A DYNAMIC LOAD BALANCING ALGORITHM COMPUTATIONAL GRID USING ZERO CONFIGURATION SCHEDULING

A PROJECT REPORT

Submitted by

YOGADHARANI M.

in partial fulfillment for the requirement of award of the degree
of

MASTER OF ENGINEERING
in

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering

KUMARAGURU COLLEGE OF TECHNOLOGY,

COIMBATORE 641 049

(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2013

A DYNAMIC LOAD BALANCING ALGORITHM IN COMPUTATIONAL GRID USING ZERO CONFIGURATION SCHEDULING

A PROJECT REPORT

Submitted by

YOGADHARANI M.

in partial fulfillment for the requirement of award of the degree
of

MASTER OF ENGINEERING

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE 641 049
(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2013

BONAFIDE CERTIFICATE

Certified that this project work titled **"A DYNAMIC LOAD BALANCING ALGORITHM IN COMPUTATIONAL GRID USING ZERO CONFIGURATION SCHEDULING"** is the bonafide work of MS.YOGADHARANI M. (1120108025), who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other students.

Prof. N. JAYAPATHI M.Tech.,	Ms. P. DEVAKI M.E.,
HEAD OF THE DEPARTMENT	SUPERVISOR
Professor	Assistant Professor
Dept. of Computer Science and Engineering	Dept. of Computer Science and Engineering
Kumaraguru College of Technology	Kumaraguru College of Technology
Coimbatore – 641 049	Coimbatore – 641 049

Submitted for the Project Viva-Voce examination held on ____________

-------------------------------	-----------------------------
Internal Examiner	**External Examiner**

ABSTRACT

In Grid networks, peers are having heterogeneous configurations. In large peer to peer networks balancing the load all over network is a big issue. Balancing the load across the network is important to manage the resources efficiently. Existing load balancing mechanisms distribute the load equally to all peers. But it is not optimal in Grid networks. The load has to be dedicated based on its capacity. So propose a performance-based load delegation which will delicate the load based on each peer's load and performance. The policies are defined by the peers based on its capacity. Data-aware scheduling where the replication of the accessibility of data is considered. Further, the training of the system should be made online, during scheduling in form the of a self-learning system. Learning scheduling decision with machine learning techniques would lead to a zero-configuration scheduling system which can easily integrated into existing middleware solutions.Grid has standards based on persistent, addresses security issues; resources are more powerful, more diverse, better connected, data intensive, facing problems of autonomic configuration and management, not much scalable. But in P2P has much scalability, fault tolerance, self-configuration, automatic problem determination, higher variable behaviour, but lack of infrastructure, security problems, less concerned with qualities of service. The main objective is to distribute the load at run time, to reduce makespan i.e., the time difference between the start and finish of a sequence of jobs or tasks and improve its performance in a heterogeneous network.

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for enabling me to complete this project. I express my profound gratitude to **Padma Bhushan Arutselvar Dr.N.Mahalingam B.Sc., F.I.E. Chairman, Dr.B.K.Krishnaraj Vanavarayar B.com., B.L., Co-Chairman, Mr.M.Balasubramaniam M.Com., MBA., Correspondent, Mr.Sankar Vanavarayar MBA., PGDIEM., Joint Correspondent** and **Dr.S.Ramachandran Ph.D., Principal** for providing the necessary facilities to complete my project.

I take this opportunity to thank **Prof.N.Jayapathi M.Tech.,** Head of the Department, Department of Computer Science and Engineering, for his support and motivation. Special thanks to my Project Coordinator **Dr.V.Vanitha M.E., Ph.D.,** Senior Associate Professor, Department of Computer Science and Engineering, and project committee members.

I register my sincere thanks to my guide **Ms.P.Devaki M.E.,** Assistant Professor, Department of Computer Science and Engineering. I am grateful for her support, encouragement and ideas. I would like to convey my honest thanks to all **Teaching** and **Non Teaching** staff members of the department and my classmates for their support.

I dedicate this project work to my **Parents** for no reasons but feeling from bottom of my heart that without their love, this work would not be possible.

-YOGADHARANI M.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AFTO	Adjusted Fair Task Order
AWF	Accept When Fit
CG	Computational Grid
DLBA	Dynamic Load Balancing Algorithm
EDF	Earliest Deadline First
FCFS	First Come First Serve
GGF	Global Grid Forum
GRMS	Grid Resource Management System
GUI	Graphical User Interface
WLAN	Wireless Local Area Network

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF GRID

Grid computing has been increasingly considered as a promising next-generation computing platform that supports wide area parallel and distributed computing since its advent in the mid-1990s [Foster, 1999]. It couples a wide variety of geographically distributed computational resources such as PCs, workstations, and clusters, storage systems, data sources, databases, computational kernels, and special purpose scientific instruments and presents them as a unified integrated resource.

Grids address issues such as security, uniform access, dynamic discovery, dynamic aggregation, and quality of services. In computational grids, heterogeneous resources with the different systems in different places are dynamically available and distributed geographically. The user's resource requirements in the grids vary depending upon their goals, time constraints, priorities and budgets. Allocating tasks to the appropriate resources in the grid, so that performance requirements are satisfied. Allocating the resources to the proper users so that utilization of resources and the profits generated are maximized is also an extremely complex problem. From a computational perspective, it is impractical to build a centralized resource allocation mechanism in such a large scale distributed environment. A computational grid is less expensive than purchasing more computational resources while obtaining the same amount of computational power for their computational tasks.

1.1.1 Scheduler Architecture

The scheduler architecture adopted in the GRIDLAB infrastructure which is shown in Fig 1.1, the main modules of which are the following:

Queuing System

Each time a task is submitted for execution, its characteristics (for example, the task deadline) are stored on a database, which is the core of the queuing system module.

Queuing Order

This unit addresses the task-queue ordering problem; that is, it determines the order in which the tasks are considered for assignment to the available resources. The queuing order unit communicates with the queuing system module, where the tasks requesting service along with their respective characteristics are stored and with the resource discovery module, which determines the available resources of the infrastructure.

Resource Discovery

This module determines the available resources of the grid.

Processor Assignment

The information collected by the queuing order unit is then passed through the processor assignment unit, which determines the processor on which each task is assigned.

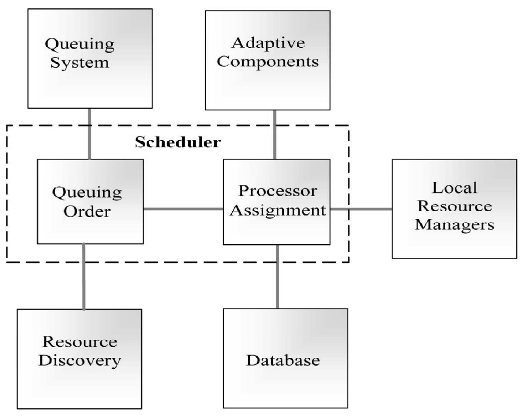


Fig.1.1 Scheduler Architecture

Adaptive Components

This module is responsible for

1. Predicting the workload of the tasks requesting service and
2. Estimating the task-ready times _i through the respective communication delays and the processor release times _j of the tasks already allocated to processor j.

The adaptive component module provides the information required by the scheduler to perform the task queue ordering and the processor assignment functions. A method for predict the task workload in the particular case of 3D rendering applications.

Local Resource Manager

This module is responsible for implementing the task execution locally as instructed by the scheduler.

1.1.2 Load Balancing

Load balancing is a computer networking methodology to distribute the workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy. The load balancing service is usually provided by dedicated software or hardware, such as a multilayer switch or a Domain Name System server shown in Fig 1.2.

Load balancing technique to enhance resources, utilizing parallelism, exploiting throughput improvisation, and to reduce response time through an appropriate distribution of the application. Load balancing algorithms can be defined by their implementation of the following policies:

Information policy: It states the workload of task information to be collected, when it is to be collected and from where.

Triggering policy: It determines the appropriate period to start a load balancing operation.

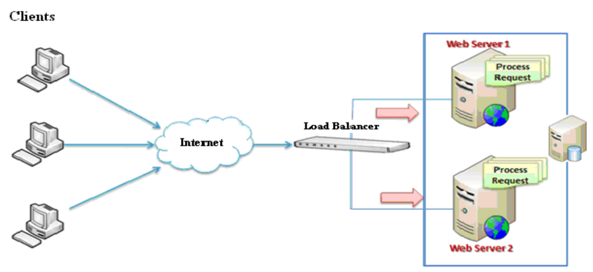


Fig.1.2 Load Balancing

Resource type policy: It orders a resource as server or receiver of tasks according to its availability status.

Location policy: It uses the results from the resource type policy to find a suitable partner for a server or receiver.

Selection policy: It defines the tasks that should be migrated from overloaded resources (source) to most idle resources (receiver).

Load balancing algorithms are defined by two types such as static and dynamic. Static load balancing algorithms allocate the tasks of a parallel program to workstations. Multi computers with dynamic load balancing allocate or reallocate resources at runtime based on task information, which may determine when and whose tasks can be migrated. Karthick Kumar (2000) has implemented the dynamic load balancing algorithm to multi computers based on resource type policy.

Load Sharing Vs Load Balancing

Load sharing means one can split the traffic from a network to be transported by different routers (paths). That is configure half of the hosts with one default gateway and the second half with the other. E.g.: PBR configuration in routers for link distribution for host subnets.

On the other hand, load balancing means distributing the traffic evenly and dynamically among different paths to avoid link congestion and saturation. This can be done in a packet-by-packet basis or per destination in a round-robin fashion. The packets sent by a host follow different paths to the same destination. All paths belong to all hosts.

1.2 LITERATURE SURVEY

In this section, papers related to load balancing and scheduling jobs on a grid computing environment were discussed.

1.2.1 FAIR SCHEDULING ALGORITHMS IN GRIDS (Doulamis, 2007)

Scheduling and resource management are important in optimizing multiprocessor grid resource allocation and determining its ability to deliver the negotiated Quality-of-Service (QoS) requirements. This need has been confirmed by the Global Grid Forum (GGF) in the special working group dealing with the area of scheduling and resource management for grid computing. The resource manager receives information about the job characteristics and determines when and on which processor each job will execute.

Proposed a new algorithm for fair scheduling, and compare it to other scheduling schemes such as the Earliest Deadline First (EDF) and the First Come-First Served (FCFS) schemes. This algorithm uses a max-min fair sharing approach for providing fair access to users. When there is no shortage of resources, the algorithm assigns to each task to the resource which has enough computational power to finish within its deadline. When there is congestion, the main idea is to fairly reduce the CPU rates assigned to the tasks so that the share of resources that each user gets is proportional to the user's weight. The weight of a user may be defined as the user's contribution to the infrastructure or the price is willing to pay for services or any other socioeconomic consideration. In this algorithm, all tasks whose requirements are lower than their fair share CPU rate are served at their demanded CPU rates. However, the CPU rates of tasks whose requirements are larger than their fair share CPU rate are reduced to fit the total available computational capacity in a fair manner. Three different versions of fair scheduling are: The Simple Fair Task Order (SFTO), which schedules the tasks according to their respective fair completion times, the Adjusted Fair Task Order (AFTO), which

refines the SFTO policy by ordering the tasks using the adjusted fair completion time, and the Max-Min Fair Share (MMFS) scheduling policy, which simultaneously addresses the problem of finding a fair task order and assigning a processor to each task based on a max-min fair sharing policy.

The traditional scheduling schemes such as the EDF and the FCFS are using three different error criteria. It has been overcome by reducing the cycle time reduction and concentrate on to improve the performance of the system.

1.2.2 TASK RESOURCE ALLOCATION IN GRID USING SWIFT SCHEDULER (K. Somasundaram, 2009)

The task of Grid resource broker and scheduler is to dynamically identify and characterize the available resources and to select and allocate the most appropriate resources for a given job. The resources are typically heterogeneous locally administered and accessible under different local policies. Advance reservation is currently being added to Portable Batch System (PBS).

In a Grid Scheduler, the mapping of Grid resources and an independent job in optimized manner is so hard where couldn't predict optimized mapping. So the combination of uninformed search and informed search will provide the good optimal solution for mapping resources and jobs, to provide minimal turnaround time with minimal cost and minimize the average waiting time of the jobs in the queue. A heuristic algorithm is an algorithm that ignores whether the solution to the problem can prove to be correct, but which usually produces a good solution. Heuristics are typically used when there is no known way to find an optimal solution, or when it is desirable to give up finding the optimal solution for an improvement in run time.

The primary objective is to investigate effective resource allocation techniques based on computational economy through simulation. To simulate millions of resources and thousands of users with varied requirements and study scalability of systems, algorithms, efficiency of resource allocation policies and satisfaction of users. In this simulation model applications are in the areas of biotechnology, astrophysics, network design, and high-energy physics in order to study usefulness of resource allocation techniques. This work will have the significant impact on the way resource allocation is performed for solving problems on grid computing systems.

Provides level of determinism on the waiting time of each job as disadvantage. The advantage of FCFS shows up when the jobs at the head of the ready queue cannot be scheduled immediately due to insufficient system resources, but jobs further down the queue would be able to execute given the currently available system resources.

1.2.3 UNSPLITTABLE MAX-MIN DEMAND ALLOCATION – A ROUTING PROBLEM (Pal Nilsson)

In a routing problem involving max-min fair allocation of bandwidth to demands in a communication network is dealt with. The considered problem has been studied in its simplified form already in, and has also, as a convex optimization problem, been solved for bifurcated (splittable) flows. It addresses a more difficult version when only non-bifurcated (unsplittable) flows are allowed. Such an assumption, also called a requirement of single-path flows, results in non-convex problem formulations which are inherently hard. Unsplittable flows are often a realistic restriction due to the used routing protocol, or simply an explicit management requirement, stipulating avoidance of packet resequencing in receiving nodes. With the network given in terms of topology and link capacities, the following traffic engineering problem is identified: The demand between each source-destination (S-D) node-pair must be associated with a single path such that a

sufficient volume of flow can be routed on demands' single paths simultaneously, without exceeding the link capacities. By sufficient volume that is equitable among different S-D pairs are addressed i.e., a fair sharing of resources. Particularly, the problem of assigning a single-path flow to each demand such that the flow distribution in Max-Min Fair (MMF). Consequently, once each S-D pair is assigned a single path, the problem is reduced to max-min fair sharing of corresponding link capacities, for which an efficient polynomial time algorithm exists. Thus essentially, the considered problem amounts to the very hard task of appropriate path selection.

It provides single-path flow.It reduce to the max-min fair sharing of corresponding link capacities, and identify traffic problems as advantage. The disadvantage is too difficult to identify path selection.

1.2.4 A DYNAMIC ERROR BASED FAIR SCHEDULINGALGORITHM FOR A COMPUTATIONAL GRID (Daphne Lopez, 2009)

Various types of grids have been developed to support applications and categorized as Computational Grids, Data Grids and Service Grids. Computational Grid (CG) represents a new computational framework whose efficient use requires schedulers that allocate user's tasks to the grid resources in an acceptable amount of time.

An efficient use of distributed resources is highly dependent on the resource allocation by grid schedulers, where user requirements and job characteristics must be also considered. Moreover, due to the changeability of a CG, machines and jobs to be scheduled may vary over time, and therefore, any grid scheduler must generate optimal schedules at a minimal amount of time in order to rapidly adapt itself to the changes of the grid. Major issues that can easily handled in conventional computing environments become seriously challenging problems in grids mainly because a grid consists of multiple administrative domains. Two very

crucial issues among them are security and scheduling which have been investigated and researched over time.

The demand for scheduling is to achieve high-performance computing. The motivation is to develop a good scheduling algorithm that can perform effectively and efficiently in terms of minimizing the error to achieve fairness and reduce the cost and time. Here proposes a fair scheduling algorithm based on the service time error. Fair Share is a widely used queuing algorithm for prioritorizing jobs based on of a "share". The first part explains the algorithm and secondly the simulation of the experiment with GridSim toolkit is presented. The simulator defines the workload of resources, the arrival time of independent jobs, length of each job and other parameters. Finally compare the performance with FCFS and Round Robin.

Genetic algorithm methods minimize the total task completion time as advantage. The major issue in genetic algorithm is the process of finding a good solution can take a very long time as disadvantage.

1.2.5 RECITATION OF LOAD BALANCING ALGORITHMS IN GRID COMPUTING ENVIRONMENT USING POLICIES AND STRATEGIES - AN APPROACH (M.Kamarunisha, 2011)

The rapid development in computing resources has enhanced the performance of computers and reduced their costs. This availability of low cost powerful computers coupled with the popularity of the Internet and high-speed networks has led the computing environment to be mapped from distributed to grid environments. In fact, computing architectures are allowed the emergence of a new computing paradigm known as grid computing. Grid is a type of distributed system which supports the sharing and coordinated use of geographically distributed and multi owner resources, independently from their physical type and location, in dynamic virtual organizations that share the same goal of solving large-scale applications.

In Grid computing, individual users can access computers and data, transparently, without having to consider location, operating system, account administration, and other details. In grid computing, the details are abstracted, and the resources are virtualized. Grid computing has emerged as a new and important field and can be visualized as an enhanced form of distributed computing. Sharing in a grid is not just a simple sharing of files but of hardware, software, data, and other resources. Thus a complex yet secure sharing is at the heart of the grid.

It uses centralized and sender-initiated load balancing algorithm and the disadvantage is sender-initiated policy found to yield performance not far from optimal, also particularly light to moderate systems loads.

1.2.6 IMPROVING PERFORMANCE IN LOAD BALANCING PROBLEM ON THE GRID COMPUTING SYSTEM (Prabhat Kr.Srivastava, 2011)

Grid computing is a type of parallel and distributed system that enables the distribution, selection and aggregation of geologically resources dynamically at run time depending on their availability, capability, performance, cost, user quality-of –self-service requirement. Grid computing, individual users can retrieve computers and data, transparently, without taking into account the location, operating system, account administration, and other details. In grid computing, the details are abstracted, and the resources are virtualized. Grid computing should enable the job in question to be run on an idle machine elsewhere on the network. Grids functionally bring together globally distributed computers and information systems for creating a universal source of computing power and information. A key characteristic of grids is that resources (e.g., CPU cycles and network capacities) are shared among various applications, and therefore, the amount of resources available to any given application highly fluctuates over time. Load balancing is a technique to enhance resources, utilizing parallelism, exploiting throughput improvisation, and to reduce response time through an appropriate distribution of the application. Load balancing algorithms are two types static and dynamic.

(i) Static Load Balancing Algorithm

Static load balancing algorithms allocate the tasks of a parallel program to workstations based on either the load at the time nodes are allocated to some task, or based on an average load of workstation cluster as shown in Fig 1.3. The decisions related to load balance are made at compile time when resource requirements were estimated.

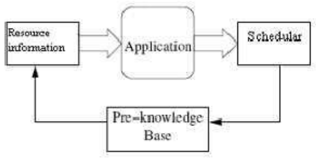


Fig.1.3 Static Load Balancing

(ii) Dynamic Load Balancing Algorithm

Dynamic load balancing algorithm shown in Fig 1.4 make changes to the distribution of work among workstations at a run-time; it use current or recent load information when making distribution decisions. Multi computers with dynamic load balancing allocate/reallocate resources at runtime based on task information, which may determine when and whose tasks can be migrated. As a result, the dynamic load balancing algorithms can provide a significant improvement in performance over static algorithms.

As a result, dynamic load balancing algorithm can provide a major improvement in performance over static algorithms. However, this comes at the

additional cost of collecting and maintaining load information, so it is important to keep these overheads within reasonable limits.

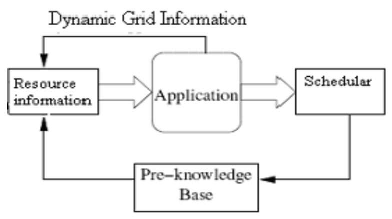


Fig.1.4 Dynamic Load Balancing

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ. These three parameters are:

- The load balancing decision maker.
- Information is used to make the load balancing decision, and
- The place where the load balancing decision is made.

The advantage is to improve resources, utilizing parallelism, exploiting throughput managing and to reduce response time through proper distribution of the application and performance is still an issue in dynamic load balancing.

1.2.7 A NOVEL LOAD BALANCING ALGORITHM FOR COMPUTATIONAL GRID (Saravana kumar E, 2010)

The Grid is emerging as a wide-scale distributed computing infrastructure that promises to support resource sharing and coordinated problem solving in dynamic multi-institutional Virtual Organizations. The idea is similar to

the former Meta computing where the focus was limited to computation resources, whereas grid computing takes a broader approach. The computational grid is the cooperation of distributed computer systems where user jobs can be executed on either local or remote computer systems. With its multitude of heterogeneous resources, a proper scheduling and efficient load balancing across the grid are required for improving the performance of the system. A widely used performance metric is the Average Response Time of tasks. The response time of a task is the time elapsed between its initiation and its completion. Minimizing the average response time is often the goal of load balancing. The system load is a measure of the amount of work that a computer system performs. If loads at some computers are typically heavier than at others, or if some processors execute tasks more slowly than others, will become heavily loaded. The load balancing aims to have all processor's equally heavy workloads over the long term.

In general, any load balancing algorithm consists of two basic policies—a transfer policy and a location policy. The transfer policy decides if there is a need to initiate load balancing across the system. By using workload information, it determines when a node becomes eligible to act as a sender (transfer a job to another node) or as a receiver (retrieve a job from another node). The location policy determines a suitably under loaded processor. In other words, it locates complementary nodes to/from which a node can send/receive workload to improve the overall system performance. Location-based policies can be broadly classified as a sender initiated, receiver initiated, or symmetrically initiated. Based on the information that can be used, load-balancing algorithms are classified as static, dynamic, or adaptive. In a static algorithm, the scheduling is carried out according to a predetermined policy. The state of the system at the time of the scheduling is not taken into consideration.

On the other hand, a dynamic algorithm adapts its decision to the state of the system. Adaptive algorithms are a special type of dynamic algorithms where the

parameters of the algorithm and/or the scheduling policy itself is changed based on the global state of the system. According to another classification, based on the degree of centralization, load-scheduling algorithms could be classified as centralized or decentralized. In a centralized system, only a single processor does the load scheduling. Such algorithms are bound to be less reliable than decentralized algorithms, where many, if not all, processors do load scheduling in the system. Load balancing involves assigning to each processor work proportional to its performance, thereby minimizing the response time of a job. Normally load balancing is done by migrating the job to buddy processors. A set of processors to which a processor is directly connected constitutes its buddy set.

To present a load-balancing algorithm adapted to the heterogeneous grid computing environment. It attempts to propose an adaptive decentralized sender-initiated load balancing algorithm for computational grid environments. Compare to former Meta computing where the focus was limited to computation resources, whereas grid computing takes a broader approach.

1.2.8 A DYNAMIC LOAD BALANCING ALGORITHM IN COMPUTATIONAL GRID USING FAIR SCHEDULING (U. Karthick Kumar, 2011)

One of the most challenging issues in grid computing is efficient scheduling of tasks. Here proposed a load balancing algorithm for fair scheduling, and compare it to other scheduling schemes such as the Earliest Deadline First, Simple Fair Task order, Adjusted Fair Task Order and Max Min Fair Scheduling for a computational grid. It addresses the fairness issues by using mean waiting time. It scheduled the task by using fair completion time and rescheduled by using mean waiting time of each task to obtain load balance. This algorithm scheme tried to provide optimal solution so that it reduces the execution time and expected price for the execution of all the jobs in the grid system is minimized.

Dynamic Load Balancing

Load balancing should take place when the scheduler schedules the task to all processors. There are some particular activities which change the load configuration in grid environment. In Fig.1.5 shows an event diagram of dynamic load balancing algorithm. The activities can be categorized as following:

- Arrival of any new job and queuing of that job to any particular node.
- Scheduler schedules the job to particular processor.
- Reschedule the jobs if load is not balanced
- Allocate the job to processor when it is free.
- Release the processor after it completes the whole job.

Initialization of Algorithm:

Number of tasks that have to be scheduled and workload of tasks are submitted to number of processors.

Scheduling Task:

Scheduler allocates number of demanded tasks to number of processors based on fair completion time of each task.

Load Balancing Algorithm:

It applied when the processor task allocation is excessive than the other after scheduling the task. Rescheduled the task for upper bound and lower bound processor based on workloads.

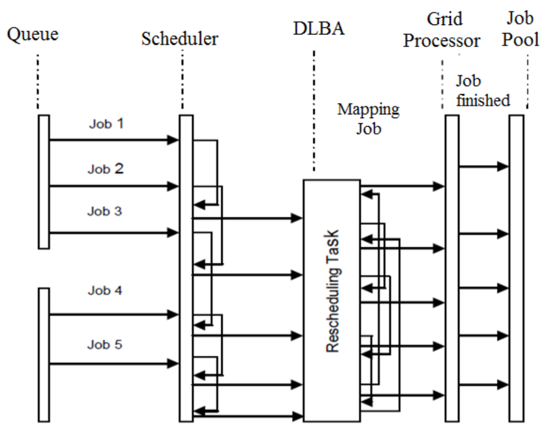


Fig.1.5 an Event Diagram for Dynamic Load Balancing Algorithm

Termination:

This process is repeated until all the processor is balanced. Finally, obtain the optimal solution from the above process. The Fig 1.6 shows the flow chart of load balancing algorithm. It has proved the best results in terms of makespan and execution cost. In particular the algorithm allocates the task to the available processors so that all requesting task get equal amount of time that satisfied their demand.

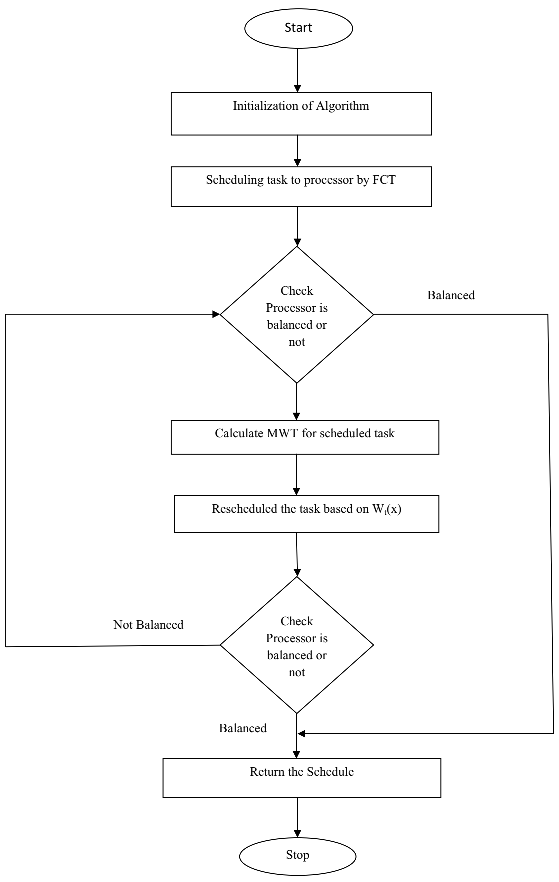

Start
Initialization of Algorithm
Scheduling task to processor by FCT
Check Processor is balanced or not
Balanced
Calculate MWT for scheduled task
Rescheduled the task based on $W_t(x)$
Not Balanced
Check Processor is balanced or not
Balanced
Return the Schedule
Stop

Fig. 1.6 Flow Chart of Algorithm

1.3 PROBLEM DEFINITION

Systems are having heterogeneous network in computational grid. It is less expensive than purchasing more computational resources to obtain the same amount of computational power for the computational tasks. In a large scale distributed environment build a centralized resource mechanism is impractical from computational view. In the grids, the user's resource needs are based on goals, time constraints, priorities and budgets. Performance requirements are satisfied by assigning tasks to the relevant resource in grid. The complex is allocation of resources to proper user's who utilize the maximum resources and the maximum profit. A key characteristic of grids resources are shared among various applications, and therefore, the amount of resources available to any given application highly varies over time. Load balancing is one of the big issues in grid computing. Static load balancing is a system with servers and computers where servers balance the load among all computers in a round robin fashion. The problem of determining which group can arrived job should be allocated to and how its load can be distributed among computers in the group to optimize the performance. Also in proposed algorithms, which guarantee finding a load distribution over computers in a group that leads to the minimum response time or computational cost.

CHAPTER 2

IMPLEMANTATION OF DYNAMIC LOAD BALANCING ALGORITHM IN COMPUTATIONAL GRID USING ZERO CONFIGURATION SCHEDULING

2.1 EXISTING SYSTEM

Heterogeneous resources are distributed and dynamically available in different places of the distinct system in computational grid. The resources requirements are varying depending upon the time constraints, goals, budgets and priorities. Purchasing the computational resources is expensive than computational grid in order to obtain the same amount of computational power for their computational tasks. Delegating tasks to the resources becomes extremely complicated problem in grid. The ultimate problem is allocation of resources to proper user's who utilize the maximum resources and the maximum profit. Centralized resource allocation mechanism becomes impractical in computational perspective.

2.1.1 Drawbacks

- The most profound problem of this scheduling structure is its bad fault tolerance and lack of scalability
- The local user community is often not allowed to bypass the central scheduler for submitting jobs.

2.2 PROPOSED SYSTEM

Load Balancing algorithm tried to improve the performance and total completion time finally proposed a zero configuration scheduling for grid environment. Here proposed an enhanced algorithm which more efficiently implements two policies in load balancing algorithm. These two policies are: Information Policy, and Transfer policy. Dependent on the current system state, the load decision maker has to decide whether to accept an offered job or not. If the local system is already highly loaded, it compares with transfer policy and additionally to near an optimal system which reduces the delay and improves the performance based on zero configuration scheduling which is shown in Fig 2.1.

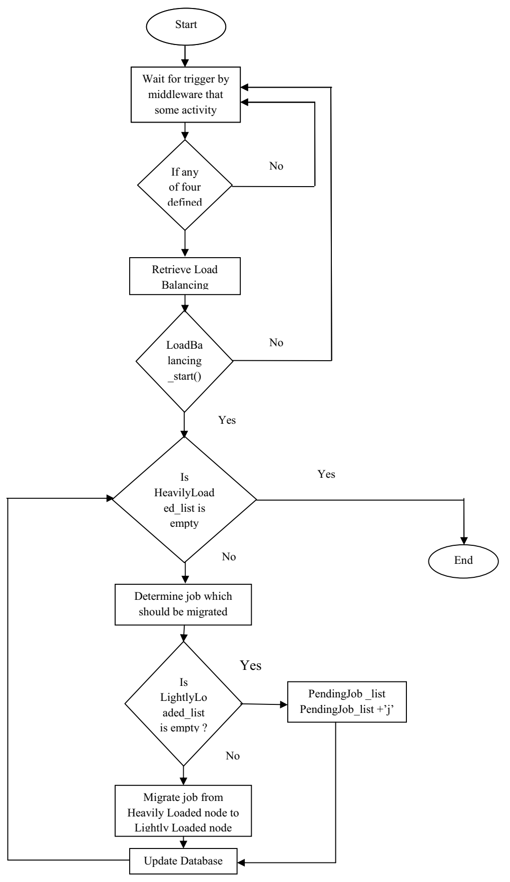

Start
Wait for trigger by middleware that some activity
If any of four defined
No
Retrieve Load Balancing
LoadBalancing _start()
No
Yes
Is HeavilyLoaded_list is empty
Yes
End
No
Determine job which should be migrated
Is LightlyLoaded_list is empty ?
Yes
PendingJob _list PendingJob_list +'j'
No
Migrate job from Heavily Loaded node to Lightly Loaded node
Update Database

Fig.2.1 Workflow of Zero configuration scheduling

Advantages

- This approach has significant improvement over the previous algorithm.
- To apply middleware can provide self-.configuration, and guarantees the interoperability.
- It reduces delay, improves performance and efficiency of the grid systems.

2.3 OVERVIEW OF THE PROJECT

Grid computing is mainly focusing on resource sharing. The scheduling task is the one of a major problem in grid computing. The proposed load balancing algorithm for fair scheduling which is used to compare it to other scheduling schemes such as the Earliest Deadline First, Simple Fair Task order, Adjusted Fair Task Order and Max Min Fair Scheduling for a computational grid. It scheduled the task by using fair completion time and rescheduled by using mean waiting time of each task to obtain load balance. This algorithm scheme tries to provide optimal solution so that it reduces the execution time and expected price for the execution of all the jobs in the grid system is minimized. The performance of the proposed algorithm compared with other algorithms by using simulation. Load balancing algorithms can be defined by their implementation of the following policies. Information policy: It states the workload of task information to be collected, when it is to be collected and from where. Triggering policy: It determines the appropriate period to start a load balancing operation. Resource type policy: It orders a resource as server or receiver of tasks according to its availability status. Location policy: It uses the results of the resource type policy to find a suitable partner for a server or receiver. Selection policy: It defines the tasks that should be migrated from overloaded resources (source) to most idle resources (receiver).

2.4 MODULES

- INFORMATION POLICIES
- GRMS (GRID RESOURCE MANAGEMENT SYSTEM)

- ADAPTIVE DECISION MAKING
- LOAD DELEGATION

2.4.1 Module Description

Information Policies

This policy becomes relevant if more than one exchange partner is available in the grid. Thus, there exists more than one possibility to delegate a job to a remote grid participant. For such scenarios, the location policy determines as a first step the sorted subset of possible delegation targets

Transfer Policy

After the location policy has been applied, the transfer policy specifies whether a job should be delegated to a certain partner or not. For this purpose, the policy is applied separately on each partner in a re determined order. Every time the transfer policy is consulted, it decides whether the job should be executed locally or delegated to the considered partner.

GRMS (Grid Resource Management Systems)

The GRMS layer consists of a waiting queue and a scheduler. The waiting queue stores all locally submitted jobs while the scheduler executes a specific scheduling strategy in order to assign jobs from the waiting queue onto the available local resources. On MPP system layer, this approach allows the realization of priorities for jobs of different user groups. Usually, the scheduling strategies are formulated by the system provider to fulfill the users' needs. Although many special-purpose algorithms exist that are tailored for certain MPP system owner priorities, use the basic and simple First-Come-First-Serve (FCFS) algorithm as an example on GRMS. This heuristic starts the first job from the waiting queue whenever enough idle resources are available.

Adaptive Decision Making

The current state of the system is crucial when deciding on whether to accept or decline foreign workload, e.g., allowing for additional remote jobs, if the local system is already highly loaded, seems to be inappropriate.

Dependent on the current system state, the load decision maker has to decide whether to accept an offered job or not. Thus, represent the acceptance of a job by an output value of 1 and the corresponding refusal of a job by _1.

Load Delegation

To test the robustness of the pair wise learned rule bases, apply to the 6-month workloads within the same setups. To this end, only two site grids are considered and every partner applies its egoistically learned rule base. Note that AWF is not used in these scenarios anymore. The changes in AWRT and SA are depicted when both partners applied their learned rule bases to be previously unknown job submissions.

Obviously, the Evolutionary Load Systems still decrease the AWRT significantly in all cases. This indicates a high robustness with respect to submission changes. The AWRT improvements in comparison to the AWF transfer policy and in additionally to near optimal solutions. Although AWF is a quite naïve exchange method, it is sufficient for the training purpose. However, for the matter of comparison, it is more meaningful to compare this approach against the best achievable solutions. Unfortunately, there exists no such algorithm that can generate optimal solutions under the given information restrictions. Thus, apply a request-based exchange method (OPT) that actually relaxes all information restrictions and allows also backfilling of jobs in the queues.

CHAPTER 3

RESULTS

3.1 IMPLEMENTATION

Dynamic load balancing algorithms make changes to the distribution of work among workstations at a run-time; use current or recent load information when making distribution decisions. Multicomputer with dynamic load balancing allocates/reallocate resources at runtime based on a priori task information, which may determine when and whose tasks can be migrated. Zero configuration is a set of techniques that automatically creates a usable Internet Protocol (IP) network without manual operator's intervention or special configuration servers. It is known as Wireless Auto Configuration or WLAN Auto Configuration.As a result, wireless auto configuration can provide a significant improvement in Performance over other algorithms. To apply middleware, this provides a high level abstraction, self configuration, and guarantees the interoperability.

3.2 Screen Shots

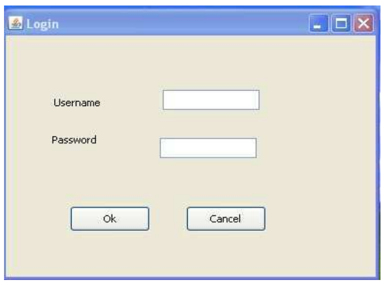
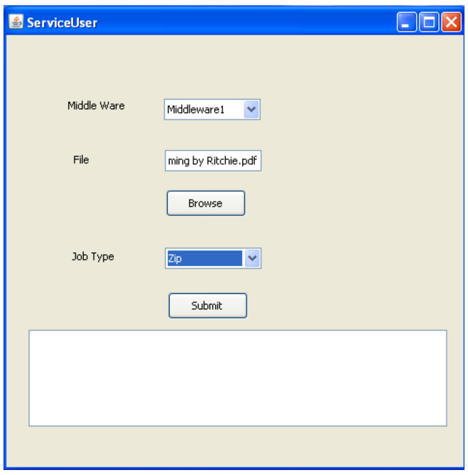


Fig.3.1 User Login

Login Successful

Fig.3.2 Acknowledgement

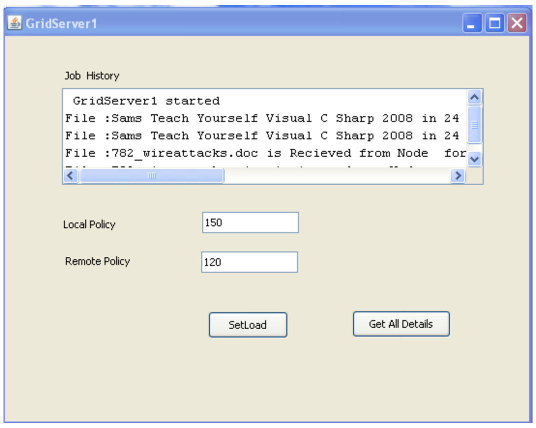

ServiceUser

Middle Ware: Middleware1

File: ming by Ritchie.pdf

Job Type: Zip

Fig.3.3 User   Mail Window


GridServer1

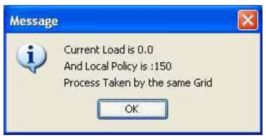
Job History

```
 GridServer1 started
File :Sams Teach Yourself Visual C Sharp 2008 in 24
File :Sams Teach Yourself Visual C Sharp 2008 in 24
File :782_wireattacks.doc is Recieved from Node  for
```

Local Policy: 150

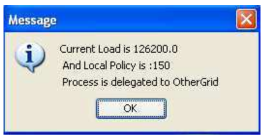
Remote Policy: 120

Fig.3.4 Grid server


Message

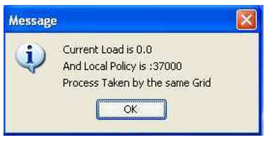
Current Load is 0.0
And Local Policy is :150
Process Taken by the same Grid

Fig.3.5 Policy and Load value


Message

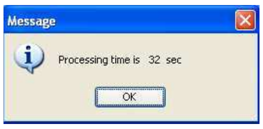
Current Load is 126200.0
And Local Policy is :150
Process is delegated to OtherGrid

Fig.3.6 Job Delegate to other Middleware


Message

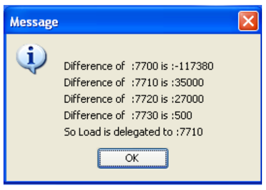
Current Load is 0.0
And Local Policy is :37000
Process Taken by the same Grid

Fig.3.7 Accept Remote Job


Message

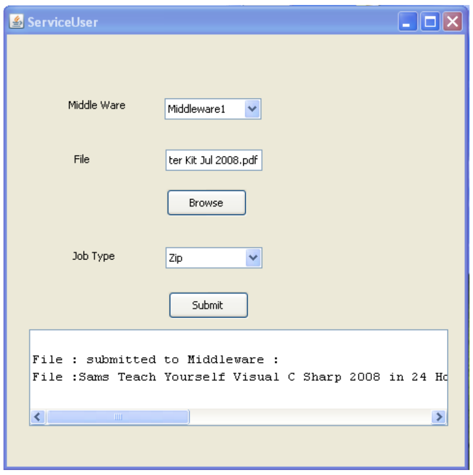
Processing time is   32 sec

Fig.3.8 Processing Time


Message

Difference of :7700 is :-117380
Difference of :7710 is :35000
Difference of :7720 is :27000
Difference of :7730 is :500
So Load is delegated to :7710

Fig.3.9 Load Delegation


ServiceUser

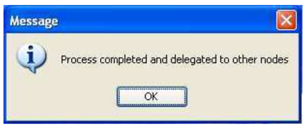
Middle Ware: Middleware1

File: ter Kit Jul 2008.pdf

Job Type: Zip

```
File : submitted to Middleware :
File :Sams Teach Yourself Visual C Sharp 2008 in 24 Ho
```

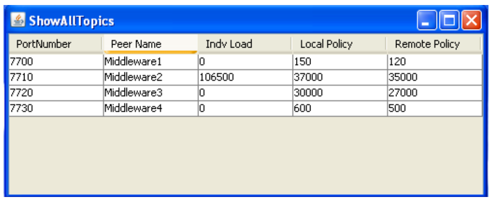
Fig.3.10 Job complete


Message

Process completed and delegated to other nodes

OK

Fig.3.11 Job Complete Acknowledgement


ShowAllTopics

PortNumber	Peer Name	Indv Load	Local Policy	Remote Policy
7700	Middleware1	0	150	120
7710	Middleware2	106500	37000	35000
7720	Middleware3	0	30000	27000
7730	Middleware4	0	600	500

Fig.3.12 Get Details

3.3 DATABASE DESIGN

This table shows the node details such as node name, IP address, port number, group, status.Table.1 Node Details

Nodename	IPAddress	Port	Group	Status
Middleware1	192.68.1.6	7700	G1	Connect
Middleware2	192.68.1.7	7710	G1	Connect
Middleware3	192.68.1.6	7720	G1	Connect
Middleware4	192.68.1.7	7730	G1	Connect

This table shows the policy details such as group number, peer name, individual load, local policy, remote policy.

Table.2 Policy Details

GroupNo	Peername	IndvLoad	Loc_pol	Rem_pol
7700	Middleware1	0	150	120
7710	Middleware2	0	37000	25000
7720	Middleware3	0	30000	27000
7730	Middleware4	0	600	500

3.4 CONCLUSION AND FUTURE WORK

Grid application performance remains a challenge in dynamic grid environment. Dynamic load balancing algorithm allocates the tasks to the particular processors, reschedule the jobs if load is not balanced and get equal shares of the resources. In this project, the zero configuration scheduling is used to schedule the job. Here, based on the current system state, load decision maker decides whether to accept the job or not. If the peer is highly loaded, the load decision maker compares the transfer policy in addition to near optimal solution. This improves the performance based on zero configuration scheduling. In future, QoS constrains such as reliability can used as performance measure.

APPENDIX

SAMPLE SOURCE CODE

Middleware

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Timer;
public class MiddleWares extends JFrame implements Runnable
{
private JLabel jLLocalPolicy1;
private JLabel jLRemotePolicy;
private JLabel jLJObHistory;
private JLabel jLBackGround;
private JTextField jTF_LP;
private JTextField jTF_RP;
private JTextArea jTALoadHistory;
private JScrollPane jScrollPane1;
private JButton jBSetLoad;
private JButton jBGetAll;
private JPanel contentPane;
final static int serPort = 7700;
Socket soc,soc1;
ServerSocket ss;
ObjectInputStream ois;
ObjectOutputStream oos;
String name="GridServer1";
int setPort;
GRMS rp = new GRMS(serPort);
public MiddleWares()
{
super();
initializeComponent();
Thread t= new Thread(this);
t.start();
initializeComponent call
try{
rp.node_no = serPort;
DBCon db=new DBCon();
```

```
db.getPolicies(serPort);
String rp=String.valueOf(db.rp);
String lp=String.valueOf(db.lp);
jTF_LP.setText(lp);
jTF_RP.setText(rp);
}catch(Exception n){
n.printStackTrace();
}
this.setVisible(true);
}
private void initializeComponent()
{
jLLocalPolicy1 = new JLabel();
jLRemotePolicy = new JLabel();
jLJObHistory = new JLabel();
jLBackGround = new JLabel();
jTF_LP = new JTextField();
jTF_RP = new JTextField();
jTALoadHistory = new JTextArea();
jScrollPane1 = new JScrollPane();
jBSetLoad = new JButton();
jBGetAll = new JButton();
contentPane = (JPanel)this.getContentPane();
jLLocalPolicy1.setText("Local Policy");
jLRemotePolicy.setText("Remote Policy");
jLJObHistory.setText("Job  History");
jLBackGround.setIcon(new ImageIcon("images//icon1.png"));
jTF_LP.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jTF_LP_actionPerformed(e);
}
});
jTF_RP.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jTF_RP_actionPerformed(e);
}
});
jScrollPane1.setViewportView(jTALoadHistory);
jBSetLoad.setText("SetLoad");
jBSetLoad.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jBSetLoad_actionPerformed(e);
}
});
jBGetAll.setText("Get All Details");
jBGetAll.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jBGetAll_actionPerformed(e);
}
});
contentPane.setLayout(null);
addComponent(contentPane, jLLocalPolicy1, 61,186,105,18);
addComponent(contentPane, jLRemotePolicy, 63,224,110,18);
addComponent(contentPane, jLJObHistory, 62,32,160,18);
addComponent(contentPane, jTF_LP, 204,182,100,22);
addComponent(contentPane, jTF_RP, 203,223,100,22);
addComponent(contentPane, jScrollPane1, 59,54,436,100);
addComponent(contentPane, jBSetLoad, 210,285,83,28);
addComponent(contentPane, jBGetAll, 359,284,102,28);
addComponent(contentPane, jLBackGround, 0,-20,555,442);
this.setTitle("GridServer1 ");
this.setLocation(new Point(650, 0));
this.setSize(new Dimension(550, 433));
this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
private void addComponent(Container container,Component c,int x,int y,int
width,int height)
{
c.setBounds(x,y,width,height);
container.add(c);
}
private void jTF_LP_actionPerformed(ActionEvent e)
{
System.out.println("\njTF_LP_actionPerformed(ActionEvent e) called.");
}
private void jTF_RP_actionPerformed(ActionEvent e)
{
System.out.println("\njTF_RP_actionPerformed(ActionEvent e) called.");
}
private void jBSetLoad_actionPerformed(ActionEvent e)
{
System.out.println("\njBSetLoad_actionPerformed(ActionEvent e) called.");
int lp=Integer.parseInt(jTF_LP.getText());
int rp=Integer.parseInt(jTF_RP.getText());
if(rp<lp)
{
```

```
DBCon dc=new DBCon();
dc.setPolices(lp,rp,serPort);
JOptionPane.showMessageDialog(this,"Inserted Successfully");
}
else
JOptionPane.showMessageDialog(null,"Remote policy should be less than Local
policy");
}
private void jBGetAll_actionPerformed(ActionEvent e)
{
System.out.println("\njBGetAll_actionPerformed(ActionEvent e) called.");
new GetAllDetails();
}
public  void run()
{
try{
Socket soc1;
ServerSocket ss;
ObjectInputStream ois1;
ObjectOutputStream oos1;
String pro="",filenam="";
String zipfile="";
String unzipfile="";
String zipFileNam="";
int clientPort = 0;
byte[] data;
ss=new ServerSocket(serPort);
System.out.println("Server1 Started in :"+serPort);
jTALoadHistory.append(" GridServer1 started");
Timer timer = new Timer();
timer.schedule(rp, 1000, 1000);
while(true)
{
int subPort = 0;
double load=0.0;
soc1=ss.accept();
ois1=new ObjectInputStream(soc1.getInputStream());
String msg =(String) ois1.readObject();
if( msg.equals("delegate")){
clientPort=(Integer)ois1.readObject();
pro= (String)ois1.readObject();
filenam =(String)ois1.readObject();
data=(byte[])ois1.readObject();
jTALoadHistory.append("\nFile :"+filenam+" is Recieved from Node  for process :
"+pro);
```

```
load = data.length/100;
System.out.println("Load of the file is :"+load);
subPort = rp.process(load,clientPort);
if(subPort == 0)
{
FileOutputStream fos = new FileOutputStream("received\\"+filenam);
fos.write(data);
fos.flush();
if(pro.equals("Zip"))
{
System.out.println( " in side zip");
MyZip mz = new MyZip(filenam);
mz.doZip(filenam);
zipfile=mz.outFilename;
zipFileNam=mz.zipFileNam;
}
else{
System.out.println( " in side zip");
MyUNzip uz=new MyUNzip(filenam);
uz.doUnzip();
unzipfile=uz.outFilename;
zipFileNam=uz.unzipfile;
}
File fi= new File(zipfile);
if(fi.exists())
{
System.out.println("File Exists");
FileInputStream fis = new FileInputStream(fi);
data=new byte[fis.available()];
fis.read(data);
fis.close();
}
fos.close();
System.out.println("ZipFile :"+zipfile);
}
else
{
Socket socSub;
DBCon.setHost(subPort);
String subHost=DBCon.host;
System.out.println("Host :"+subHost +"subPort :"+subPort);
socSub=new Socket(subHost,subPort);
oos=new ObjectOutputStream(socSub.getOutputStream());
oos.writeObject("delegate");
oos.writeObject(serPort);
```

```
oos.writeObject(pro);
oos.writeObject(filenam);
oos.writeObject(data);
ois = new ObjectInputStream(socSub.getInputStream());
zipFileNam=(String)ois.readObject();
data=(byte[])ois.readObject();
}
oos1=new ObjectOutputStream(soc1.getOutputStream());
System.out.println("ZipFile :"+zipfile);
oos1.writeObject(zipFileNam);
oos1.writeObject(data);
oos1.flush();
jTALoadHistory.append("\nFile :"+zipFileNam+".zip is send to  Node  as :
"+pro+"file");
DBCon.update(filenam,name);
oos1.close();  ois1.close();
}
else if( msg.equals("same")){
filenam =(String)ois1.readObject();
data=(byte[])ois1.readObject();
pro = (String)ois1.readObject();
FileOutputStream fos = new
FileOutputStream("received\\"+filenam);fos.write(data);
fos.flush();
if(pro.equals("Zip"))
{
System.out.println( " in side zip");
MyZip mz = new MyZip(filenam);
mz.doZip(filenam);
zipfile=mz.outFilename;
zipFileNam=mz.zipFileNam;
}
Else
{
System.out.println( " in side zip");
MyUNzip uz=new MyUNzip(filenam);
uz.doUnzip();
unzipfile=uz.outFilename;
zipFileNam=uz.unzipfile;
}
File fi= new File(zipfile);
if(fi.exists()){
System.out.println("File Exists");
FileInputStream fis = new FileInputStream(fi);
data=new byte[fis.available()];
```

```
fis.read(data);
fis.close();
 }
fos.close();
System.out.println("ZipFile :"+zipfile);
oos1=new ObjectOutputStream(soc1.getOutputStream());
System.out.println("ZipFile :"+zipfile);
oos1.writeObject(zipFileNam);
oos1.writeObject(data);
oos1.flush();
jTALoadHistory.append("\nFile :"+zipFileNam+".zip is send to  Node  as :
"+pro+"file");
DBCon.update(filenam,name);
oos1.close();  ois1.close();
}
}
}
catch(Exception e)
{
e.printStackTrace();
}
}
public static void main(String[] args)
{
JFrame.setDefaultLookAndFeelDecorated(true);
JDialog.setDefaultLookAndFeelDecorated(true);
try
{
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAnd
Feel");
}
catch (Exception ex)
{
System.out.println("Failed loading L&F: ");
System.out.println(ex);
}
new MiddleWares();
}
}
```

Service user

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

```
import java.util.Timer;
import java.net.*;
import java.sql.*;
public class ServiceUser extends JFrame
{
private JLabel jLMidWare;
private JLabel jLFile;
private JLabel jLJobType;
private JTextField jTF_FileName;
private JComboBox jCBMidWare;
private JComboBox jCBJobType;
private JTextArea jTAWorkDetail;
private JScrollPane jScrollPane1;
private JButton jBBrowse;
private JButton jBSubmit;
private JLabel jLBackGround;
private JPanel contentPane;
Socket soc;
Socket soc1;
ServerSocket ss;
ObjectInputStream ois;
ObjectOutputStream oos;
final static int serPort=7700;

String fn, fpath;
String job="";
String midWare="";
public ServiceUser()
{
super();
initializeComponent();
this.setVisible(true);
}
private void initializeComponent()
{
String[]
combo1_str={"Select","Middleware1","Middleware2","Middleware3","Middleware
4"};
String[]  combo2_str={"Select Job","Zip","UnZip"};
jLMidWare = new JLabel();
jLFile = new JLabel();
jLJobType = new JLabel();
jTF_FileName = new JTextField();
jCBMidWare = new JComboBox(combo1_str);
jCBJobType = new JComboBox(combo2_str);
```

```
jTAWorkDetail = new JTextArea();
jScrollPane1 = new JScrollPane();
jBBrowse = new JButton();
jBSubmit = new JButton();
jLBackGround = new JLabel();
contentPane = (JPanel)this.getContentPane();
jLMidWare.setText("Middle Ware");
jLFile.setText("  File");
jLJobType.setText("Job Type");
jTF_FileName.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jTF_FileName_actionPerformed(e);
}
});
jCBMidWare.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jCBMidWare_actionPerformed(e);
}
});
jCBJobType.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jCBJobType_actionPerformed(e);
}
});
jScrollPane1.setViewportView(jTAWorkDetail);
jBBrowse.setText("Browse");
jBBrowse.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jBBrowse_actionPerformed(e);
}
});
jLBackGround.setIcon(new ImageIcon("images//icon1.png"));
jBSubmit.setText(" Submit ");
jBSubmit.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)
{
jBSubmit_actionPerformed(e);
}
});
contentPane.setLayout(null);
addComponent(contentPane, jLMidWare, 64,64,60,18);
```

```
addComponent(contentPane, jLFile, 64,120,60,18);
addComponent(contentPane, jLJobType, 68,220,60,18);
addComponent(contentPane, jTF_FileName, 164,119,100,22);
addComponent(contentPane, jCBMidWare, 163,66,100,22);
addComponent(contentPane, jCBJobType, 164,220,100,22);
addComponent(contentPane, jScrollPane1, 23,305,433,100);
addComponent(contentPane, jBBrowse, 165,160,83,28);
addComponent(contentPane, jBSubmit, 167,266,83,28);
addComponent(contentPane, jLBackGround, 0,-10,555,450);
this.setTitle("ServiceUser");
this.setLocation(new Point(250,100));
this.setSize(new Dimension(481,481));
this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
private void addComponent(Container container,Component c,int x,int y,int
width,int height)
{
c.setBounds(x,y,width,height);
container.add(c);
}
private void jTF_FileName_actionPerformed(ActionEvent e)
{
System.out.println("\njTF_FileName_actionPerformed(ActionEvent e) called.");
}
private void jCBMidWare_actionPerformed(ActionEvent e)
{
Object o = jCBMidWare.getSelectedItem();
midWare=o.toString();
System.out.println(">>" + ((o==null)? "null" : o.toString()) + " is selected.");
}
private void jCBJobType_actionPerformed(ActionEvent e)
{
Object o = jCBJobType.getSelectedItem();
job=o.toString();
System.out.println(">>" + ((o==null)? "null" :o) + " is selected.");
}
private void jBBrowse_actionPerformed(ActionEvent e)
{
JFileChooser file=new JFileChooser();
int val=file.showOpenDialog(this);
if(val==JFileChooser.APPROVE_OPTION){
fn=file.getSelectedFile().getName();
fpath=file.getSelectedFile().getPath().toString();
jTF_FileName.setText(fpath);
System.out.println(">>  Path   :"+fpath+" >>--File name :"+fn);
```

```
}
}
private void jBSubmit_actionPerformed(ActionEvent e)
{
String error="";
String tf=jTF_FileName.getText();
Object job = (String)jCBJobType.getSelectedItem();
Object midWare = (String)jCBMidWare.getSelectedItem();
if(tf.equals(""))
{
error="Browse  File and Retry\n";
}
if(job.equals("Select Job"))
{
error+="Choose  any Job and Retry\n";
}
if(midWare.equals("Select"))
{
error=error+"Choose a  Node and Retry";
}
System.out.println(" ok"+error
if(error.length()<2)
{
if(DBCon.checkDB(fn))
{
System.out.println("trueeeeeeeeksj;lakdjgal;j;k,fgbmdf,.");
ArrayList list=DBCon.getpeerdetail(fn);
sendSame( list,job );
}
else
{
Send( midWare, job );
}
}
else
JOptionPane.showMessageDialog(null,error);
}
void Send(Object oN,Object oP)
{
try{
jTAWorkDetail.append("\nFile : submitted to Middleware :");
new DBCon();
DBCon.setDetailOf(oN);
String host=DBCon.host;
int port=DBCon.port;
```

```
if(port == 0)
JOptionPane.showMessageDialog(null,"The  MiddleWare is not available
"+host,"Please Choose Another"+port,JOptionPane.ERROR_MESSAGE);
Else
{
System.out.println(" host "+host+"Port :"+port);
FileInputStream fis = new FileInputStream(fpath);
byte[] data=new byte[fis.available()];
fis.read(data);
soc=new Socket(host,port);
oos=new ObjectOutputStream(soc.getOutputStream());
oos.writeObject("delegate");
oos.writeObject(serPort);
oos.writeObject(oP);
oos.writeObject(fn);
oos.writeObject(data);
ois = new ObjectInputStream(soc.getInputStream());
String zipnam=(String)ois.readObject();
System.out.println("Returned File :"+zipnam);
data=(byte[])ois.readObject();
FileOutputStream fos = new FileOutputStream("middleware//"+zipnam);
fos.write(data);
fos.flush();
fis.close();
fos.close();
oos.close();
 ois.close();
jTAWorkDetail.append("\nFile :"+zipnam+" is Received form MiddleWare "+oN+"
as : "+oP+"file");
}
}catch(Exception e){
e.printStackTrace();
}
}
void sendSame( ArrayList list,Object oP )
{
try
{
FileInputStream fis = new FileInputStream(fpath);
byte[] data=new byte[fis.available()];
fis.read(data);
soc=new Socket(list.get(0).toString(),Integer.parseInt((String)list.get(1)));
oos=new ObjectOutputStream(soc.getOutputStream());
oos.writeObject("same");
oos.writeObject( fn );
```

```
oos.writeObject( data );
oos.writeObject( oP );
ois = new ObjectInputStream(soc.getInputStream());
String zipnam=(String)ois.readObject();
System.out.println("Returned File  our Method :"+zipnam);
data=(byte[])ois.readObject();
FileOutputStream fos = new FileOutputStream("middleware//"+zipnam);
fos.write(data);
fos.flush();
fis.close();
fos.close();
oos.close();
ois.close();
jTAWorkDetail.append("\nFile :"+zipnam+" is Received from SameMiddleware as
:"+oP+"file");
}
catch(Exception e)
{
e.printStackTrace();
}
}
```

REFERENCES

1. Brian Towles, William J. Dally, Fellow," Guaranteed Scheduling For Switches With Configuration Overhead", IEEE/ACM Transactions on Networking, Vol. 11, No. 5, Octomber 2003.

2. Daphne Lopez, S. V. Kasmir raja," A Dynamic Error Based Fair Scheduling Algorithm for a Computational Grid", Journal of Theoretical and Applied Information Technology - 2009 JATIT.

3. Doulamis, N.D.; Doulamis, A.D.; Varvarigos, E.A.; Varvarigou, T.A "Fair Scheduling Algorithms in Grids" IEEE Transactions on Parallel and Distributed Systems, Volume18, Issue 11, Nov. 2007 Page(s):1630 – 1648.

4. Foster, I., and Kesselman, C. (editors), "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, USA, 1999.

5. K.Somasundaram, S.Radhakrishnan," Task Resource Allocation in Grid using Swift Scheduler", International Journal of Computers, Communications & Control, ISSN 1841-9836, EISSN 1841-9844 Vol. IV, 2009.

6. Kyeong-Deok Moon, Young-Hee Lee, and Young-Sung Son, Chae-Kyu Kim, "Universal Home Network Middleware Guaranteeing Seamless Interoperability among the Heterogeneous Home Network Middleware", IEEE Transactions on Consumer Electronics, Vol. 49, No. 3, AUGUST 2003.

7. M.Kamarunisha, S.Ranichandra, T.K.P.Rajagopal, "Recitation of Load Balancing Algorithms In Grid Computing Environment Using Policies And Strategies An Approach," International Journal of Scientific & Engineering Research Volume 2, Issue 3, March- 2011.

8. Pal Nilsson1 and Michał Pi´oro," Unsplittable max-min demand allocation a routing problem".

9. Prabhat Kr.Srivastava, Sonu Gupta, Dheerendra Singh Yadav," Improving Performance In Load Balancing Problem On The Grid Computing System", International Journal of Computer Applications (0975 – 8887) Volume 16– No.1, February 2011.

10. R. Buyya, D. Abramson, and J.Giddy Nimro," An Architecture for a Resource Management and Scheduling System in a Global Computational Grid," Proc. Fourth Int'l Conf. High Performance Computing in Asia- Pacific Region, 2000.

11. Rajkumar Buyya, David Abramson, and Jonathan Giddy," A Case for Economy Grid Architecture for Service Oriented Grid Computing ".

12. Saravana kumar E. and Gomathy Prathima," A novel load balancing algorithm for computational grid," International Journal of Computational Intelligence Techniques, ISSN: 0976–0466 & EISSN:0976–0474 Volume 1, Issue 1, 2010, PP-20-26.

13. U. Karthick Kumar," A Dynamic Load Balancing Algorithm in Computational Grid Using Fair Scheduling", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 5, No 1, September 2011 ISSN (Online): 1694-0814.

LIST OF PUBLICATIONS

1. M. Yogadharani, P. Devaki, "A Dynamic Load Balancing Algorithm In Computational Grid Using Zero Configuration Scheduling" International Conference, Advanced Computing, Machines and Embedded Technology, J.K.K.Nattraja College of Engineering and Technology, komarapalayam, 8th and 9th March 2013.

2. M. Yogadharani, P. Devaki, "Implementation of Auto Configuration Scheduling In Computational Grid Using Dynamic Load Balancing Algorithm", International Journal of Technical Journals/Google Journals, May 2013.