

P.Mehra et.al(1993) demonstrated automated learning of meaningful load-index functions from workload data [22].Load-balancing systems use workload indices to dynamically schedule jobs. The approach uses comparator neural networks, one per site, which learns to predict the relative speedup of an incoming job using only the resource utilization patterns, observed prior to the job's arrival. The learning algorithm overcomes the lack of job-specific information by learning to compare the relative speedups of different sites with respect to the same job, rather than attempting to predict absolute speedups.

B.S.Siegall et.al(1994) presented an architecture for a system that supports the automatic generation of parallel programs with dynamic load balancing[23].The measurements demonstrate that load balancing overhead can be kept low by proper adjustment of load balancing parameters, and that the load balancer can rapidly adjust the work distribution in a heterogeneous environment. The results also show that techniques that overlap load balancing with computation are effective in reducing load balancing overhead. However the nature of algorithm is best suited for cluster computing rather than load-balancing. M.J.Zaki (1996) examined the behavior of global vs. local and centralized vs. distributed load balancing strategies [24]. In this paper different schemes were analyzed for various applications under varying program and system parameters. A hybrid compile-time and runtime modeling and decision process is presented which customizes the best scheme along with automatic generation of parallel code with calls to runtime library for load balancing. However the method requires use of compilers specific to certain machines and further suffers from the case that nodes themselves cannot act as server thereby reducing scalability.

M.Y.Wu(1995) presented parallel incremental scheduling which is a new approach for load balancing [25].This paper provides an overview of parallel incremental scheduling. It combines the advantages of static scheduling and dynamic scheduling, adapts to dynamic problems and produces high quality load balance. However the presence of central control makes it unsuitable for grid environments.

A.Bevilacqua(1999)introduces a method to obtain efficient load balancing for data parallel applications through dynamic data assignment and a simple priority mechanism on a heterogeneous cluster of workstations assuming no prior knowledge about the workload [26]. This strategy reduces the overhead due to communication so that it could

be successfully employed in other dynamic balancing approaches. While this algorithm is suitable for data centric applications/grids but performance deteriorate rapidly for computation centric applications/grids. S.Ichikawa et.al (2000) describes static load balancing scheme for partial differential equation solvers in a distributed computing environment[27].The method considers both computing and communication time to minimize the total execution time with automatic data partitioning and processor allocation. However, large scale and non-embarrassingly parallel applications are not discussed. A.Osman et.al(2002) presented taxonomy for describing and classifying growing number of different load balancing techniques [28].

C.Ernemann(2002) addresses the potential benefit of sharing jobs between independent sites in a grid computing environment [29].The usage of multi-site applications leads to even better results under the assumption of a limited increase on job execution time due to communication overhead. The results show that the collaboration between sites by exchanging jobs even without multi-site execution significantly improves the average weighted response time.

Jeniffer M Schopf(2002) presented a general architecture for scheduling on grid is presented in [30]. A Grid scheduler (or broker) must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed, and information about the systems is often limited or dated. Furthermore, the idea has not been put to practical use and experimental results limits to well behaved, embarrassingly parallel examples. H.D.Karatza et.al(2003) presented load sharing and job scheduling in a network of workstations (NOW) [31]. Along with traditional methods of load sharing and job scheduling, it also examines methods referred to as epoch load sharing and epoch scheduling respectively. Again the algorithm is not suitable for highly flexible organization such as grid.

H D.Karatza et.al (2002) presented a load sharing is key to the efficient operation of distributed systems [32]. This paper investigates load sharing policies in a heterogeneous distributed system, where half of the total processors have double the speed of the others. Processor performance is examined and compared under a variety of workloads. A priori knowledge of job execution time is not considered in this paper which leads to high overheads for estimating job times. Further for quite a long initial time the algorithm gives errorneous result. C.Juei Yu et.al(2005) presented a prediction-based scheduling algorithm that predicts task execution time and then allocates tasks among workers according to the predictions [33].This leads to high overheads for estimating job times. Further for quite a long initial time the algorithm gives errorneous results. Junwei Cao et.al(2003) presented an agent based grid management infrastructure is coupled with a performance-driven task scheduler that has been developed for local grid load balancing [34].The agents require extensive interference at the client end thus reducing the effectiveness.

Menno et.al(2004) presented the impact of fluctuations in the processing speed on the performance of grid applications [35].Experiments shows that burstiness in the processing speed has a dramatic impact on running times which heightens the need for

dynamic load balancing schemes to realize good performance. Another aspect on which more research has to be done on the aspect of selecting the best predicting methods for processor speeds.

Babak Hamidzadeh et.al(1995) described a general model for describing and evaluating heterogeneous systems that considers the degree of uniformity in the processing elements and the communication channels as a measure of the heterogeneity in the system. The performance of a class of centralized scheduling algorithms referred to as SASH in computing environments with different degrees of heterogeneity is investigated. This approach was compared over DSS in highly heterogeneous system and the work demonstrated that, even with a small no of processors, performing a sophisticated scheduling technique on dedicated processor can produce substantial improvements in total execution times [36].

Aly E. El-Abd et.al(1997) proposed a novel neural based solution to the problem of dynamic load balancing in homogeneous distributed systems [37].The winner-Take-All (WTA) neural network model is used for implementing the selection and location policies of a typical dynamic load balancing algorithm. Again the process is well suited if initial estimations are highly accurate which in uncentralized environment like grid computing is not possible.G.Labonte et.al(1999)presented the implementation of self-organizing map neural networks on distributed parallel computers consisting of identical and of disparate workstations [38]. The implementation is able to reduce the computational time required by SOMs to a fraction of the time, required by a single computer. Issues related to dynamic scheduling have not been discussed.

M.Atun et.al(2000) described a new implementation of Kohonen Self-Organizing Map for static load balancing problem and examines variations of the algorithm [39]. The algorithm preserves neighborhood relations. Load balancing is incorporated into SOM algorithm. Because of the high degree of retention required in building neighborhood trees it is difficult to use the algorithm for larger no of nodes.

Attila Gursoy et.al(2000) implemented static load balancing algorithm based on Self-Organizing Maps (SOM) for a class of parallel computations where the communication pattern exhibits spatial locality [40]. The communication overhead can be reduced if the physically nearby and heavily communicating tasks are mapped to the same processor or to the same group of processors. However issues related to dynamic load balancing has been overlooked.Maris described back propagation neural network based dynamic load balancing algorithm which distributes remote procedure calls among cluster servers, based on statistics about execution time of remote procedure calls. The implementation was used in cluster of Enterprise JavaBeans containers, although the described approach can be used with any type of distributed components that uses synchronous remote invocation protocol, for example, WEB services. Neural network based approach was compared to traditional static algorithms of EJB load balancing. In test cases the presented algorithm produced up to 176% performance increase compared with Round-Robin load balancing policy. However this algorithm is much slower than static load balancing algorithms. Neural network causes significant overhead [41].

## 2.1 Problem Definition

Resource management and scheduling in Grid computing environments is a complex undertaking. Users can literally submit thousands of jobs at a time without knowing - or caring - where they will run. Given a set of processors, when a job arrives a decision maker must decide where it should be served in order to maximize or minimize the given performance measure.

Load balancing can be static or dynamic. With static load balancing, the task and data distribution is determined at the compile time. Static load balancing is useful only to problems that have a rather static workload among the processors through out execution. Although static load balancing can solve many problems (e.g. those caused by processor heterogeneity and non uniform loops) for most regular applications, the transient external load due to multiple users on a network of workstations necessitates a dynamic approach to load balancing. With dynamic load balancing work is assigned to nodes at run time and information about the status of the node and application can be used to optimize the assignment.

Traditional load balancing algorithms make several simplifying assumptions; (i) completion time of a job is not affected by the loads on resources other than the CPU; (ii) some moving average of CPU queue length is a significant determinant of completion time; and (iii) simple decision rules such as always sending a job to the least loaded site, can be determined a priori and perform well in reality.

Load balancing for heterogeneous applications is harder because different tasks have different costs, and data dependencies between the tasks can be very complex. It poses new challenge including dynamic and unpredictable behavior, multiple administrative domains. We must ensure that no local workstation loses access to required local resources by a task allocation from some external workstations.

Due to highly heterogeneous and complex computing environments, effective load balancing is a very difficult problem, even though dynamic load balancing scheme is used. Load balancing is very challenging to achieve. Traditional approaches to the load balancing are either overly conservative or not portable. They require a human designer to specify a formula for computing load, the load index as a function of the current and recent utilization levels of various resources. They also require manual setting of all policy parameters. Not only, there are many parameters, but also they are sensitive to installation-specific characteristics of hardware devices as well as to the prevalent load patterns.

The value of load average cannot guarantee that if a site with lower load average value is given a job, it will complete earlier than another with higher value. They also ignore resources other than the CPU.

Load balancing have been dealt with earlier, but traditional ways require manual setting of parameters and are not efficient as they are not adaptive to the current state of grid. All

these and other drawbacks make it necessary to make use of artificial intelligence and machine learning in load balancing. With increasing demands for high precision autonomous control over wide operating envelopes, conventional control engineering approaches are unable to adequately deal with system complexity, non-linearity, spatial and temporal parameter variations, and with uncertainty.

The resources in the grid are heterogeneous and geographically distributed. Availability, usage and cost policies vary depending on the particular user, time priorities and goals. Load balancing in such an environment is very challenging to achieve. A lot of factors have to be taken into consideration.

**CHAPTER 3**  
**PROPOSED SOLUTION**

### 3. PROPOSED SOLUTION

The proposed solution comes as a solution to the problems mentioned in problem statement. Load balancing is very difficult and challenging task to achieve. We consider load balancing in the following manner: Assume a set of  $n$  parallel heterogeneous nodes ( $N=1,2,3\dots n$ ) and a set of  $m$  independent jobs ( $J=1,2,3\dots m$ ); the jobs arrive one by one, and has to be assigned to exactly one of the nodes, thereby increasing the load on that node. The main objective of load distribution is the division of workload amongst available group of nodes in such a way so that overall completion time of the parallel program is minimized. The workload of a node consists of the combined demands of resources from all of the local processes. The task of load balancing can be viewed as a strategy-learning task, which can be decomposed into learning of load indices and policies. The load indices are used by the policies to make decisions to balance the load. It is hard to find an optimal load balancing solution for a specific application due to rapid changing requirements of application.

Initially each node has a list of jobs to be executed and the time when they have to start. Based on the job's characteristics as well as information about load history at various sites, it is determined where to execute each incoming job. When job reallocation is required, the appropriate jobs will be selected from the job queue on a node and transferred to another node. A job is migrated from one node to another; so its net effects are reduced load on resources local to the originating node and increased load on resources local to the remote node.

We incorporated machine learning and artificial intelligence as a vehicle for automation of load balancing. They have an ability to learn how to do tasks based on the data given for training or initial experience. Since Neural Nets are best at identifying patterns or trends in data. Their ability to learn by examples makes them very flexible and powerful. Either humans or other computer techniques can use neural networks with their remarkable ability to derive meaning from complicated or imprecise data, to extract pattern and detect trends that are too complex to be noticed. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze.

The technique of Neural Networks which we adopted for load balancing in Grid is Kohonen Maps. The Kohonen Map is a Self-Organizing Map of Neural Network meaning that no expected output exists to judge by, and the network finds and reinforces patterns on its own.

#### 3.1 Methodology

Our approach consists of clustering up of nodes and then mapping of tasks to these nodes. Clustering is grouping up of objects that belong together. In our case objects are nodes and clustering criteria is their utilization. The load conditions we assumed are lightly loaded, normal load and heavy loaded nodes. The nodes are grouped up in term of current and past information concerning load state, average memory, status and CPU speed. Each node maintains load information and other parameter log ready to consume whenever load balancing is invoked. The combined effect of load, memory, status and speed is calculated using the following formula.



$$U = ((C1 * \text{Avg.Memory}) + (C2 * \text{Load}) - (C3 * \text{Node status}) - (C4 * \text{CPU speed}))$$

Where,

C1, C2, C3, C4 are weighting constants determining the contribution of each factor.

Avg. memory = Average used memory.

Status = Average uptime of the node.

CPU speed = current CPU speed.

Load (n) = (current CPU utilization / total possible CPU utilization).

Kohonen network is employed for the selection of a set of optimal nodes on to which incoming tasks are to be transferred. Each neuron represents a host in the system. It accepts as input the utilization of each node. During self organization process, the cluster unit whose weight vector matches the input pattern most closely is chosen as a winner. The winner neuron identifies the candidate host to which the utilization is minimum in accordance with the available information about the system. After repetitive step of learning and training the lightly loaded nodes in the network are identified. After that we have the set of optimal nodes, we have to find a mapping, which minimizes the execution time of the job. Tasks are divided into sub-tasks considering task information i.e. (task type, task size). These sub-tasks are mapped to the optimal nodes already calculated by the KNN load balancer.

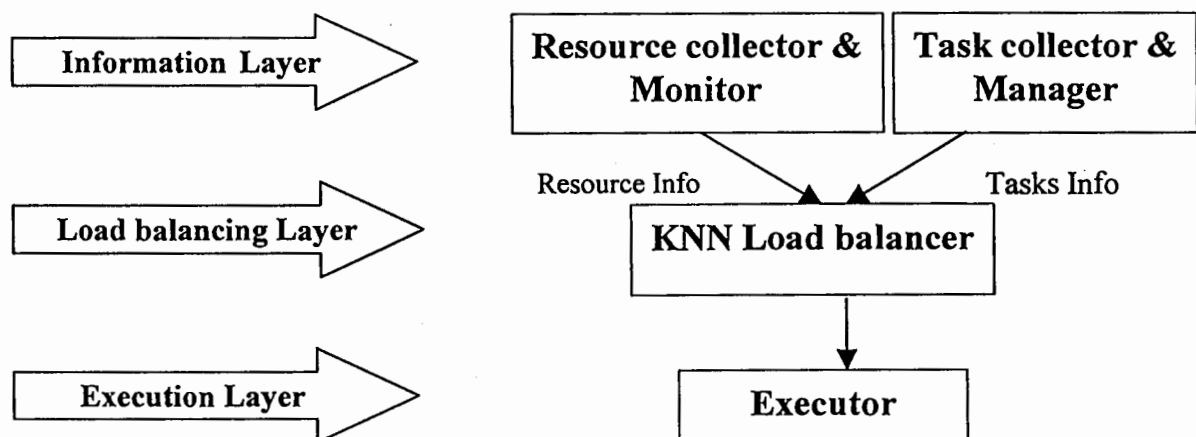


Fig 3.1 Layered Diagram

## 3.2 Functional Modules

Functional modules of the proposed solution are as follows:

### 3.2.1 Resource Collector

Resource Collector is the most significant module. As it functions as a daemon and without this daemon the working of the system is impossible. This module communicates directly with Linux kernel. Its functionalities are as follows:

- Gathers resource information in grid of cluster, particularly the information is related to node id, node speed, node status, used memory etc.
- It creates and save the log file on the basis of accumulated information from each node present in grid.
- Start or stop the Resource Monitor & Analyzer, Task Controller and KNN Load Balancer daemon.
- It parses the generated log file according to the requirements.
- Allows process migration on each node.

### 3.2.2 Resource Monitor

It examines the resource information accumulated by Resource collector, displays it, arrange the information in user readable form. The information is updated after the time period specified by user.

### 3.2.3 Resource Analyzer

It display load statistics, read the log file generated by resource collector and generate dynamic graphs for load, memory, time and status. The graphs are updated after regular time period.

### 3.2.4 KNN Load Balancer

It is important module of our system.KNN algorithm is working here. Its functioning is as follows:

- Read the parsed log file with respect to the parameters required by KNN-Learning. In short it collects the offline information of resources.
- Initialize learning rate and neighborhood parameters.
- Initialize neurons (nodes) with random values.
- Euclidean distance of input neurons is calculated to find the winner. Weights of the neuron are updated and the process is repeated to find another winner and all winner nodes are clustered up in the form of optimized nodes.
- Cumulative weight load matrix is also calculated from node to node.
- Perform task rendering and provide output results.
- Forward the task and optimal nodes information to Task-Mapping Engine, Process migration is also performed from this engine.

### 3.2.5 Task Collector

- It acknowledges tasks from external environment and keep them in a queue and writes task information in log file which includes task size, type, starting time, end time.

- Parses the log file to get required task information.

### 3.2.6 Task Manager

It gives interface to user for communication with our grid. Handles user requests for load task, edit task, save task, create task and remove task.

### 3.2.7 Task Monitor

- a) It examines all tasks present in queue and gives the status for each individual task, whether it is running, finished, or about to finish.
- b) It compares task completion time with its approximate weighted factor in order to get the efficiency.

### 3.3.8 Performance Monitor

- a) It extracts resource information from resource monitor and task information from task monitor.
- b) In case of load imbalance pass the current extracted information to KNN Load balancer in order to balance the load on grid.

**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. It is basically the design of the information processing system covering the activities of determining detailed requirements, design of data/information flow, design of database, design of user interface, physical design, and design of hardware/software configuration.

### 4.1 Object-Oriented Analysis and Design

Object-Oriented analysis is the discovery, analysis and specification of requirements in terms of objects with identity that encapsulate properties and operations, message passing, classes, inheritance, polymorphism and dynamic binding. It aims to model the problem domain the problem to be solved by developing object oriented system. Object-oriented design is the design of an application in terms of objects, classes, clusters, frameworks and their interactions.

#### 4.1.1 Class Diagram

In the unified modeling language (UML) a class diagram depicts the static view of the model or part of the model, describing what attributes and behavior it has rather than detailing the methods for achieving operations. Another purpose of class diagrams is to specify the class relationships and the attributes and behaviors associated with each class.

A class is an element that defines the attributes and behaviour that an object is able to generate. The behaviour is described by the possible messages which the class is able to understand, along with operations that are appropriate for each message. Classes may have definitions of constraints, tagged values and stereotypes. Class diagram of our system is shown in fig 4.1.

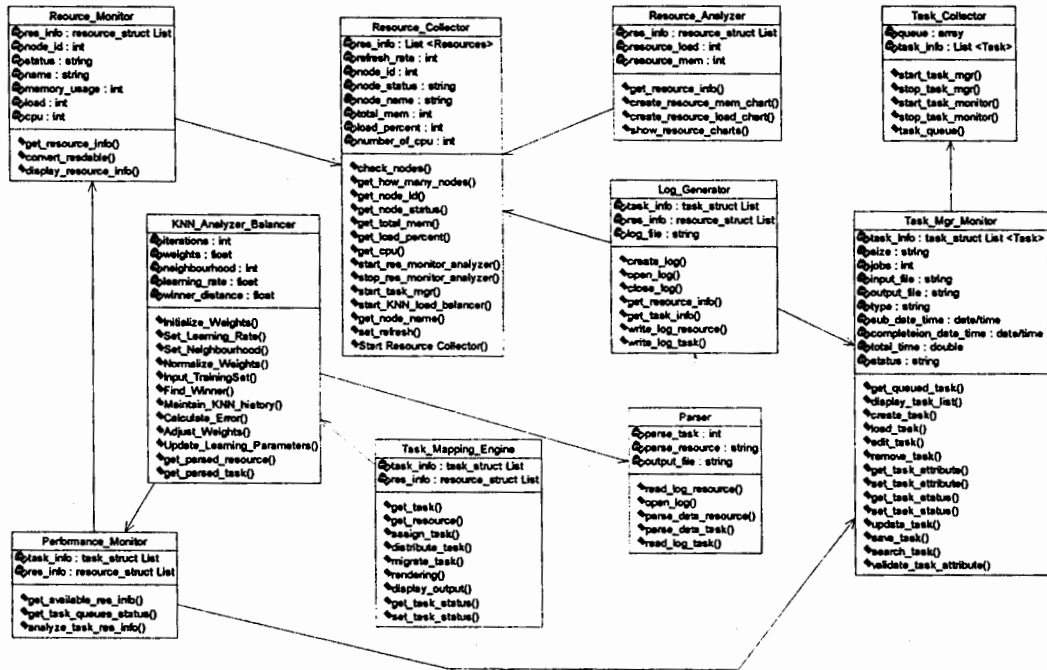


Fig 4.1 Class Diagram

## 4.1.2 Sequence Diagram

A Sequence diagram depicts the sequence of actions that occur in a system. The invocation of methods in each object, and the order in which the invocation occurs is captured in a Sequence diagram. It depicts the dynamic behavior of system. Because it uses class name and object name references, the Sequence diagram is very useful in elaborating and detailing the dynamic design and the sequence and origin of invocation of objects.

### 4.1.2.1 Collect Resource Information

In this sequence diagram the sequence of actions is as user sends request to resource collector to start. Resource collector fulfils the request and sends request for log generation to log generator. Then request for write resource information is sent to log generator. After log file generation log generator sends request for parsing log file to parser.

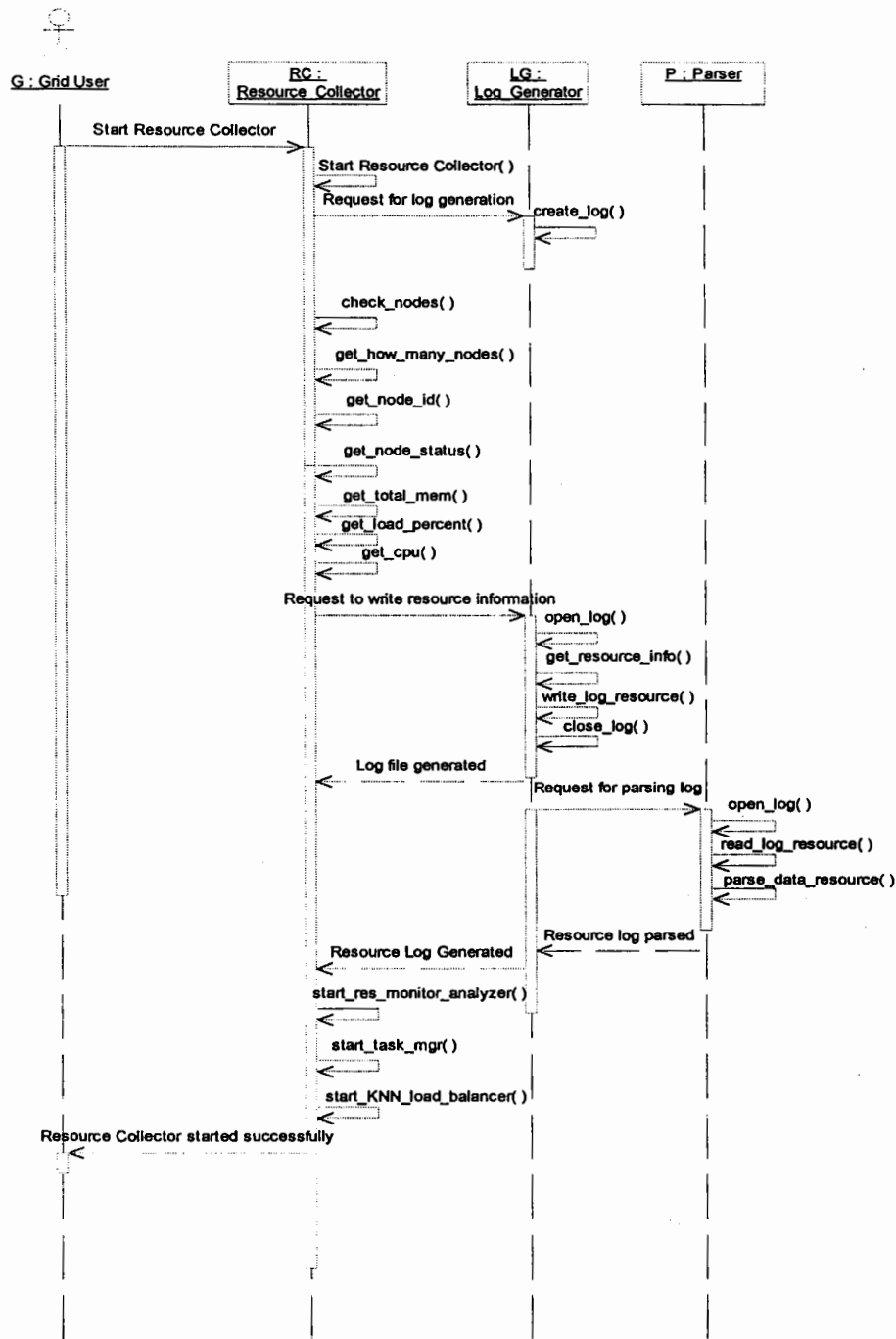
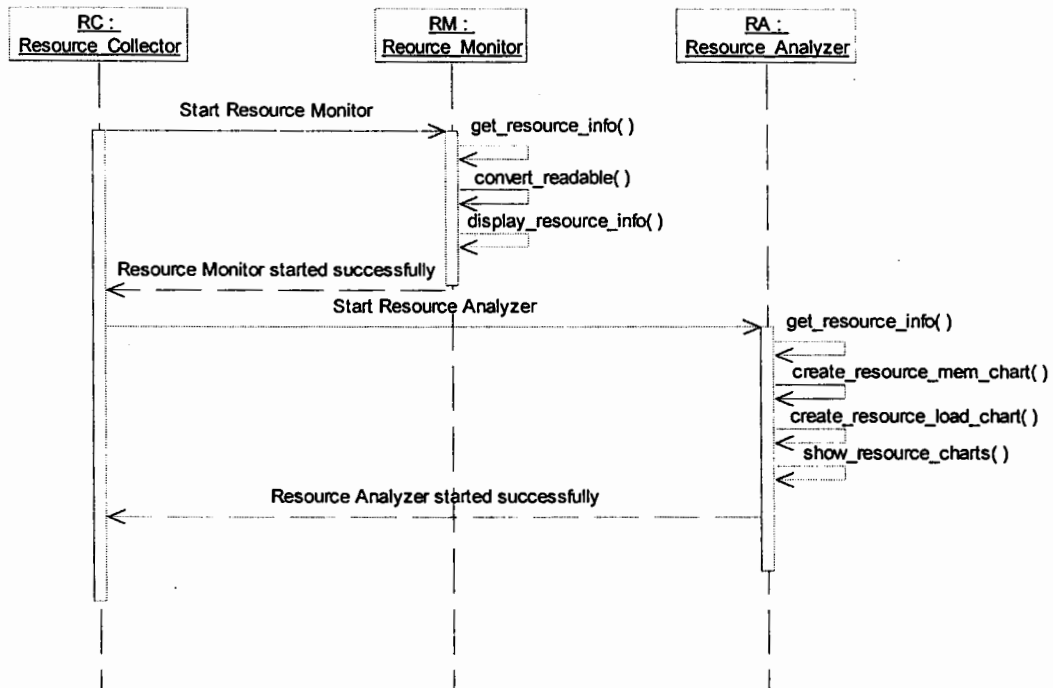


Fig 4.2 Collect Resource Information

### 4.1.2.2 Analyze and Monitor Resource Information

In Fig 4.3 resource collector sends the request to start resource monitor. A request is made from resource collector to start resource analyzer. Resource monitor and analyzer are started successfully.



**Fig 4.3 Analyze and Monitor Resource Information**



### 4.1.2.3 Load balancing and Re-balancing

In Fig 4.4 request for load balancing and rebalancing is send from performance monitor to KNN Analyzer-Balancer. Parsed data is sent from parser to KNN Load balancer. After performing certain steps load balancing is performed successfully.

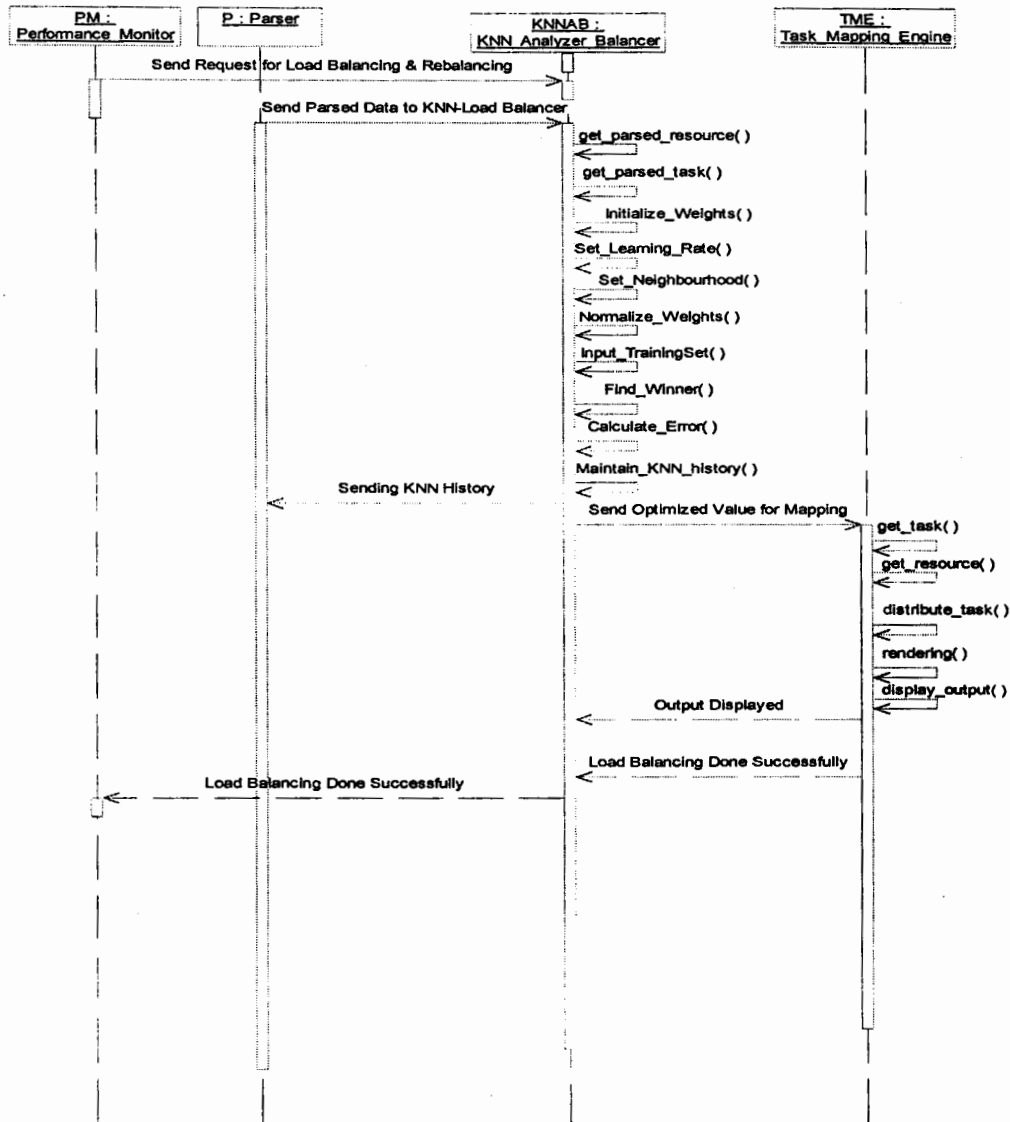


Fig 4.4 Load balancing and Re-Balancing

#### 4.1.2.4 Performance Monitor

In Fig 4.5 request from resource information and task information is sent to performance monitor.

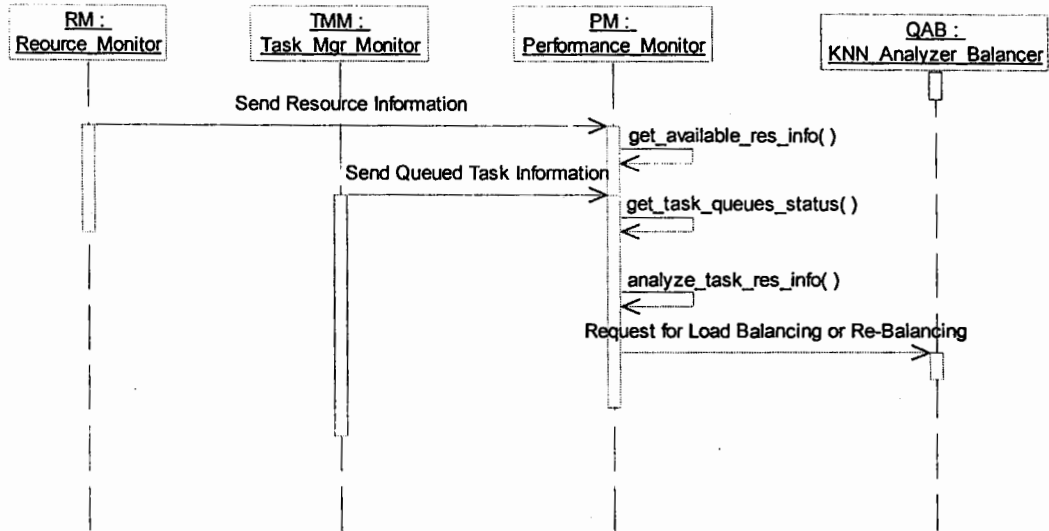


Fig 4.5 Performance Monitor

### 4.1.2.5 Task Scheduling

In fig 4.6 Parser sends the request to KNN load balancer for scheduling tasks. After performing the functions optimized value is sent to task mapping engine. And scheduling is done successfully.

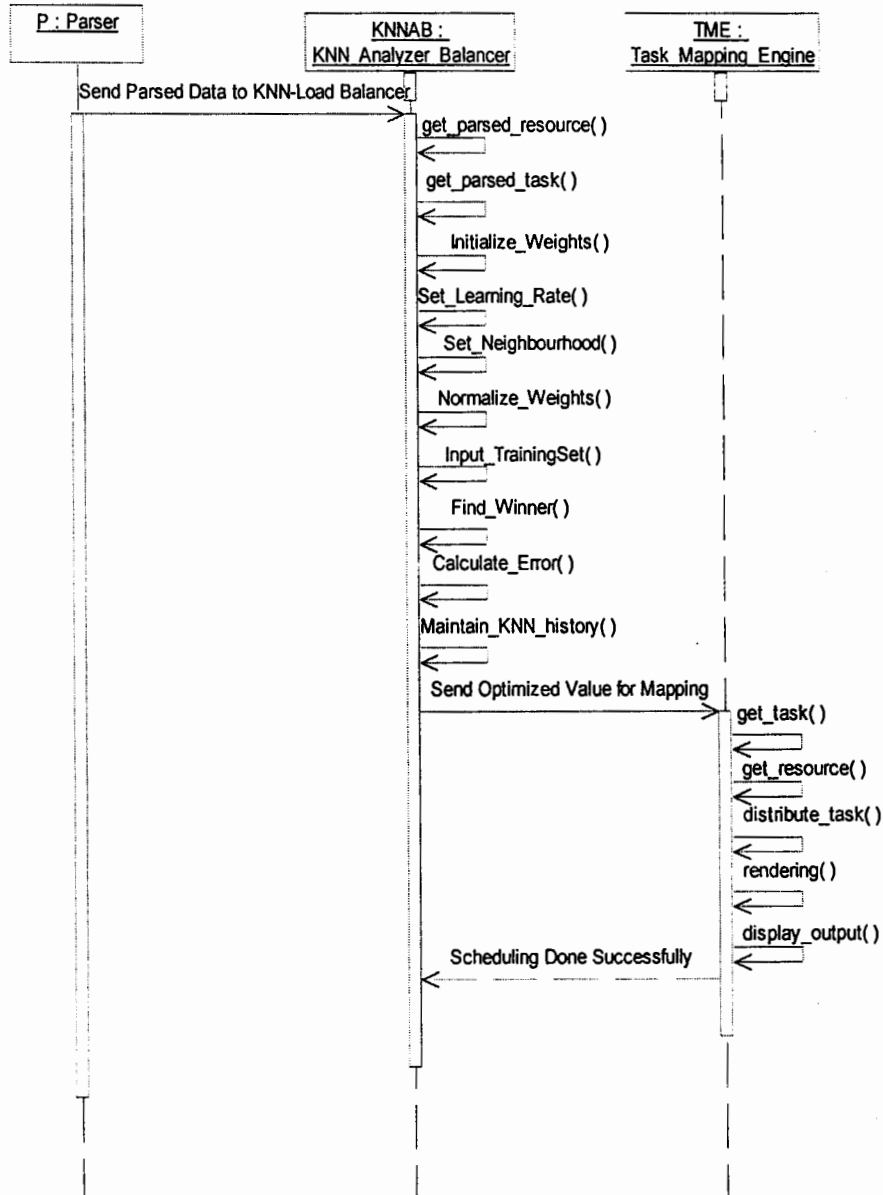


Fig 4.6 Scheduling Tasks

#### 4.1.2.6 Task Log Generation

In fig 4.7 request for task log generation is sent from task monitor to log generator. Request for parsing is sent from log generator to parser. Log is parsed and generated successfully.

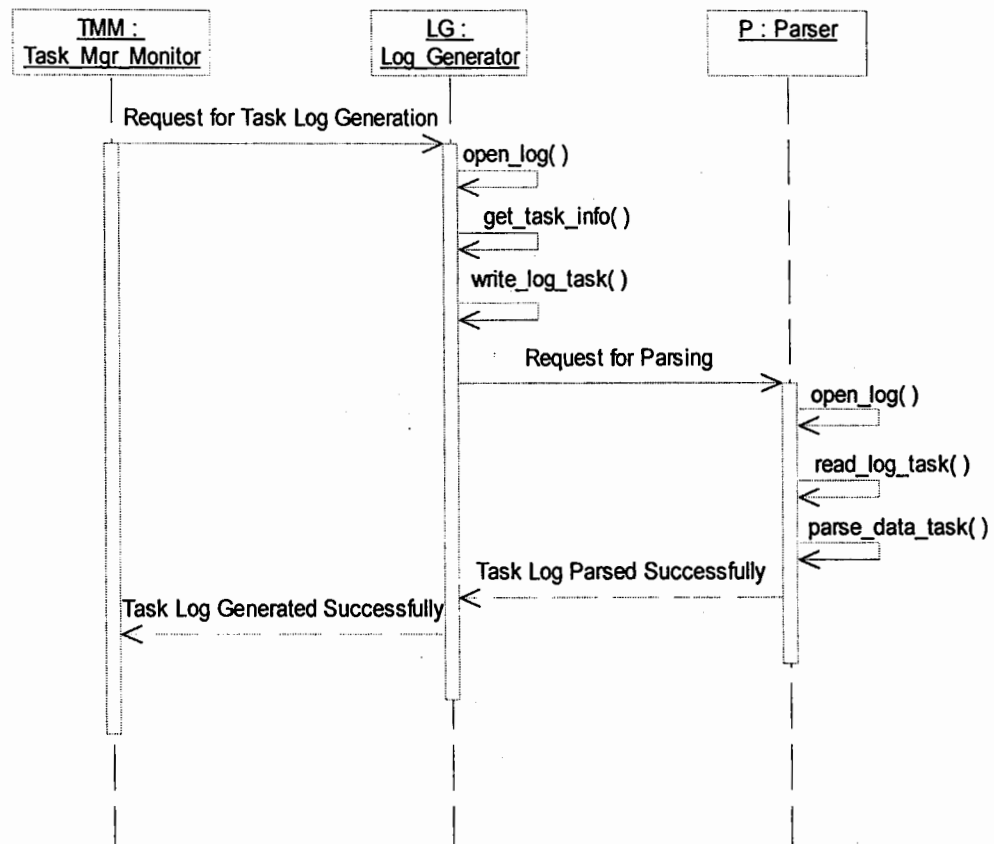


Fig 4.7 Task Log Generation

### 4.2 System Architecture

System Architecture is the design or set of relations between the parts of a system. It is the most important, pervasive, top level, decisions, and their associated rationale about the overall structure.

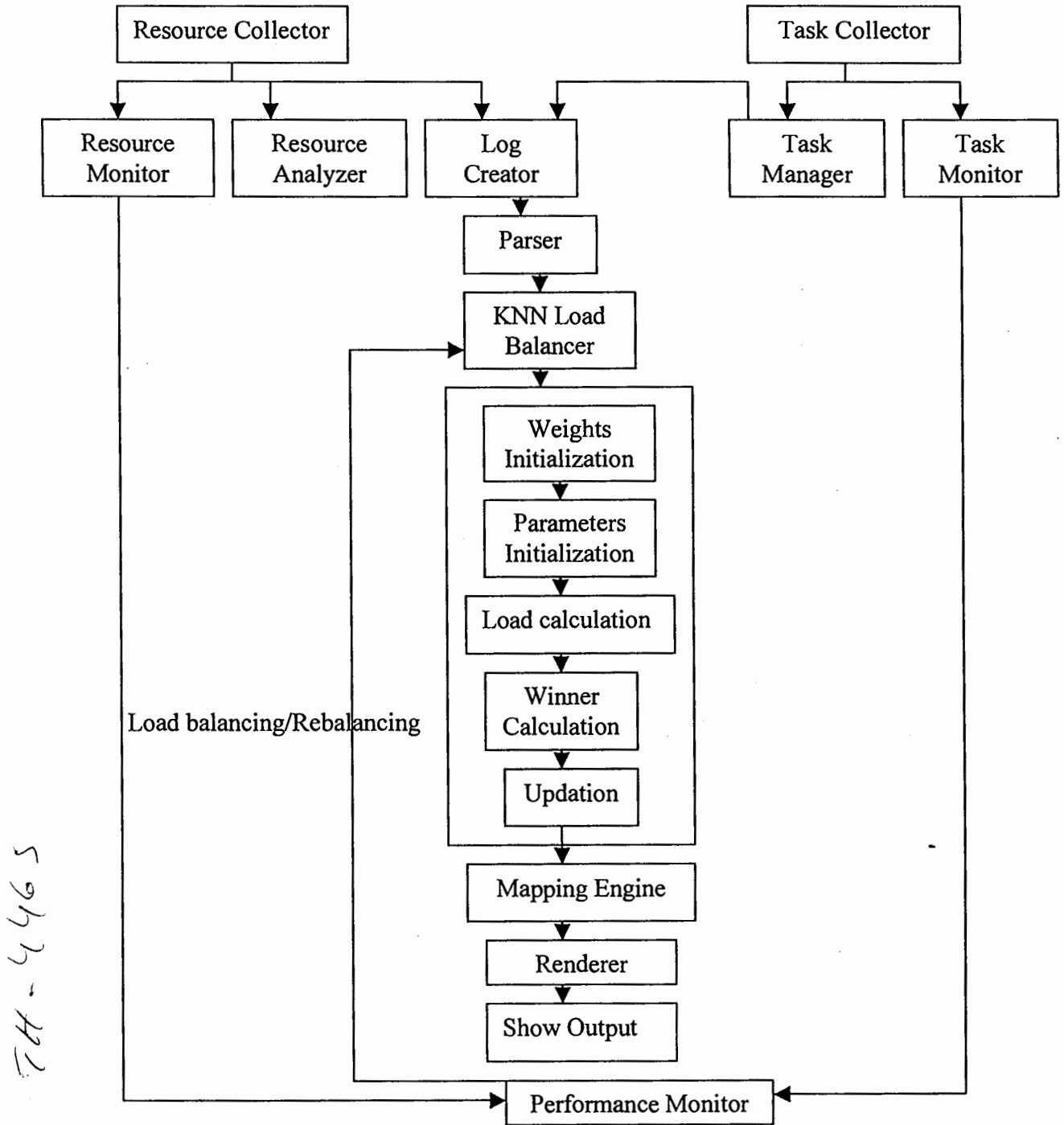


Fig 4.8 Architecture diagram

**Resource collector** is a daemon which directly communicates with Linux kernel to collect resource information in grid. **Resource monitor** examines and arrange this resource information. **Resource analyzer** displays resource information. **Log generator** generates log or history file of each node and executed tasks. Tasks from the external environment are handled by **Task Collector**. User's requests for the execution or termination of task are handled by **Task manager**. Task monitor keeps on monitoring the task and give status information to each task for performance evaluation. **KNN loadbalancer** starts with the initialization phase in which connection weights, learning rate and neighborhood parameters are initialized. Kohonen layer is constructed based on current and past information of the system resources. After repetitive steps of training a set of underloaded nodes is the output. The **Mapping engine** maps the incoming task to the less loaded nodes calculated by the load balancer. **Performance monitor** keep on monitoring the task and resource information and signal the load imbalance to load balancer.

**CHAPTER 5**  
**SYSTEM DEVELOPMENT**

## 5. System Development

System Development is the phase in which we transfer the proposed system into an executable software. It is the execution, or practice of a plan, a method, or any design for doing something. Implementation is the actual writing of the code. If the design has been done correctly and with efficient detail then coding becomes a simple task. This step involves making the final design decisions and translating the design diagrams and specifications into the syntax of the chosen programming language. It also involves the practical development process, to interactively compile, link and debug components.

### 5.1 Tools

Tools play an important role in the implementation of a system. A programming tool is a program or application that software developers use to create, debug, or maintain other programs and applications. These tools can be divided into different categories which are as:

#### 5.1.1 Languages

A high-level programming language that is interpreted by another program at runtime rather than compiled by the computer's processor as other programming languages (such as C and C++) are. As C++ is an object oriented programming (OOP) language so it provides all the facilities of OOP. The languages used in the project are C/C++.

#### 5.1.2 Editors

Editors are used for designing the interface and for editing any application. The editors used in the project are as

- KWrite(KDE Environment)
- gedit
- Qt designer(Qt is a mature cross-platform GUI toolkit written in C++).

#### 5.1.3 Office tools

Some of the office tools used in this project are 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 Benchmarking tools

Benchmarking is identifying the highest standard of excellence, learning and understanding those standards, and finally adapting and applying them to improve performance. Benchmark



is the result of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, by running a number of standard tests and trials against it. The benchmarking tool we used is POV-Ray™. POV-Ray™ is short for the Persistence of Vision™ Raytracer, a tool for producing high-quality computer graphics. POV-Ray for Unix is essentially a version of the POV-Ray rendering engine prepared for running on a Unix or Unix-like operating system (such as GNU/Linux).

### 5.1.5 Grid Management tools

Grid management tools are used to configure and use grid environment which will be comprised of individual nodes interconnected and sharing computational resources. We patched openmosix to Linux kernel. OpenMosix is a kernel extension for single-system image clustering.

## 5.2 Pseudocode

Pseudo code is a language that uses the vocabulary of one language (i.e. English) and the overall syntax of another (i.e. structured program language). It is free syntax of natural language that describes processing features.

### 5.2.1 Resource Collector

Resource Collector is the most significant module. The main steps carried out in this module are as follows

1. Read linux proc file and gathers resource information in grid of cluster, particularly the information is related to node id, node speed, node status, used memory, load percent etc.
2. It creates and save the log file on the basis of accumulated information from each node present in grid.
3. Set cluster nodes according to configuration.
4. Start or stop the Resource Examiner & Evaluator, Task Controller and KNN Load balancer daemon.
5. It parses the generated log file according to the requirements.
6. Allows process migration on each node.

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 Evaluator

1. It display load, memory statistics.
2. Read the log file generated by resource collector and generate dynamic graphs for load, memory, time and status.

3. The graphs are updated after regular time period.
4. It analyzes load and memory of all nodes.
5. It gives information about all nodes.

### 5.2.3 KNN Load Balancer

It is important module of our system.KNN algorithm is working here. Its functioning is as follows:

1. Read the parsed log file with respect to the parameters required by KNN- Learning. In short it collects the offline information of resources.
2. Initialize learning rate and neighborhood and other parameters.
3. Initialize neurons (nodes) with random values.
4. Euclidean distance of input neurons is calculated to find the winner.
5. Weights of the neuron are updated and the process is repeated to find another winner and all winner nodes are grouped up in the form of optimized nodes.
6. The nodes are grouped up in term of current and past information concerning load state, avg.memory, status and CPU speed. Each node maintains load information and other parameter log ready to consume whenever load balancing is invoked.
7. Each neuron represents a host in the system. It accepts as input the utilization of each node. During self organization process, the cluster unit whose weight vector matches the input pattern most closely is chosen as a winner.
8. The winner neuron identifies the candidate host to which the utilization is minimum in accordance with the available information about the system.
9. Cumulative weight load matrix is also calculated from node to node.
10. Forward the task and optimal nodes information to Task-Mapping Engine.
11. Perform task rendering and provide output results.

### 5.2.4 Task collector

1. It gathers tasks from outside environment and store them in a queue and writes task information in log file which includes status, size, starting time, end time, completion time.
2. User submits the task by providing task input file, output file, type and size.
3. Parses the log file to get required task information.

## **CHAPTER 6**

### **RESULTS AND CONCLUSION**

## 6. RESULTS AND CONCLUSION

In this chapter we discuss the experiments performed to validate the proposed solution and the graphs generated from these experiments. The main objective of our thesis is load balancing in heterogeneous environment using Artificial Neural Network. The performance of our approach is proved by executing and testing our application with varying no of processors, task size, no of tasks. The experimental setup on which the experiments were performed is described as follows.

### 6.1 Experimental Setup

Our Grid system is a grid of cluster with 6 nodes of different specifications with Linux Operating System installed on them interconnected via Ethernet LAN. As a fundamental base we have adapted openmosix as the process/task monitoring and management tool. Openmosix is used as a Linux Kernel patch for grid configuration. Povray is used as benchmarking tool.

### 6.2 Description of Experiments

To prove the authenticity of our approach we performed the experiments with the case when there is No scheduling i.e. no prior scheduling is performed when a job is submitted. Secondly when a scheduler is in place and compared to other dynamic algorithms. We conducted experiments with trained and not trained neural network. Different experiments were performed with varying no processors, task sizes, no of processors. Input tasks of different sizes were taken from Povray benchmarking tool.

### 6.3 Experimental Results

The experiments were conducted on a Linux operating system kernel patched with OpenMosix which is used as a fundamental base for grid. For comparison purpose we are using Static Scheduling, Dynamic Scheduling, and No Scheduling. Povray is used as benchmarking to observe the optimized performance of our system. The execution time is used as a performance metric to analyze the performance of our ANN based grid application. Graphs of different cases were generated from these experiments and which are as follows.

In fig 1 we see that the difference falls off very rapidly and KNN based soon outperforms the no scheduler approach. The rapid decrease with increased task frequency is done due to the fact that variable CPU time is available and in case of no scheduling the inability to migrate the job to lesser loaded node increases job execution time.

In fig 2 initially we see that KNN takes longer time to complete the job but as no of tasks increases the difference begins to fall off. This is because KNN consumes lot of time for training. However for large no of tasks this slowly goes down and KNN based gain in performance. A stage is reached at which time KNN actually outperform the No Scheduler

case. This is because larger tasks inherently mean larger execution time thereby increasing the probability of node being busy for extended period of time.

In fig3 KNN (With Training) is compared with other dynamic algorithms and we see that execution time starts decreasing, as the no of processors increases thus improving the performance as compared to others.

In fig 4 initially we see that execution time of KNN is higher but as no of processors increases the difference begins to fall off. And its performance is better as compared to other dynamic algorithms. This is because KNN consumes lot of time for training.

In fig 5 and fig 6 the general trend is same but in fig 6 we see that KNN takes less time in the beginning as it is trained but as no of tasks increases, KNN takes more time to complete the job as the no of processors is also fixed. But the performance of KNN is still better than other dynamic algorithms. KNN incurs some delay as no of tasks increases due to context switching between the processors.

Fig 7 and fig 8 shows that the general trend is same as before except that in fig 7 we see that KNN outperforms static scheduler after some time. We also see that the difference is much smaller in the beginning because static scheduler also takes some time for their own calculation.

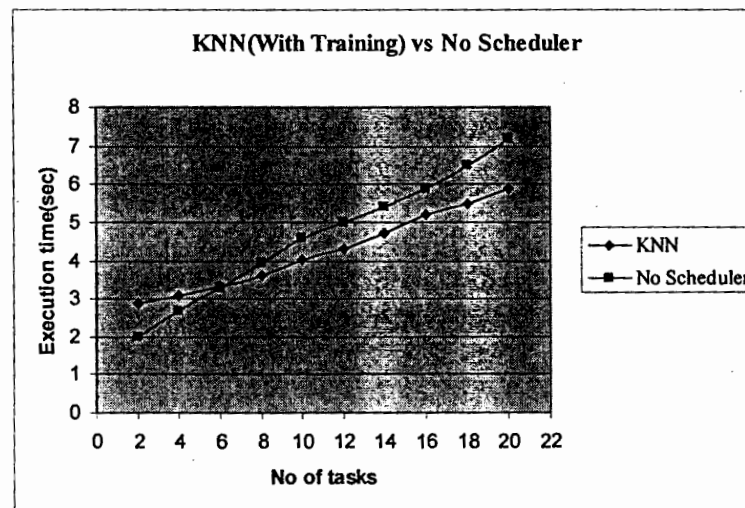


Fig 1 KNN (With Training) vs No Scheduler

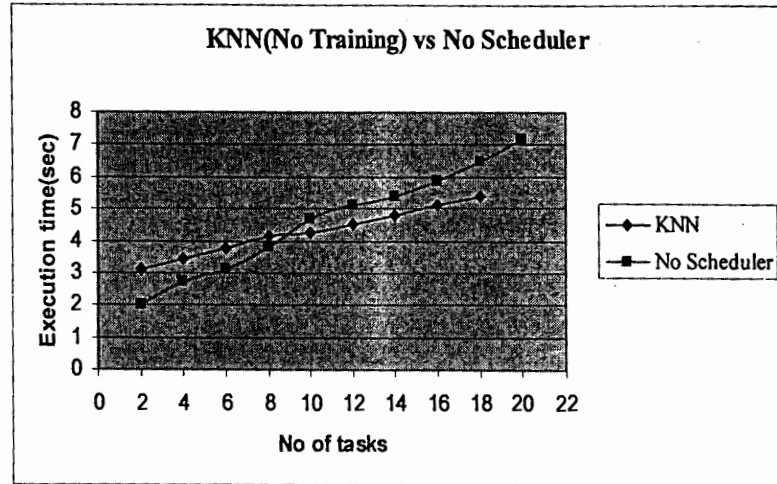


Fig 2 KNN (No Training)vs. No Scheduler

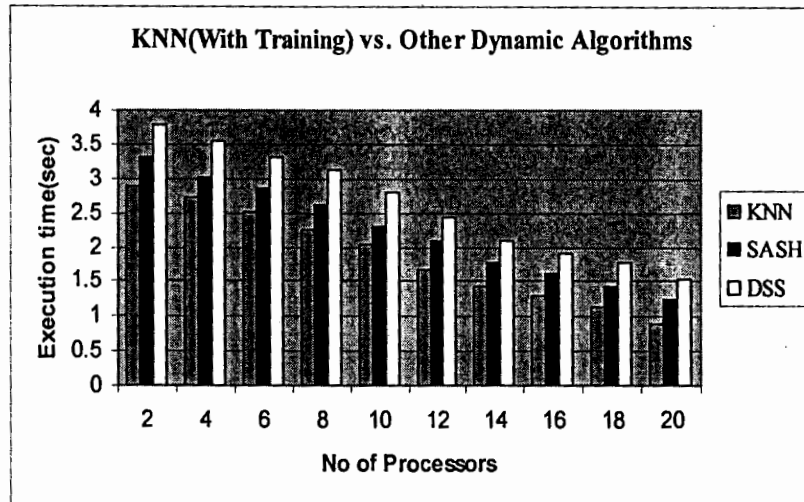


Fig 3 KNN(With Training) vs Other Dynamic Algorithms

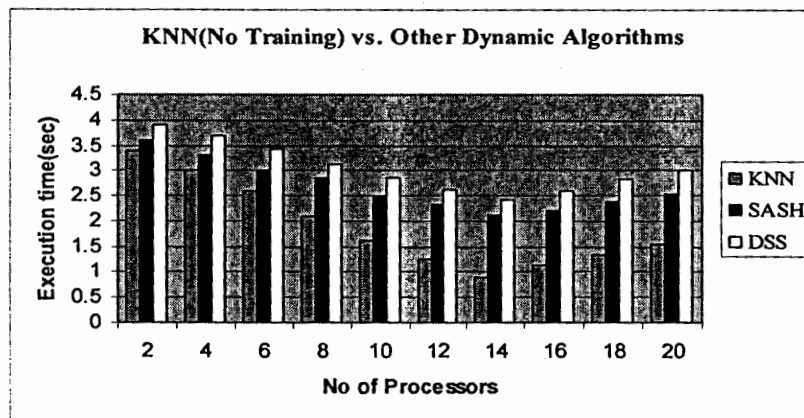


Fig 4 KNN (No Training) vs. other dynamic Algorithms

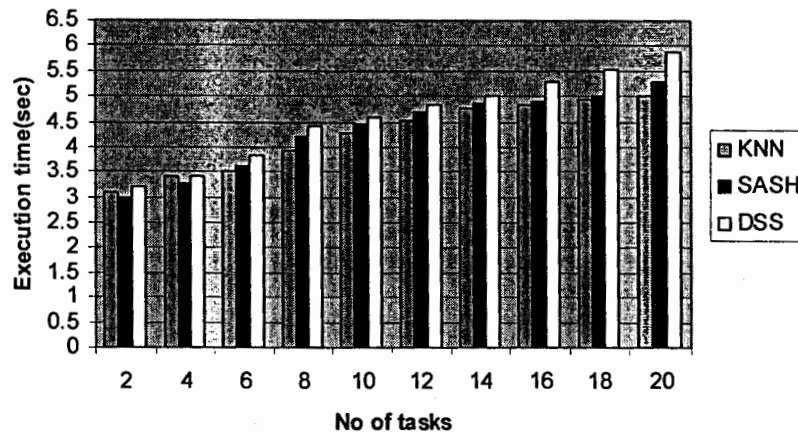


Fig 5 KNN (No Training) vs. other dynamic Algorithms w.r.t No of tasks

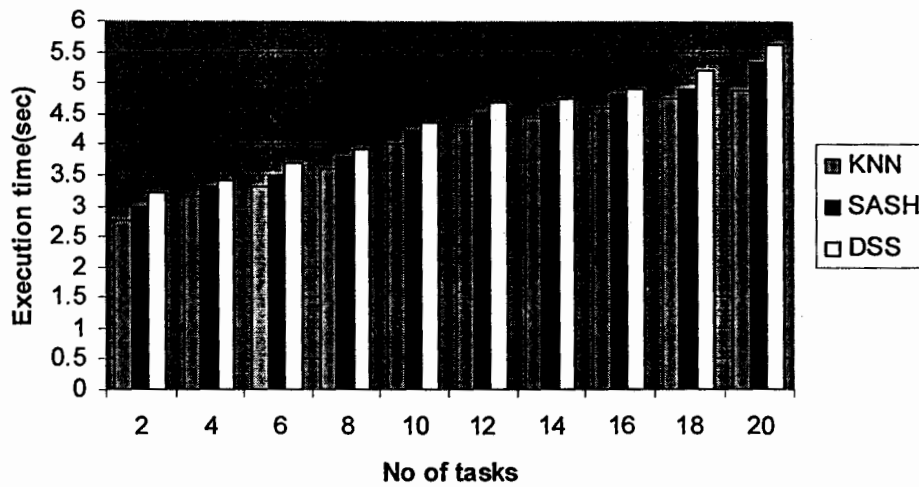


Fig 6 KNN (With Training) vs Other Dynamic Algorithms w.r.t No of tasks

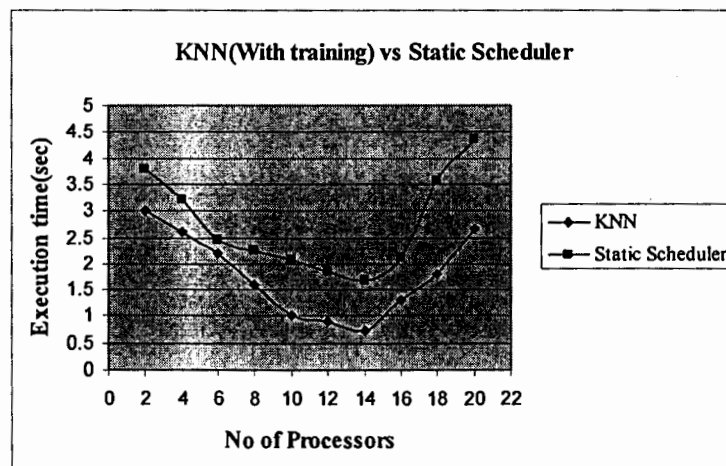


Fig 7 KNN (With Training) vs Static Scheduler

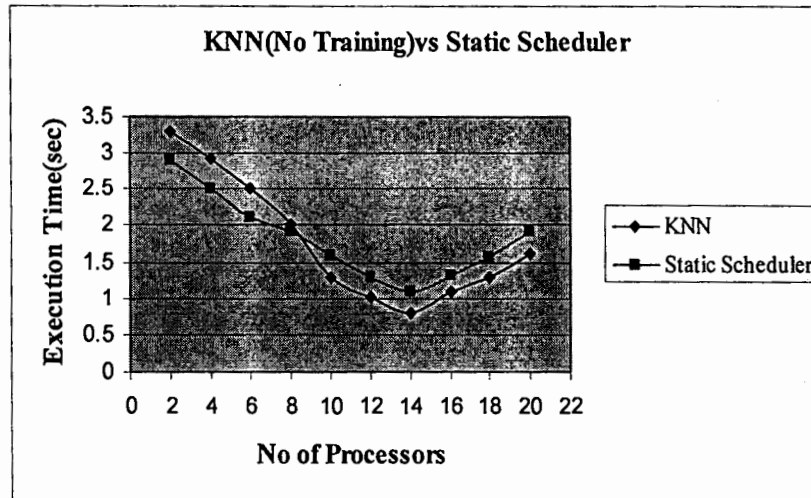


Fig 8 KNN (No Training) vs Static Scheduler

## 6.4 Conclusion

In this research we presented a load balancing technique based on artificial neural network in heterogeneous environment for computationally intensive applications. Resource management and scheduling in heterogeneous environment is challenging task to achieve. Load balancing is done in such a way so as to minimize job execution time. We incorporated SOM also known as Kohonen Networks a technique of ANN for automation of load balancing. As the results shows the performance of Kohonen networks with respect to other dynamic algorithms and the case where scheduler is in place. The results showed that the performance was analyzed for large no of tasks, processors in response to execution time. And we can see the fluctuation in execution time with varying no of processors, task sizes, no of tasks etc. As Kohonen takes much time for training but once it is trained it gives better performance in comparison with other dynamic Schedulers and Static Scheduler.



## REFERENCES AND BIBLIOGRAPHY

## REFERENCES AND BIBLIOGRAPHY

- [1] Henri Casanova. Distributed computing research issues in grid computing. SIGACT News, v.33 n.3, pp.50-70,ACM Press, Sept 2002.
- [2] Anderson, E., Brooks, J., Grassl, C., and Scott, S. (1997) Performance of the CRAY T3E multiprocessor, Proceedings of the 1997 ACM/IEEE conference on Supercomputing, 1-17, San Jose, CA, ISBN-0897-9198-58.
- [3] A.Grama,V.Kumar, A.Gupta and G.Karypis. (2003) An Introduction to Parallel Computing: Design and Analysis of Algorithms, Second Edition, Pearson Addison Wesley, ISBN-0-2016-4865-2.
- [4] <http://www.grid/Interviews/rbuyya.htm>
- [5] Ian. Foster and C. Kesselman, editors;“The Grid: Blueprint for a Future Computing Infrastructure”Morgan Kaufmann Publishers, 1999.
- [6] <http://www.teragrid.org/eot/glossary.html>
- [7] M.Litzkow, and M. Livny, “Experience with the Condor Distributed Batch System”; In proceedings of IEEE Workshop on Experimental Distributed Systems,pp 97-101, Huntsville, AL, USA, Oct 1990.
- [8] R. Buyya, D. Abramson, J. Giddy, Nimrod-G: an architecture for a resource management and scheduling system in a global computational grid, in: Proceedings of the HPC ASIA'2000, China, IEEE CS Press, USA, 2000.
- [9] I.Foster, C.Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2):115-128, 1997.
- [10] Steve J. Chapin , Dimitrios Katramatos , John F. Karpovich , Andrew S. Grimshaw, The Legion Resource Management System, Proceedings of the Job Scheduling Strategies for Parallel Processing, p.162-178, April 16, 1999
- [11] [http://en.wikipedia.org/wiki/Grid\\_computing](http://en.wikipedia.org/wiki/Grid_computing)
- [12] <http://www.research.ibm.com/journal/sj/434/joseph.html>
- [13] Ian Foster,C.Kesselman, J.M. Nick, S.Tuecke; “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”; Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
- [14] <http://www.cacs.louisiana.edu/~mgr/404/burks/foldoc/76/67.htm>
- [15] [http://meseec.ce.rit.edu/eccc756-spring2000/project99/load\\_bal.pdf](http://meseec.ce.rit.edu/eccc756-spring2000/project99/load_bal.pdf)

- [16] Chongbing Liu, "Dynamic Load Balancing in Parallel and Distributed Computation: A survey", <http://www.cs.nmsu.edu/~cliu/>
- [17] Pankaj Mehra, Benjamin Wah; "Automated Learning Of workload Measures For Load Balancing On A Distributed System"; Proc. of the International Conference on Parallel Processing, CRC Press, vol. III, pp. 263-270, Aug. 1993.
- [18] [http://www.doc.ic.ac.uk/~nd/surprise\\_96/journal/vol4/cs11/report.html](http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html)
- [19] <http://www.cs.nott.ac.uk/~gxk/courses/g5ai/006neuralnetworks/neural-networks.htm>
- [20] <http://homepages.gold.ac.uk/nikolaev/311koh.htm>
- [21] Hiroshi Nishikawa, Peter Steenkiste; "A general Architecture for load balancing in a distributed memory environment"; Proc. of 13<sup>th</sup> Intl. Conf. Dist. Computing Systems, pp. 47-54. IEEE, Pittsburgh May 1993.
- [22] Pankaj Mehra, Benjamin Wah; "Automated Learning Of workload Measures For Load Balancing On A Distributed System"; Proc. of the International Conference on Parallel Processing, CRC Press, vol. III, pp. 263-270, Aug. 1993..
- [23] B.S. Siegel, P. Steenkiste; "Automatic generation of parallel programs with dynamic load balancing"; Proc. of the Third IEEE International Symposium on High Performance Distributed Computing; San Francisco, CA, 1994.
- [24] M. Javeed Zaki, W. Li, S. Parthasarathy; "Customized dynamic load balancing for a network of workstations"; Proc. 5th IEEE Int. Symposium on High performance distributed computing, Syracuse, New York, August 1996.
- [25] M. You Wu; "Parallel Incremental Scheduling"; Parallel Processing Letters, vol 5, No(4):659-670, 1995.
- [26] A. Bevilacqua; "A Dynamic Load Balancing Method on a Heterogeneous Cluster of Workstations"; Informatica (Slovenia), 23(1); March 1999.
- [27] S. Ichikawa, S. Yamashita; "Static Load Balancing of Parallel PDE Solver for Distributed Computing Environment" Proc. ISCA 13th Int'l Conf. Parallel and Distributed Computing Systems (PDCS2000); pp. 399--405 (2000).
- [28] A. Osman, H. Ammar; "Dynamic load balancing strategies for parallel computers"; Scientific Annals of Cuza Univ; vol 11: 110-120, 2002.
- [29] Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Ramin Yahyapour; "On Advantages of Grid Computing for Parallel Job Scheduling"; 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), 2002.

- [30] Jeniffer M Schopf, "A general Architecture for scheduling on the grid", special issue of Journal of Parallel and Distributed Computing on grid computing, 2002.
- [31] Helen D. Karatza,"A Comparison of Load Sharing and Job Scheduling in a Network of Workstations"; Int'l journal of Simulation: Systems, Science Technology, UK Simulation Society, Volume 4, no: 3&4, pp. 4-11, 2003.
- [32] Helen D. Karatza, Ralph C. Hilzer "Load Sharing in heterogeneous distributed systems", Proceedings of the 2002 Winter Simulation Conference (WSC'02) - Volume 1, 2002.
- [33] Cheng-Juei Yu , Po-Han Chen, and Sheng-De Wang ; "Adaptive Task Scheduling Algorithms for Master-Worker Applications in Grid Computing"; Eleventh Workshop on Compiler Techniques for High-Performance Computing,2005.
- [34] Junwei Cao<sup>1</sup>, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd; "Agent-Based Grid Load Balancing Using Performance-driven Task Scheduling"; Proc.of 17th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2003); Nice, France; April 2003.
- [35] Menno Dobber, Ger Koole, Rob van der Mei; "Dynamic Load Balancing for Grid Applications";11<sup>th</sup> Int'l Conference, Proceedings HiPC;Bangalore,India, dec19-22,2004.
- [36] Babak Hamidzadeh, David J. Lilja, Yacine Atif, "Dynamic scheduling Techniques for Heterogeneous Computing Systems", Concurrency: practice & Experience, Vol. 7, No. 7,pp. 633-652,October 1995.
- [37] Aly E. EI-Abd and Mohamed I. EI-Bendary, "A Neural Network Approach for Dynamic Load Balancing In Homogeneous Distributed Systems", 30th Hawaii International Conference on System Sciences (HICSS) Volume 1: Software Technology and Architecture, 1997.
- [38] G. Labonté and M. Quintin; "Network Parallel Computing for SOM Neural Networks",To appear in the proceedings of the High Performance Computing Symposium 1999.
- [39] Murat Atun and Attila Gürsoy, "A new load-balancing algorithm using self-organizingmaps";Euro-Par2000,ParallelProcessing.6thInternationalEuro-ParConference Germany,2000.
- [40] Attila Gürsoy, Murat Atun, "Neighborhood Preserving Load Balancing: A Self-Organizing Approach", Euro-Par Parallel Processing, LNCS 1900, pp. 324-41, 2000.
- [41] Maris Orbidans, "A Neural Network based dynamic load balancing Algorithm", <http://maris.site.lv/loadbalancing/>

**APPENDIX-A**

## A. A NOVEL ANN-BASED LOAD BALANCING TECHNIQUE FOR HETEROGENEOUS ENVIRONMENT

Muniza Salim, Ammara Manzoor, Khalid Rashid  
Dept. of Computer Science, Faculty of Basic and Applied Sciences,  
International Islamic University, Islamabad, Pakistan.

### Abstract

Grid computing is an emerging computing paradigm and is distinguished from distributed computing by its efficient and optimal utilization of heterogeneous, loosely coupled resources tied to work load management. However, complexity incurred in efficient management of heterogeneous, geographically distributed and dynamically available resources has become one of the most challenging issues in grid computing. A lot of parameters have to be taken into consideration to efficiently utilize the grid resources. Many heuristics has been proposed in the literature to address this complex problem. Our research aims to solve load balancing decisions using Artificial Neural Networks (ANN). Since ANN are best at identifying patterns or trends in data, their ability to learn by examples makes them very flexible and powerful. In this research we have developed and evaluated a completely new scheduling-cum-load balancing module for a scaleable grid. Experimental results suggest that once trained, ANN outperforms other heuristic approaches for large tasks. However for small tasks, ANN suffers from extensive overheads.

**Keywords:** Grid Computing, Load balancing, Artificial Neural Networks

### INTRODUCTION

Advances in networking infrastructure and ever changing computational requirements have led to the development of grid infrastructure that provides a way to meet the needs of these demanding applications. Grid provides predictable, consistent and uniform access to geographically distributed resources like computers, data repositories, scientific instruments, and advanced display devices (Foster and Kesselman, 1999).

Grids are often considered as the successors of distributed computing. Grid computing is the more general case of distributed computing which enables collaborative multi-site computation (Foster et al., 2001). Grid integrates and coordinates resources and users that live within different control domain as they lack central control (Foster, 2002). Moreover the Computational capabilities of a Grid vary significantly over time as the resources are added and removed dynamically. OGSA (Open Grid Services Architecture) which defines uniform exposed services semantics (the grid service), defines standard mechanism for creating, naming and discovering transient grid service instances, provides transparency for service instances and supports integration with underlying native platform facilities. (Foster and Kesselmen, 2002).

The objective of load balancing is to minimize the response time and execution time of a

program by trying to equally spreading the load on processors and maximizing their utilization. The goal of grid task scheduling is to achieve high system throughput and to meet the application needs within available computing resources.

Kohonen network or Self-Organizing Map (SOM) is a type of neural network that works with unsupervised training. It takes input data and trains itself. Kohonen based his neural network on the associative neural properties of the brain (Kohonen, 2001). Self-Organizing Map is called a topology-preserving map because there is a topological structure imposed on the nodes in the network. Self-organizing behavior is the spontaneous formation of well-organized structures, patterns or behaviors, from random initial conditions.

In this paper we address load balancing heuristic based on Self-organizing neural network which allows adaptive and reliable management and scheduling of tasks. Load balancing is accomplished by scheduling of the tasks.

The basic description of problem domain is as follows: Resource management and scheduling in Grid computing environment is a very complex task. Thousands of jobs are submitted by users without knowing where they will execute. When a job arrives, the decision maker must decide where it should be served on a given set of processors in order to maximize or minimize the given performance measure. Load balancing can be static or dynamic. Static load balancing determines the

task and data distribution at compile time. It is useful only to problems that have predictable resource requirements and load variations. Although it can solve many problems i.e. those caused by processor heterogeneity and non uniform loops for most regular applications, the transient external load due to multiple users on a network of workstations necessitates a dynamic approach to load balancing. With dynamic load balancing work is assigned to nodes at run time and information about the status of the node and application can be used to optimize the task.

Traditional load balancing algorithms make several simplifying assumptions; (i) completion time of a job is not affected by the loads on resources other than the CPU; (ii) some moving average of CPU queue length is a significant determinant of completion time; and (iii) simple decision rules such as always sending a job to the least loaded site, can be determined a priori and perform well in reality. Load balancing for heterogeneous applications is difficult because different tasks have different costs, and data dependencies between the tasks can be very complex. It creates new challenge including dynamic and unpredictable behavior, multiple administrative domains. Due to highly heterogeneous and complex computing environments, effective load balancing is a very difficult problem even though if dynamic load balancing is used, load balancing is very challenging to achieve. Load imbalance is another serious problem in which load from heavily loaded nodes is migrated to lightly loaded ones by the use of efficient load balancing. Traditional approaches to the load balancing are either overly conservative or not portable as they require a human designer to specify a formula for computing load, the load index as a function of the current and recent utilization levels of various resources. They also require manual setting of all policy parameters and are not efficient as they are not adaptive to the current state of grid. They are also sensitive to installation-specific characteristics of hardware devices as well as to the prevalent load patterns. Load average value cannot guarantee that if a site with lower load average value is given a job, it will complete earlier than another with higher value. They also ignore resources other than the CPU. All these and other drawbacks make it necessary to make use of artificial intelligence and machine learning in load balancing.

The literature surveyed for analysis of problem domain is as follows: Load balancing architecture that can deal with applications with heterogeneous tasks is presented by Nishikawa and Steenkiste (1993). However it lacks in several areas e.g. the architecture has to be evaluated using more

complex applications and larger systems. Additionally Interface of the current system is at low level and user has to specify large no of functions.

Mehra and Wah(1993) demonstrated automated learning of meaningful load-index functions from workload data. The approach uses comparator neural networks, one per site, which learns to predict the relative speedup of an incoming job using only the resource utilization patterns, observed prior to the job's arrival. The learning algorithm overcomes the lack of job-specific information by learning to compare the relative speedups of different sites with respect to the same job, rather than attempting to predict absolute speedups.

Siegal and Steenkiste(1994) presented an architecture for a system that supports the automatic generation of parallel programs with dynamic load balancing. The measurements demonstrate that load balancing overhead can be kept low by proper adjustment of load balancing parameters, and that the load balancer can rapidly adjust the work distribution in a heterogeneous environment. The results also showed that techniques that overlap load balancing with computation are effective in reducing load balancing overhead. However the nature of algorithm is best suited for cluster computing rather than load-balancing.

Zaki et al (1997) examined the behavior of global vs. local and centralized vs. distributed load balancing strategies. Different schemes were analyzed for various applications under varying program and system parameters. A hybrid compile-time and runtime modeling and decision process is presented which customizes the best scheme along with automatic generation of parallel code with calls to runtime library for load balancing. However the method requires use of compilers specific to certain machines and further suffers from the case that nodes themselves cannot act as server thereby reducing scalability.

Parallel incremental scheduling, a new approach for load balancing is presented by M.Y.Wu (1995). It combines the advantages of static scheduling and dynamic scheduling, adapts to dynamic problems and produces high quality load balance. However the presence of central control makes it unsuitable for grid environments.

Bevilacqua (1999) introduces a method to obtain efficient load balancing for data parallel applications through dynamic data assignment and a simple priority mechanism on a heterogeneous cluster of workstations assuming no prior knowledge about the workload. This strategy reduces the overhead due to communication so that it could be successfully employed in other dynamic

balancing approaches. While this algorithm is suitable for data centric applications/grids but performance deteriorate rapidly for computation centric applications/grids.

S.Ichikawa et al. (2000) describes static load balancing scheme for partial differential equation solvers in a distributed computing environment. This method considers both computing and communication time to minimize the total execution time with automatic data partitioning and processor allocation. However, large scale and non-embarrassingly parallel applications are not discussed. The taxonomy for describing and classifying growing number of different load balancing techniques is presented by Osman (2002). The potential benefit of sharing jobs between independent sites in a grid computing environment is discussed by Ernemann (2002). In this paper the usage of multi-site applications leads to even better results under the assumption of a limited increase on job execution time due to communication overhead. The results showed that the collaboration between sites by exchanging jobs even without multi-site execution significantly improves the average weighted response time. Schopf(2002) presented a general architecture for scheduling on grid. A Grid scheduler (or broker) must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed, and information about the systems is often limited or dated. Furthermore, the idea has not been put to practical use and experimental results limits to well behaved, embarrassingly parallel examples.

Karatza (2003) et al. presented load sharing and job scheduling in a network of workstations (NOW). Along with traditional methods of load sharing and job scheduling, it also examines methods referred to as epoch load sharing and epoch scheduling respectively. Again the algorithm is not suitable for highly flexible organization such as grid.

Load sharing policies in a heterogeneous distributed system is investigated by Karatza (2002), where half of the total processors have double the speed of the others. Processor performance is examined and compared under a variety of workloads. A priori knowledge of job execution time is not considered in this paper which leads to high overheads for estimating job times. Further for quite a long initial time the algorithm gives errorneous result. Cheng.JueiYu et al. (2005) presented a prediction-based scheduling algorithm that predicts task execution time and then allocates tasks among workers according to the predictions. This leads to high overheads for estimating job times. Further for quite a long initial time the algorithm gives errorneous results.

Cao et al. (2003) presented an agent based grid management infrastructure is coupled with a performance-driven task scheduler that has been developed for local grid load balancing. The agents require extensive interference at the client end thus reducing the effectiveness. The impact of fluctuations in the processing speed on the performance of grid applications is presented by Dobber et al. (2004). Experiments shows that burstiness in the processing speed has a dramatic impact on running times which heightens the need for dynamic load balancing schemes to realize good performance. Another aspect on which more research has to be done on the aspect of selecting the best predicting methods for processor speeds. Babak Hamidzadeh et al. (1995) described a general model for describing and evaluating heterogeneous systems that considers the degree of uniformity in the processing elements and the communication channels as a measure of the heterogeneity in the system. The performance of a class of centralized scheduling algorithms referred to as SASH in computing environments with different degrees of heterogeneity is investigated. This approach was compared over DSS in highly heterogeneous system and the work demonstrated that, even with a small no of processors, performing a sophisticated scheduling technique on dedicated processor can produce substantial improvements in total execution times. Aly E. El-Abd(1997) et al. proposed a novel neural based solution to the problem of dynamic load balancing in homogeneous distributed systems. The winner-Take-All (WTA) neural network model is used for implementing the selection and location policies of a typical dynamic load balancing algorithm. Again the process is well suited if initial estimations are highly accurate which in uncentralized environment like grid computing is not possible.

Labonte (1999) presented the implementation of self-organizing map neural networks on distributed parallel computers consisting of identical and of disparate workstations. The implementation is able to reduce the computational time required by SOMs to a fraction of the time required by a single computer. Issues related to dynamic scheduling have not been discussed. Atun (2000) described a new implementation of Kohonen Self-Organizing Map for static load balancing problem and examines variations of the algorithm. The algorithm preserves neighborhood relations. Load balancing is incorporated into SOM algorithm. Because of the high degree of retention required in building neighborhood trees it is difficult to use the algorithm for larger no of nodes.

Gursoy (2000) implemented static load balancing algorithm based on Self-Organizing Maps (SOM) for a class of parallel computations where the



communication pattern exhibits spatial locality. The communication overhead can be reduced if the physically nearby and heavily communicating tasks are mapped to the same processor or to the same group of processors. However issues related to dynamic load balancing has been overlooked. Maris described back propagation neural network based dynamic load balancing algorithm which distributes remote procedure calls among cluster servers, based on statistics about execution time of remote procedure calls. The implementation was used in cluster of Enterprise JavaBeans containers, although the described approach can be used with any type of distributed components that uses synchronous remote invocation protocol, for example, WEB services. Neural network based approach was compared to traditional static algorithms of EJB load balancing. In test cases the presented algorithm produced up to 176% performance increase compared with Round-Robin load balancing policy. However this algorithm is much slower than static load balancing algorithms. Neural network causes significant overhead.

## MATERIALS AND METHODS

The methodology comes as a solution to the problems mentioned in problem domain. Load balancing is very difficult and challenging task to achieve. We consider load balancing in the following manner: Assume a set of  $n$  parallel heterogeneous ( $N=1,2,3...n$ ) and a set of  $m$  independent jobs ( $J=1,2,3..m$ ); the jobs arrive one by one, where each job has an associated load vector and has to be assigned to exactly one of the nodes, thereby increasing the load on that node. The main objective of load distribution is the division of workload amongst available group of nodes in such a way so that overall completion time of the parallel program is minimized. The workload of a node consists of the combined demands of resources from all of the local processes. The task of load balancing can be viewed as a strategy-learning task, which can be decomposed into learning of load indices and policies. The load indices are used by the policies to make decisions to balance the load. It is hard to find an optimal solution to achieve load balancing for a specific application due to dynamic nature and non-reliability of computational environment.

Initially each node has a list of jobs to be executed and the time when they have to start. Based on the job's characteristics as well as information about load history at various sites, it is determined where to execute each incoming job. When job reallocation is required, the appropriate jobs will be selected from the job queue on a node and transferred to another node. A job is migrated

from one node to another; so its net effects are reduced load on resources local to the originating node and increased load on resources local to the remote node.

We incorporated machine learning and artificial intelligence as a vehicle for automation of load balancing. They have an ability to learn how to do tasks based on the data given for training or initial experience. Since Neural Nets are best at identifying patterns or trends in data. Their ability to learn by examples makes them very flexible and powerful. Either humans or other computer techniques can use neural networks with their remarkable ability to derive meaning from complicated or imprecise data, to extract pattern and detect trends that are too complex to be noticed. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze.

Neural networks with their remarkable ability to derive meaning from complicated or imprecise data can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze (Chris Stergiou, 1996).

The technique of Neural Networks which we adopted for load balancing in Grid is Kohonen Maps. The Kohonen Map is a Self-Organizing Map of Neural Network meaning that no expected output exists to judge by, and the network finds and reinforces patterns on its own.

## Functional Modules

Functional modules of the system are as follows:

### Resource Collector

Resource Collector is the most significant module of our grid. As it acts as a daemon and without this daemon the working of the system is impossible. Its functionalities are as follows:

- Gathers resource information in grid.
- It creates and save the log file on the basis of accumulated information from each node present in grid.
- Start or stop the Resource Monitor & Analyzer, Task Controller and KNN Load Balancer daemon.
- It parses the generated log file according to the conditions.

### Resource Monitor

It checks the resource information accumulated by Resource Collector, displays it,

arrange this information in user readable form. The information is updated after the specified time period given by user.

### Resource Analyzer

It display load statistics, read the log file generated by Resource Collector and generate dynamic graphs for load, memory, time and status. The graphs are updated after regular time period.

### Knn Load Balancer

This is another important module of our grid. The features of KNN Load Balancer are as follows:

- Read the parsed log file with respect to the parameters required by KNN-Learning. In short it collects the offline information of resources.
- Initialize learning rate and neighborhood parameters.
- Initialize neurons (nodes) with random values.
- Euclidean distance of input neurons is calculated to find the winner. Weights of the neuron are updated and the process is repeated to find another winner.
- Perform task rendering and provide output results.
- Forward the task and optimal nodes information to Task-Mapping Engine, Process migration is also performed from this engine.

### Task Controller

It has three sub modules.

#### Task Collector

It acknowledges tasks from external environment and keep them in a queue and writes task information in log file. It parses the log file to get required task information.

#### Task Administrator

It gives interface to user for communication with our grid. Handles user requests for load task, edit task, save task, create task and remove task.

#### Task Monitor

- It examines all tasks present in queue and gives the status for each individual task, whether it is running, finished, or about to finish.

- It matches task completion time with its approximate weighted factor in order to get the efficiency.

### Performance Monitor

- It extracts resource information from resource monitor and task information from task monitor.
- In case of load imbalance pass the current extracted information to KNN Load Balancer in order to balance the load on grid.

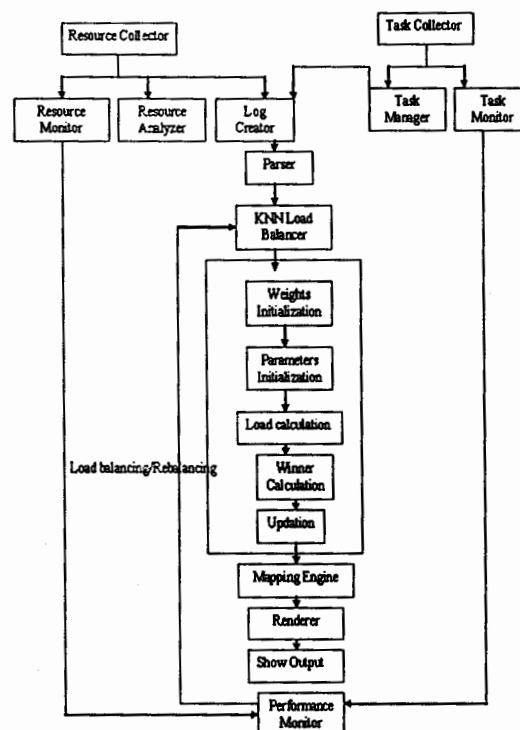


Figure 1: Architecture diagram

### ALGORITHM

**Step 1:** Initialize size of network, no of iterations.

**Step 2:** Set initial and final values of learning rate  $\alpha$  and neighborhood radius  $\theta$  respectively.

**Step 3:** Initialize the weight vectors  $w_{ij}$  of the nodes randomly.

**Step 4:** Calculate the load of each node by using the formula

Load=current CPU utilization/total possible CPU utilization

**Step 5:** While ( $t \neq t_{max}$ ) do

- Select lightly loaded node from the nodes.

- Select random input vector X from the input space S.
- Compute the Euclidean distance of each neuron j by using the formula
 
$$D(j) = \sum_i (W_{ij} - x_i)^2$$
- Find Index j such that D(j) is minimum.
- Update weight vectors of nodes by using the formula

$$W_{ij}(\text{new}) = W_{ij}(\text{old}) + \alpha [X_i - W_{ij}(\text{old})]$$

Step 6: Update learning rate.

Step 7: Reduce Neighborhood Radius.

Step 8: Update load of each node.

Step 9: Repeat from step 5 until max iterations is not reached.

### EXPERIMENTAL SETUP

Our system consists of 100 nodes with Linux Operating system on them interconnected via Ethernet LAN. As a fundamental base we have adapted openmosix as the process/task monitoring and management tool. We benchmark two computationally intensive applications which are matrix multiplication and PovRay. For the sake of comparison we compare our results with the case when dynamic scheduler is in place. We also compare our results with static scheduler.

### RESULTS

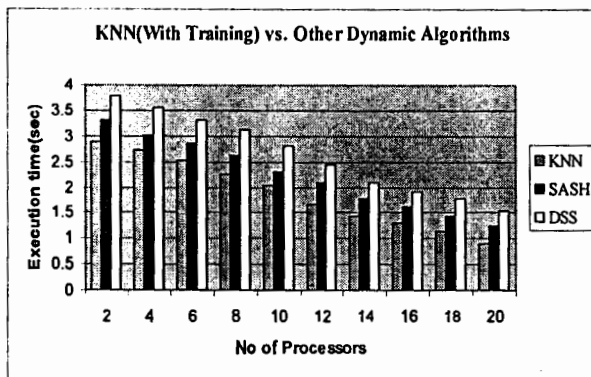


Fig 2: KNN (With Training) vs. Other Dynamic Algorithms

In the above graph KNN is compared with other dynamic algorithms and we see that execution time starts decreasing, as the no of processors increases thus improving the performance as compared to others.

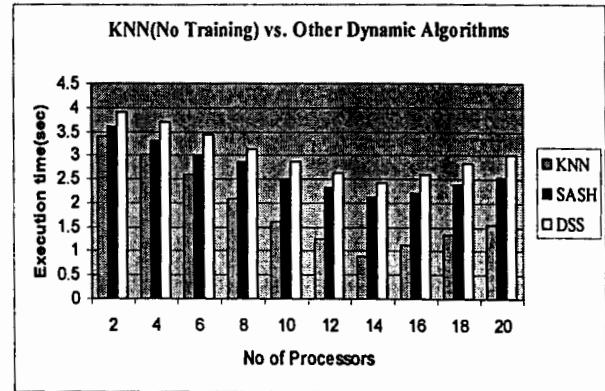


Fig 3: KNN (No Training) vs. other dynamic Algorithms

Initially we see that execution time of KNN is higher but as no of processors increases the difference begins to fall off. And its performance is better as compared to other dynamic algorithms. This is because KNN consumes lot of time for training.

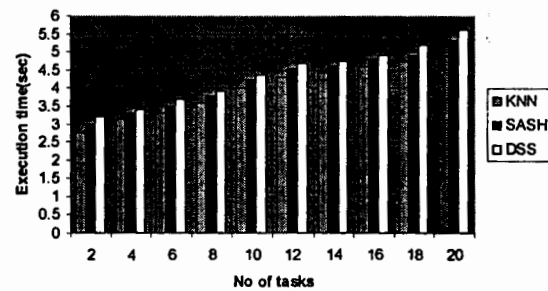


Fig 4: KNN (With Training) vs. Other Dynamic Algorithms w.r.t No of tasks

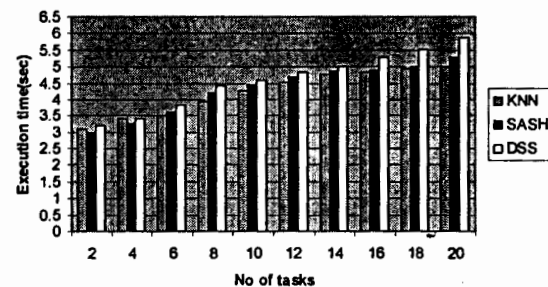


Fig 5: KNN (No Training) vs. Other Dynamic Algorithms w.r.t No of tasks

In fig 4 and fig 5 the general trend is same but in fig 4 we see that KNN takes less time in the beginning as it is trained but as no of tasks increases, KNN takes more time to complete the job as the no of processors is also fixed. But the performance of KNN is still better than other dynamic algorithms. KNN incurs some delay as no of tasks increases due to context switching between the processors.

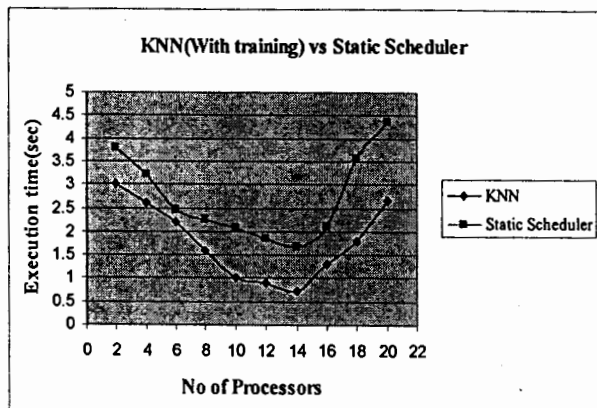


Fig 6: KNN (With Training) vs. Static Scheduler

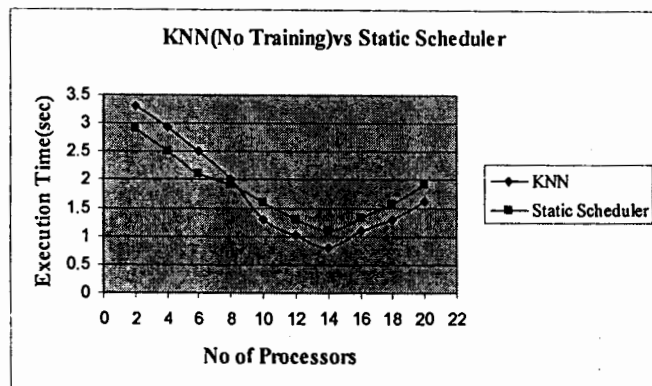


Fig7: KNN (No Training) vs. Static Scheduler

These figures show that the general trend is same as before except that in fig 6 we see that KNN outperforms static scheduler after some time. We also see that the difference is much smaller in the beginning because static scheduler also takes some time for their own calculation.

## CONCLUSION

In this paper we presented a load balancing technique based on artificial neural network for heterogeneous environment for computationally intensive applications. Resource management and scheduling in heterogeneous environment is challenging task to achieve. Load balancing is done in such a way so as to minimize job execution time. We incorporated SOM also known as Kohonen Networks a technique of ANN for automation of load balancing. As the results shows the performance of Kohonen networks with respect to other dynamic algorithms and the case where static scheduler is in place. The results showed that the performance was analyzed for large no of tasks, processors in response to execution time. And we can see the fluctuation in execution time with varying no of processors, task sizes, no of tasks etc. As Kohonen takes much time for training but once it is trained it gives better performance in

comparison with other dynamic and Static Schedulers.

## REFERENCES

Aly E. El-Abd and Mohamed I. El-Bendary, 1997. "A Neural Network Approach for Dynamic Load Balancing In Homogeneous Distributed Systems", 30th Hawaii International Conference on System Sciences (HICSS) Volume 1: Software Technology and Architecture, 1997.

Alessandro Bevilacqua,1999. "A Dynamic Load Balancing Method on a Heterogeneous Cluster of Workstations"; Informatica (Slovenia), 23(1);1999.

Attila Gürsoy, Murat Atun, 2000. "Neighborhood Preserving Load Balancing: A Self-Organizing Approach", Euro-Par Parallel Processing, LNCS 1900, pp. 324-41, 2000.

A. Osman,H.Ammar,2002."Dynamic loadbalancing strategies for parallel computers"; Scientific: 110-120, 2002.

Babak Hamidzadeh, David J. Lilja, Yacine Atif, 1995."Dynamic scheduling Techniques for Heterogeneous Computing Systems", Concurrency: practice & Experience, Vol. 7, No. 7,pp. 633-652,October 1995.

Bruce.S.Siegell, Peter Steenkiste, 1994. "Automatic generation of parallel programs with dynamic load balancing", Proc. of the Third IEEE International Symposium on High Performance Distributed Computing; San Francisco, CA, 1994.

Carsten Ernemann, Volker Hamscher, Uwe Schwiegelshohn, Ramin Yahyapour, 2002. "On Advantages of Grid Computing for Parallel Job Scheduling"; 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), 2002.

Cheng-Juei Yu ,Po-Han Chen, and Sheng-De Wang,2005."Adaptive Task Scheduling Algorithms for Master-Worker Applications in Grid Computing"; Eleventh Workshop on Compiler Techniques for High-Performance Computing, 2005.

Chris Stergiou, 1996. "What is a Neural Network", [http://wwwhomes.doc.ic.ac.uk/~nd/surprise\\_96/journal/vol1/cs11/article1.html](http://wwwhomes.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/cs11/article1.html).

Gilles Labonté and Marc Quintin, 1999. "Network Parallel Computing for SOM Neural Networks", In

the proceedings of the High Performance Computing Symposium 1999.

Hans Ulrich Heiss, M. Dormanns, 1993. "Task Assignment by Self-Organizing Maps"; In proceedings Hd93, Internal Report No. 17/93, Dep. of Computer Science, University of Karlsruhe.

Helen D. Karatza, Ralph C. Hilzer, 2002. "Load Sharing in heterogeneous distributed systems", Proceedings of the 2002 Winter Simulation Conference (WSC'02) - Volume 1, 2002.

Helen D. Karatza, 2003. "A Comparison of Load Sharing and Job Scheduling in a Network of Workstations"; Int'l journal of Simulation: Systems, Science Technology, UK Simulation Society, Volume 4, no: 3&4, pp. 4-11, 2003.

Hiroshi Nishikawa, Peter Steenkiste, 1993. "A general Architecture for load balancing in a distributed memory environment"; Proc. of 13<sup>th</sup> Intl. Conf. Dist. Computing Systems, pp. 47-54. IEEE, Pittsburgh May 1993.

Ian Foster and C. Kesselman, editors (1999). *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers.

Ian Foster, C. Kesselman, and S. Tuecke, 2001. "The Anatomy of the Grid-Enabling Scalable Virtual Organizations"; International Journal of High Performance Computing Applications, Vol. 15, No. 3, 200-222.

Ian Foster, 2002. "What is the Grid? A Three Point Checklist", GRID Today, no.100136.

Ian Foster, C. Kesselman, J.M. Nick, S. Tuecke, 2002. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration"; Open Grid Service Infrastructure WG, Global Grid Forum.

Jennifer M Schopf, 2002. "A general Architecture for scheduling on the grid", Special issue of Journal of Parallel and Distributed Computing on grid computing.

Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd, 2003. "Agent- Based Grid Load Balancing Using Performance-driven Task Scheduling"; Proc. of 17<sup>th</sup> IEEE International Parallel & Distributed Processing Symposium (IPDPS 2003); Nice, France; April 2003.

MarisOrbidans, "A Neural Network based dynamic load balancing algorithm"; <http://maris.site.lv/loadbalancing/>

Menno Dobber, Ger Koole, Rob van der Mei, 2004. "Dynamic Load Balancing for Grid Applications"; 11<sup>th</sup> Int'l Conference, Proceedings HiPC; Bangalore, India, dec19-22, 2004.

Min You Wu, 1995. "Parallel Incremental Scheduling"; Parallel Processing Letters, vol 5, No(4):659-670.

Mohammed Javeed Zaki, W. Li, S. Parthasarathy, 1997. "Customized dynamic load balancing for a network of workstations"; Proc. 5<sup>th</sup> IEEE Int. Symposium on High performance distributed computing, Syracuse, New York, August 1997.

Murat Atun and Attila Gürsoy, 2000. "A new load-balancing algorithm using self-organizing maps"; EuroPar2000, Parallel Processing, 6<sup>th</sup> International Euro-Par Conference Germany, 2000

Pankaj Mehra, Benjamin Wah, 1993. "Automated Learning of workload Measures for Loadbalancing on a Distributed System"; Proc. of the International Conference on Parallel Processing, CRC Press, vol. III, pp. 263-270, Aug. 1993.

Shuichi Ichikawa, S. Yamashita, 2000. "Static Load Balancing of Parallel PDE Solver for Distributed Computing Environment" Proc. ISCA 13<sup>th</sup> Int'l Conf. Parallel and Distributed Computing Systems (PDCS2000); pp. 399-405 (2000).

Teuvo Kohonen, 2001. "Self-Organization and Associative Memory"; Third, Extended Edition, Springer-Verlag, Berlin Heidelberg.

## **APPENDIX-B**

## B. USER MANUAL

The user manual contains the screenshots to give an insight into the software developed. Fig 1 is the main application window. Green color shows the nodes which are available or online and the red shows which are unavailable or offline. The progress bar of load and memory shows the load and currently used memory from the available memory on each node. The label to the right shows all memory. Fig 2 shows file execution. Any executable file can be executed from this window by selecting the criteria as shown in fig 2. Resource analyzer will display the resource information in the forms of graphs as shown in fig 3 and fig 4. Graphs will be generated on the basis of speed, memory, load and status and will be updated after user defined time (refresh rate). Fig 5 shows memory overview. Figure 6 shows Processes on Node. Fig 7 shows Task Submission. Fig 8 shows Queued tasks buffered in grid Scheduler. Fig 9 shows Running Jobs. Fig 10 shows Grid Node and Resource Information. Fig 11 and fig12 shows log file.

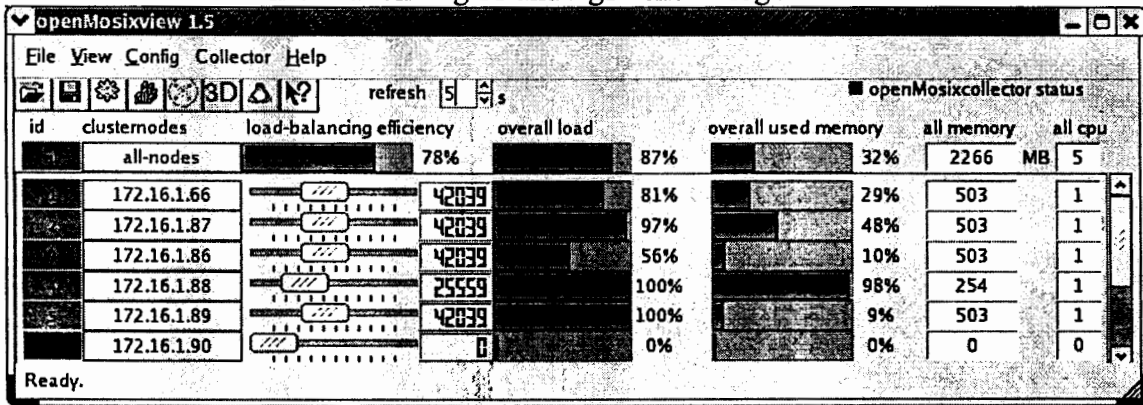


Fig 1 Resource Collector with Online and Offline nodes

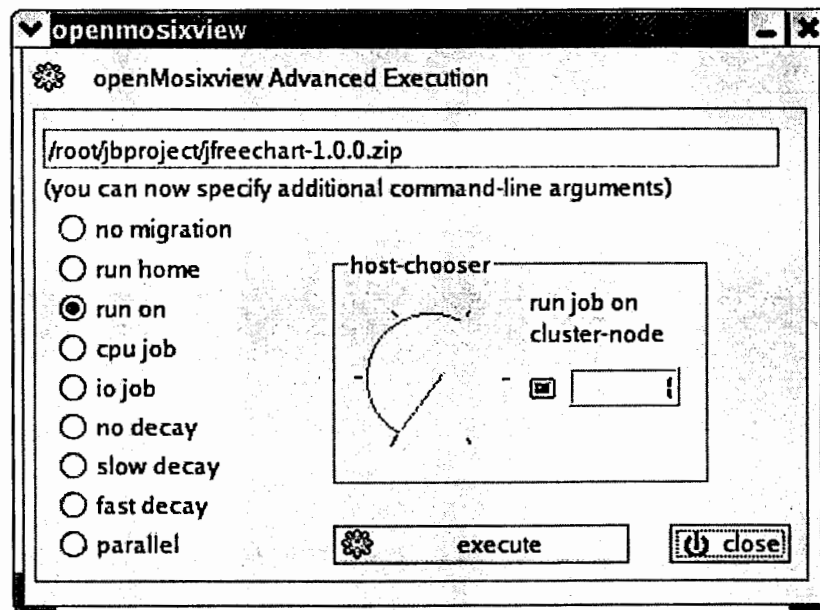


Fig 2 File Execution

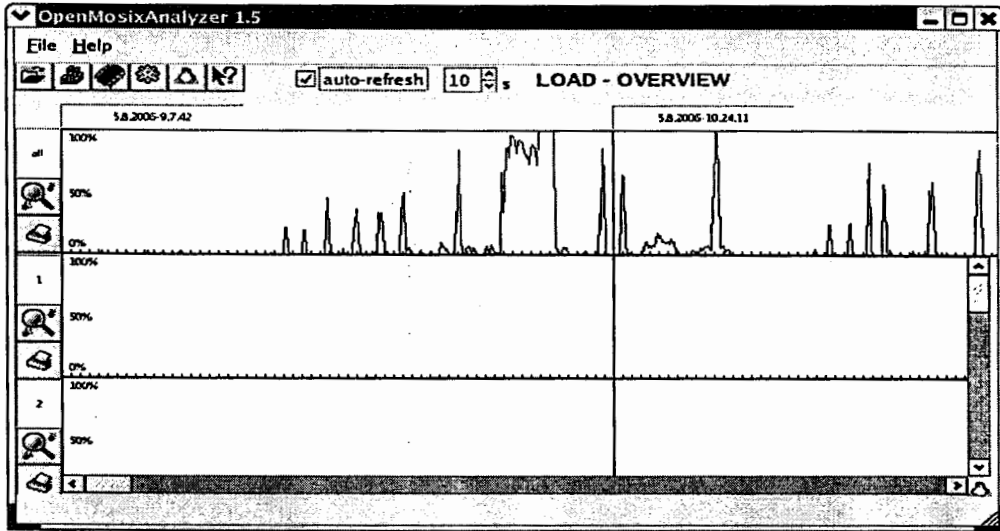


Fig 3 Load Overview

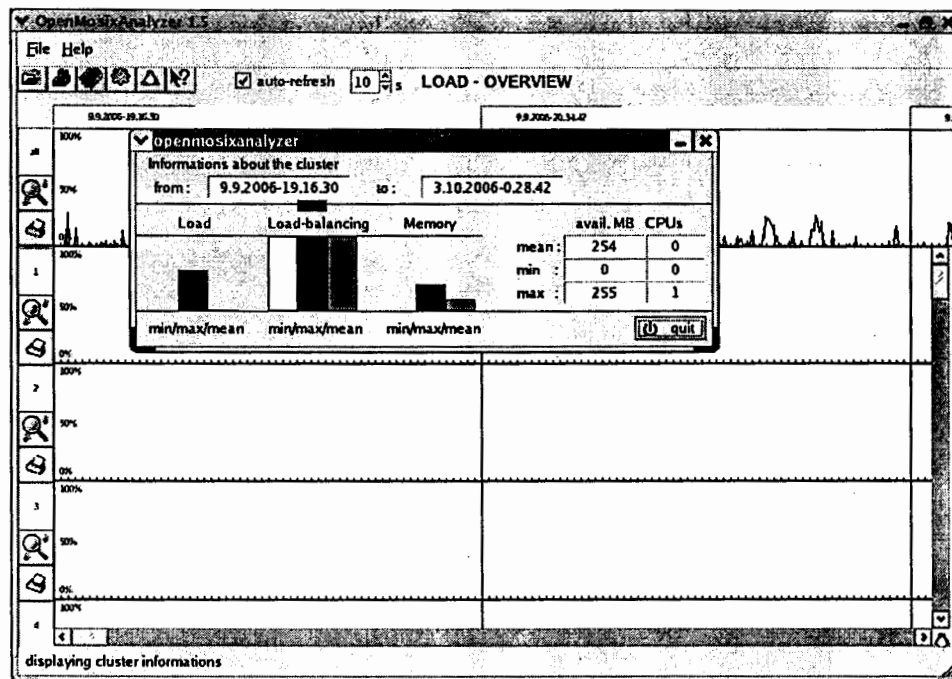


Fig 4 Resource Overview



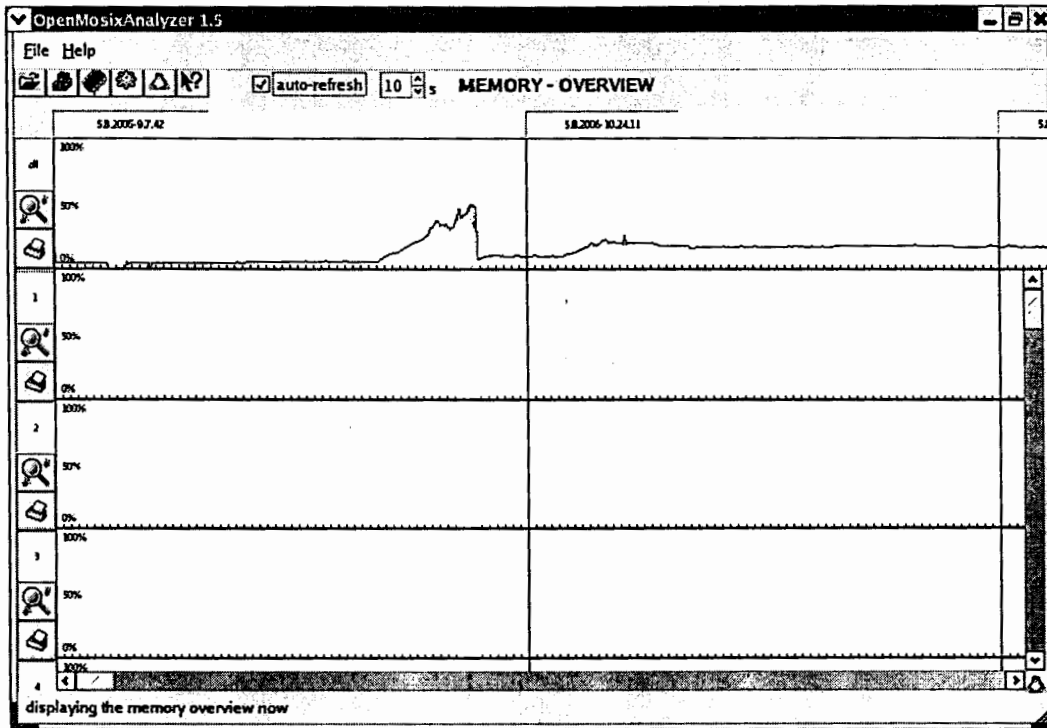


Fig 5 Memory Overview

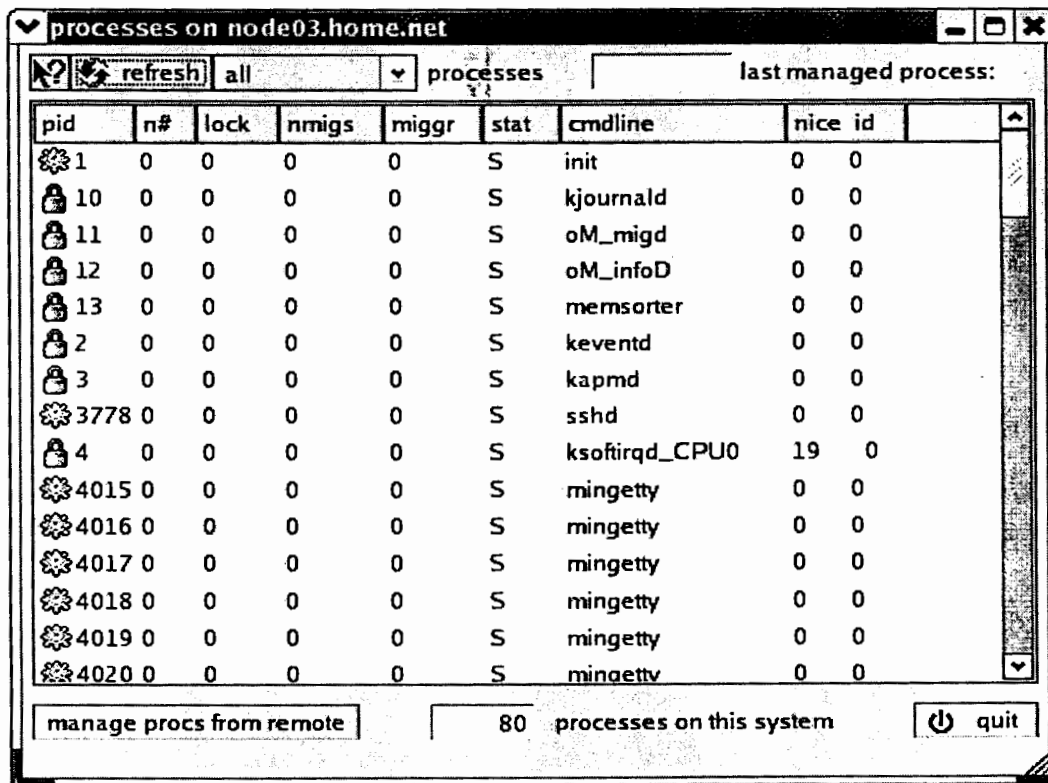


Fig 6 Processes on Grid Node

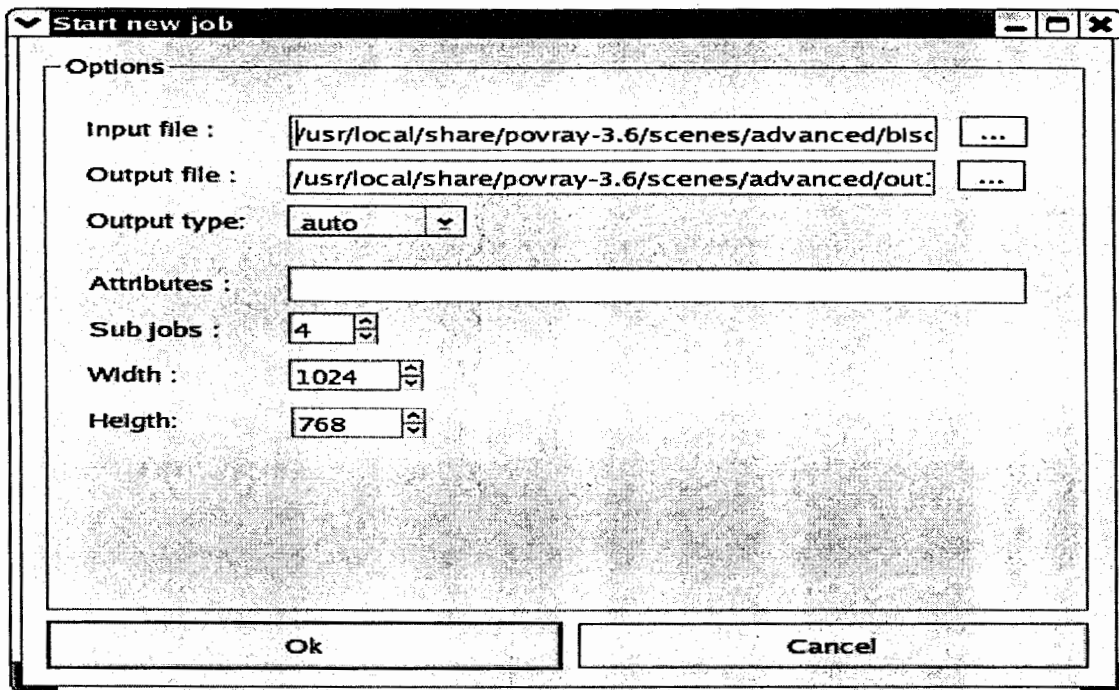


Fig7 Task Submission

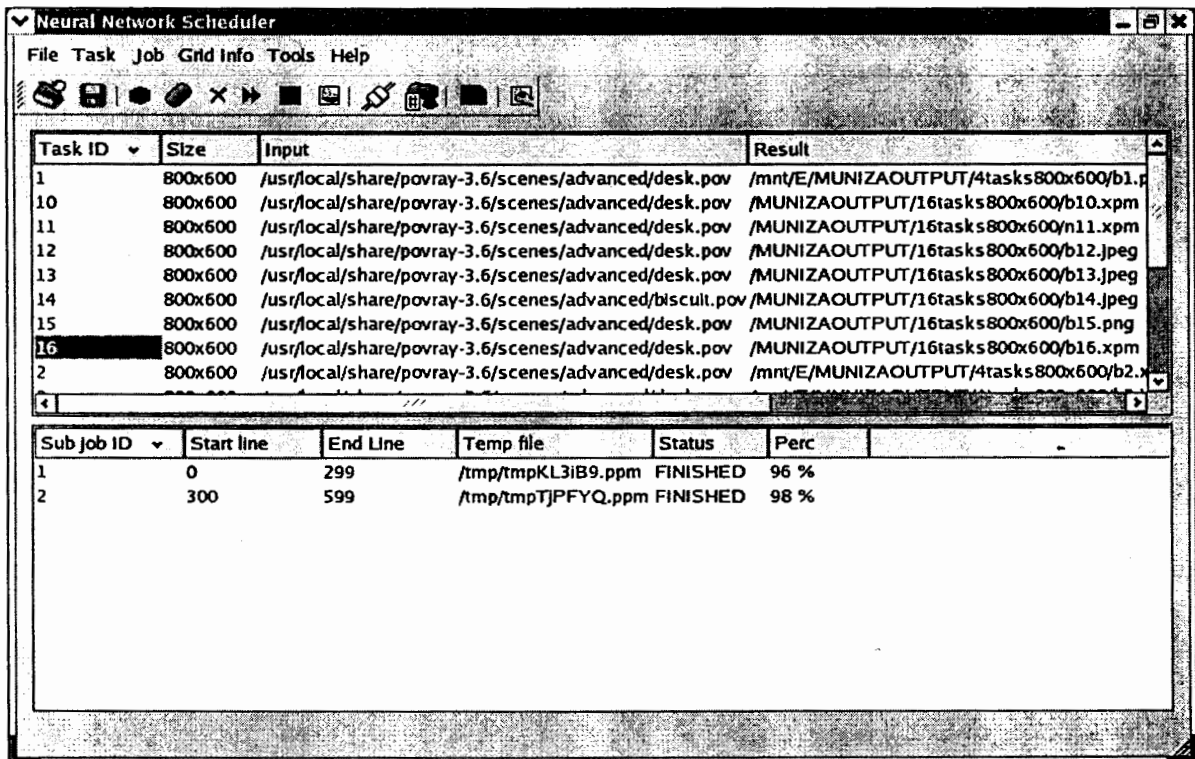


Fig 8 Queued Tasks Buffered in Neural Network Scheduler

The screenshot shows a window titled "Show running jobs" with a table containing the following data:

Task Id	Job Id	Pid	Node Id
1	13	7344	local
1	14	7345	local
1	15	7346	local
1	16	7347	local
1	17	7348	local
1	18	7349	local
1	19	7350	local
1	20	7351	local
1	21	7352	local
1	22	7353	3
1	23	7354	3
1	24	7355	3
1	25	7356	3
1	26	7357	6
1	27	7358	6
1	28	7359	6
1	29	7360	6
1	30	7361	6

An "Ok" button is located at the bottom of the dialog box.

Fig 9 Show Running Jobs

The screenshot shows a window titled "Grid Resource Information" with the following sections:

**Local**

Node Id :

Automatic proc. migration :

**Nodes**

Id	Cpus	Load	Avail. Mem	Speed	Status
1	-101	-101	0 Mb	-101	1
2	-101	-101	0 Mb	-101	1

Buttons for "Refresh" and "Ok" are located at the bottom of the dialog box.

Fig 10 Grid Node and Resource Information

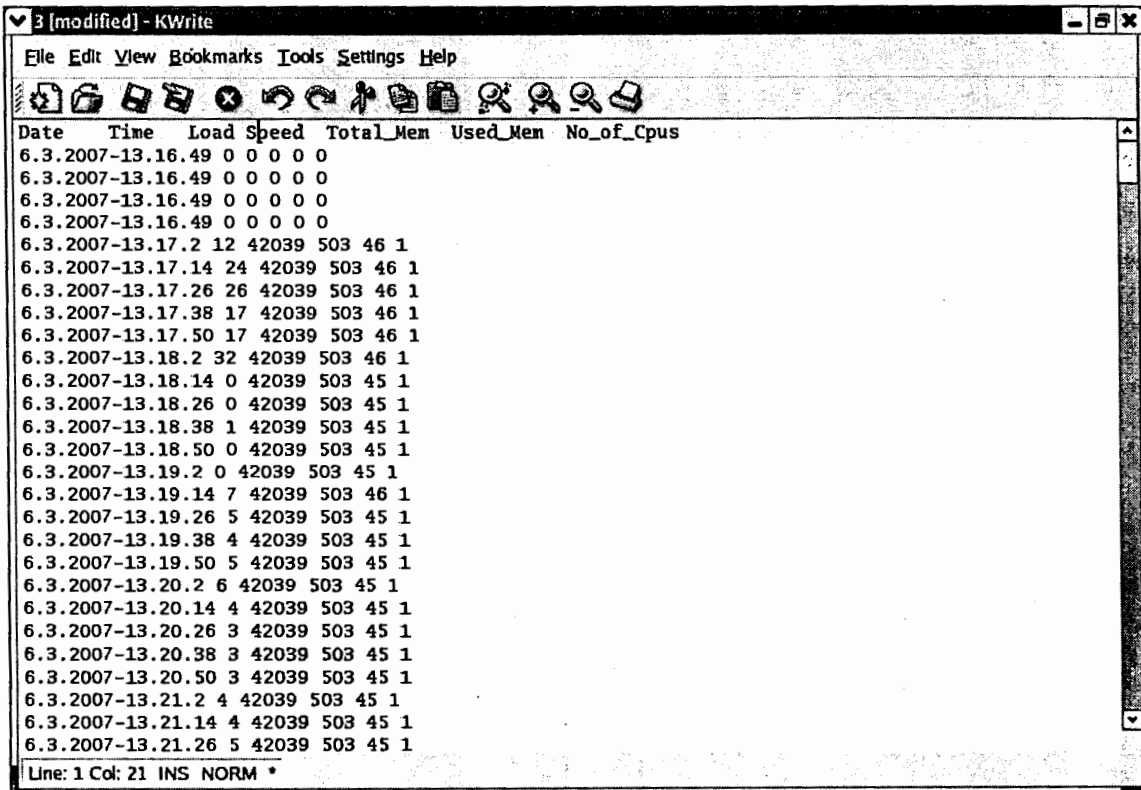
TaskLogFile - WordPad

File Edit View Insert Format Help

Size	Jobs	STATUS	SUBMIT DATE/TIME	COMPLETE DATE/TIME	TOTAL TIME
8600x400 8Node ID 81 8Avg Mem Used 812 8Avg Load 83 8CPUs 81 8Speed 825559 8Status 8Online	84	8FINISHED	806 06 06 / 11:55 PM	806 06 06 / 11:56 PM	80.166667
8600x400 8Node ID 81 8Avg Mem Used 812 8Avg Load 84 8CPUs 81 8Speed 825559 8Status 8Online	84	8FINISHED	806 06 06 / 11:58 PM	807 06 06 / 12:03 AM	80.15
81024x768 8Node ID 81 8Avg Mem Used 8148 8Avg Load 8135 8CPUs 81 8Speed 825559 8Status 8Online	84	8FINISHED	805 06 06 / 01:49 AM	807 06 06 / 12:10 AM	80.416667
8600x400 8Node ID 81 8Avg Mem Used 812 8Avg Load 81 8CPUs 81 8Speed 825559 8Status 8Online	84	8FINISHED	807 06 06 / 12:15 AM	807 06 06 / 12:16 AM	80.05
8700x500 8Node ID 81 8Avg Mem Used 812 8Avg Load 8205 8CPUs 81 8Speed 825559 8Status 8Online	82	8FINISHED	807 06 06 / 12:20 AM	807 06 06 / 12:22 AM	80.0166667
8800x500 8Node ID 81 8Avg Mem Used 812 8Avg Load 8244 8CPUs 81 8Speed 825559	86	8FINISHED	807 06 06 / 12:19 AM	807 06 06 / 12:22 AM	80.0333333

For Help, press F1

Fig11 Task Log file



Date	Time	Load	Speed	Total_Mem	Used_Mem	No_of_Cpus
6.3.2007-13.16.49	0	0	0	0	0	0
6.3.2007-13.16.49	0	0	0	0	0	0
6.3.2007-13.16.49	0	0	0	0	0	0
6.3.2007-13.16.49	0	0	0	0	0	0
6.3.2007-13.17.2	12	42039	503	46	1	
6.3.2007-13.17.14	24	42039	503	46	1	
6.3.2007-13.17.26	26	42039	503	46	1	
6.3.2007-13.17.38	17	42039	503	46	1	
6.3.2007-13.17.50	17	42039	503	46	1	
6.3.2007-13.18.2	32	42039	503	46	1	
6.3.2007-13.18.14	0	42039	503	45	1	
6.3.2007-13.18.26	0	42039	503	45	1	
6.3.2007-13.18.38	1	42039	503	45	1	
6.3.2007-13.18.50	0	42039	503	45	1	
6.3.2007-13.19.2	0	42039	503	45	1	
6.3.2007-13.19.14	7	42039	503	46	1	
6.3.2007-13.19.26	5	42039	503	45	1	
6.3.2007-13.19.38	4	42039	503	45	1	
6.3.2007-13.19.50	5	42039	503	45	1	
6.3.2007-13.20.2	6	42039	503	45	1	
6.3.2007-13.20.14	4	42039	503	45	1	
6.3.2007-13.20.26	3	42039	503	45	1	
6.3.2007-13.20.38	3	42039	503	45	1	
6.3.2007-13.20.50	3	42039	503	45	1	
6.3.2007-13.21.2	4	42039	503	45	1	
6.3.2007-13.21.14	4	42039	503	45	1	
6.3.2007-13.21.26	5	42039	503	45	1	

Line: 1 Col: 21 INS NORM \*

Fig 12 Resource Log file