RESOURCE ALLOCATION IN THE GRID WITH LEARNING AGENTS

A Project Report

Submitted by

Purushothman.M. - 71203104026
Dakshnamoorthy.P. - 71203104300
Anandh.S. - 71203104304

in partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE - 641006

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2007

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project work entitled "**RESOURCE ALLOCATION IN THE GRID WITH LEARNING AGENTS" is** the bonafide work of

Purushothman.M	-	71203104026
Dakshnamoorthy.P	-	71203104300
Anandh.S	-	71203104304

Who carried out the project work under my supervision.

SIGNATURE

Dr.S.Thangasamy

HEAD OF THE DEPARTMENT

Department of Computer Science
And Engineering,
Kumaraguru College of
Technology
Chinnavedampatti P.O.,
Coimbatore-641006

SIGNATURE

Mrs.S.Devaki

SUPERVISOR

Department of Computer Science
And Engineering,
Kumaraguru College
of Technology
Chinnavedampatti P.O.,
Coimbatore-641006

Submitted for the Viva-Voce Examination held on 24/4/07

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project work entitled "**RESOURCE ALLOCATION IN THE GRID WITH LEARNING AGENTS". **is a record of original work done by us to the best of our knowledge, a similar work has not been submitted to Anna University or any other institution, for fulfillment of the requirement of the course study.

This report is submitted in partial fulfillment of the requirements for the award of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place:Coimbatore

Date: 23/4/07

(Purushothaman.M)

(Dakshnamoorthy.P)

(Anandh.S)

ACKNOWLEDGEMENT

We express our heartful thanks to our beloved correspondent **Dr.K.ARUMUGAM** for his kind patronage in pursuing this project work successfully.

We wish to express our sincere thanks to our Principal **Dr.JOSEPH V.THANIKAL**, for providing support and necessary facilities to carryout this work..

We are highly indebted and convey our most sincere thanks to our Head of the Department **Prof. Dr.S.Thangasamy,** for his invaluable suggestions towards this project.

We deem it a pleasure and privilege to our guide **Mrs.S.Devaki,** Assistant Professor, who is always a constant source of inspiration and encouragement that helped us in the successful completing of this project most effectively.

We also express our heart full thanks to our project coordinator **Ms.S.Rajini**, Senior lecturer, Department of computer science for her help rendered to us in handling various tough spots during this project.

We would like to thank all the **Staff members** of computer science department and all those who have directly or indirectly assisted us in successfully completing this project.

We extend our thanks to our **family and friends** for all their support and rendered to us.

ABSTRACT

Grid computing is an emerging technology that enables users to share a large number of computing resources distributed over a network. We construct a multi-agent model of resource allocation for the Grid that is simplified, yet maintains the main features of the Grid environment: heterogeneity of dynamic, large-scale populations of users and resources. In our system, a large number of users submit jobs to one of the resources that are scheduled by a local scheduler according to local policies. The users are modeled as rational, selfish agents that try to maximize their utilities, (i.e., complete their jobs in the shortest possible time). The agents have no prior knowledge about the computational capabilities of resources. Instead, they utilize a simple reinforcement learning scheme to estimate the efficiency of different resources based on their past experience. Namely, an agent assigns a score to each resource that indicates how well that resource has performed in the past.

After each submitted job, the agent updates the score of the corresponding resource. We analyze the global behavior of the system by numerical simulations, and compare it with a baseline algorithm that makes use of a global knowledge of current resource loads.

LIST OF FIGURES

TABLE OF CONTENTS

1. INTRODUCTION

INTRODUCTION

Our project deals with the overview of grid computing and describes the features in the system that enables the effective utilization of the workstations in the grid. It also explains about the system environment.

1.1 OVERVIEW OF GRID

" Grid " computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and in some cases, high-performance orientation. When grid technology provides the flexibility and control on sharing relationships needed to establish Virtual Organizations, the current Grid implementation are too complex and cannot often be used efficiently by medium scale organizations. Moreover transparency is often not a notable feature of Grid Technology. Here we are presenting a grid architecture that is quite simplified and transparent yet robust enough to be used for performing high complexity problems as compared to other available implementations.

1.1.1 DISTRIBUTED Vs GRID COMPUTING

There are actually two similar trends moving in tandem – distributing computing and grid computing. Depending on the market view, the two either overlap, or distributed computing is a subset of grid computing. Grid computing got its name because it strives for an ideal scenario in which the CPU cycles and storage of millions of systems that could be harnessed by anyone who needs it, similar to the way power companies and their users share the electrical grid.

Grid computing can encompass desktop PCs, but more often than not its focus is on more powerful workstations, servers and even mainframes and super computers working on problems involving huge datasets that can run for days. And grid computing leans more towards dedicated systems, than systems primarily used for other tasks.

Large-Scale distributed computing of the variety we are covering usually refers to a similar concept, but is more geared to pooling the resources of hundreds of thousands of networked end-users PCs, which individually are more limited in their memory and processing power and, whose primary purpose is not distributed computing, but rather serving their user.

1.1.2 BENEFITS OF GRID

- **LOWER COMPUTING COSTS**

On a price-to-performance basis, the Grid platform gets more work done with less administration and budget than dedicated hardware solutions. Depending on the size of the network, the price-for-performance ratio for computing power can be literally improved by an order of magnitude.

- **FASTER PROJECT RESULTS**

The extra power generated by the Grid platform can directly impact an organizations ability to win in the market place by shortening product development cycles and accelerating research and development processes.

- **BETTER PRODUCT RESULTS**

Increased, affordable computing power means not having to ignore promising avenues or solutions because of a limited budget or schedule. The power created by Grid platform can help to ensure a higher quality product by allowing higher resolution testing and results, and can permit an organization to test more extensively prior to product release.

1.2 PROBLEM DEFINITION

Organizations that depend on access to computational power to advance their business objectives often sacrifice or scale back new projects, design ideas, or innovations due to sheer lack of computational bandwidth. Project demands simply outstrip computational power, even if an organization has significant investments in dedicated computing resources.

Even given the potential financial rewards from additional computational access, many enterprises struggle to balance the need for additional computing resources with the need to control costs. Upgrading and purchasing new hardware is a costly proposition, and with the rate of technology obsolescence, it is eventually a losing one. By better utilizing and distributing existing compute resources, grid computing will help alleviate this problem.

This project comprises of three modules:

- Job Request
- Job Scheduling and Allocation by Agents
- Process and Response

1.2.1 JOB REQUEST

In this project, totally three types of modules used:

- User
- Broker or Agent
- Resource

Every system acts as user and resource provider in the grid environment. In the first module, the user is connected to the agent system. After receiving the acknowledgement, the job will be submitted to the agent. Job can be of any type. Here, we are finding factorial for large numbers. This application will be running in a specific port. Using this port the job will be processed.

1.2.2 JOB ALLOCATION BY AGENTS OR BROKERS

Agent is main concept in this project. Agent work by allocating the job in some order. Here priority will be set for each and every resource calculate the total time of previous job request and response. That is start time and end time.

This will be maintained in a database on the server side. It stores the IP address of the resource, time, and priority. This will be updated every time the job is requested and response is sent from resource.

First, the job will be divided depending on the number of systems connected in the grid environment. Then, the job will be allocated by the order of the connection of the system in the grid environment. Next time, remaining job will be allocated depending on the response time of the resource using the priority. Every resource response time will be maintained in agent side. Depending on the response time the job will be allotted again and again.

1.2.3 PROCESS AND RESPONSE

Requested application will be running in resource side in a specific port. We can add more application in resource side using different ports. Here, factorial application will be running in all resources. The requested job will be received from agent, and then this job will be processed there.

1.3 SYSTEM ENVIRONMENT

1.3.1 HARDWARE SPECIFICATION

MINIMUM CONFIGURATION

SERVER

Processor	- Pentium III 440 MHz
RAM	- 120 MB
Hard Disk Space	- 4 GB

CLIENT

Processor	- Pentium III 400 MHz
RAM	- 60 MB

RECOMMENDED

SERVER

Processor	- Pentium IV
RAM	- 256MB
Hard Disk Space	- 10GB

CLIENT

Processor	- Pentium III

1.3.2 SOFTWARE SPECIFICATION

Language	- Java (j2sdk1.4.2_04)
Operating system	- Windows
Front end tool	- Java
Backend tool	- MS Access

SERVICES

- Software Development
- Web Designing
- Web Hosting
- Domain Registration
- Networking products
- MS-Suite
- CRM

1.4 EXISTING SYSTEM

Grid computing is an emerging technology that enables users to share a large number of computing resources distributed over a network. The dynamic, federating nature of Grid policy environments is dominated by virtual organizations (VOs) which associate heterogeneous users and resource providers. Users have resource-consuming activities, or jobs that must be mapped to specific resource providers through a resource allocation mechanism.The resource allocation mechanism may choose among alternate mappings in order to optimize some utility metric, within the bounds permitted by the VO policy environment.

It is envisioned that deployment of Grid technology will grow from its current modest scale to eventually overlay the global Web. It is not known how large individual VOs will be, but it is reasonable to imagine resource sharing among populations with tens of thousands of users and thousands of resources.

Hence, allocation mechanisms need to be highly scalable and robust to localized failures in resources and communication paths. From the perspective of a single VO, the dynamic policy environment can be

1.5 PROPOSED SYSTEM

We construct a multi-agent model of resource allocation for the Grid that is simplified, yet maintains the main features of the Grid environment: heterogeneity of dynamic, large-scale populations of users and resources. In our system, a large number of users submit jobs to one of the resources that are scheduled by a local scheduler according to local policies. The users are modelled as rational, selfish agents that try to maximize their utilities, (i.e., complete their jobs in the shortest possible time). The agents have no prior knowledge about the computational capabilities of resources. Instead, they utilize a simple reinforcement learning scheme to estimate the efficiency of different resources based on their past experience. Namely, an agent assigns a score to each resource that indicates how well that resource has performed in the past.

After each submitted job, the agent updates the score of the corresponding resource. We analyze the global behavior of the system by numerical simulations, and compare it with a baseline algorithm that makes use of a global knowledge of current resource loads. Our results illustrate that reinforcement learning can have a substantial positive effect on the quality of resource allocation in a large scale heterogenous system.

2. SYSTEM ANALYSIS

SYSTEM ANALYSIS

Analysis refers to the process of examining a system with the intent of improving the system through better procedures and methods. System analysis is a process of gathering and interpreting the facts, diagnosing problem and using the information, improvements are recommended to the system. This chapter presents the overview and analysis carried out for the development of this system. It also elucidates the requirements and specification of the system and describes the need for the system.

2.1 SYSTEM REQUIREMENTS

2.1.1 FUNCTIONAL REQUIREMENTS

The proposed system performs the job processing in a Local Area Network (LAN) environment.

The client, broker and resource provider communicate through Transmission Control Protocol/Internet Protocol (TCP/IP) using ports.

2.2 SYSTEM ANALYSIS MODEL

2.2.1 USE CASE DIAGRAM

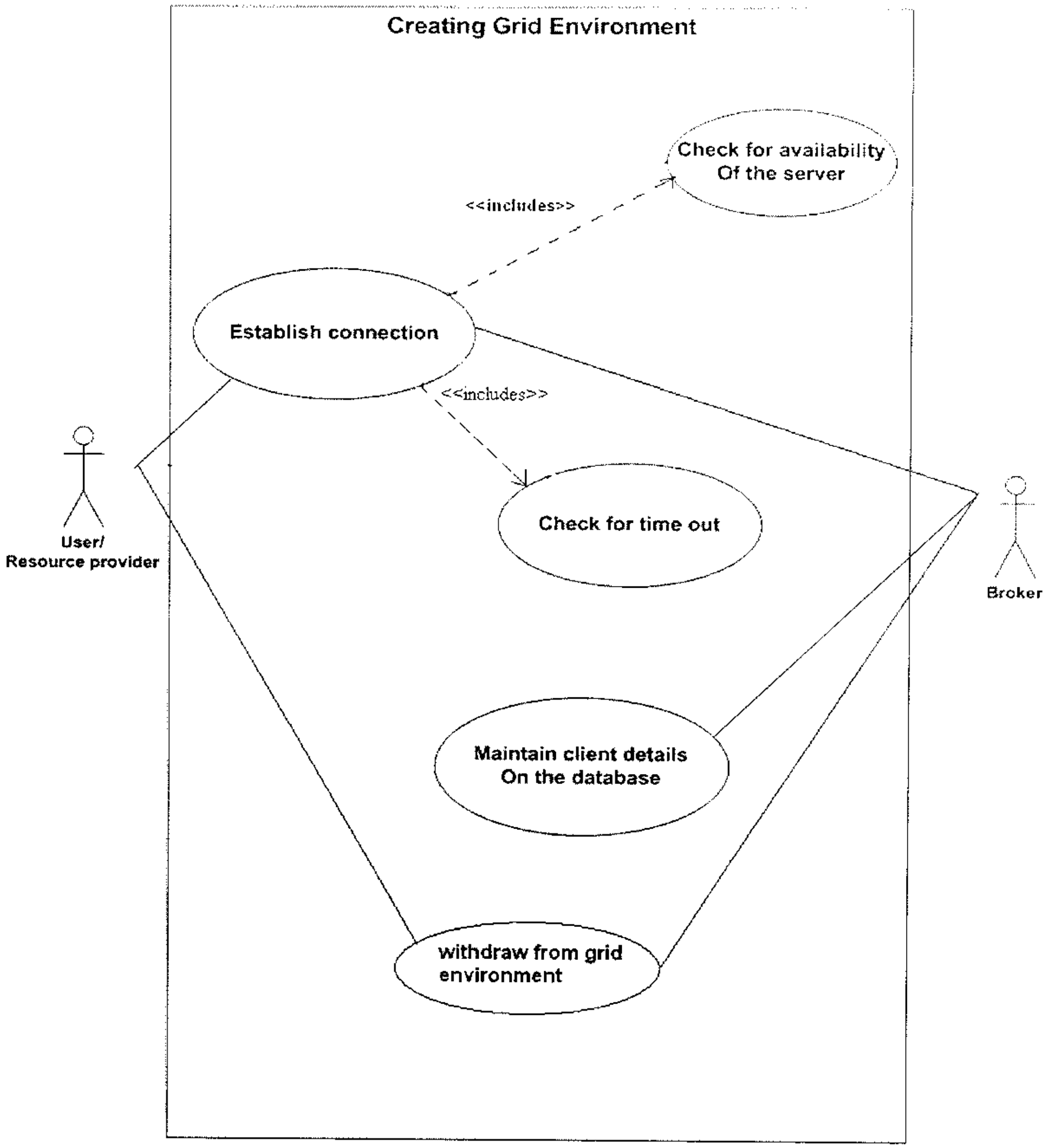


Fig.2.1 Creating Grid Environment

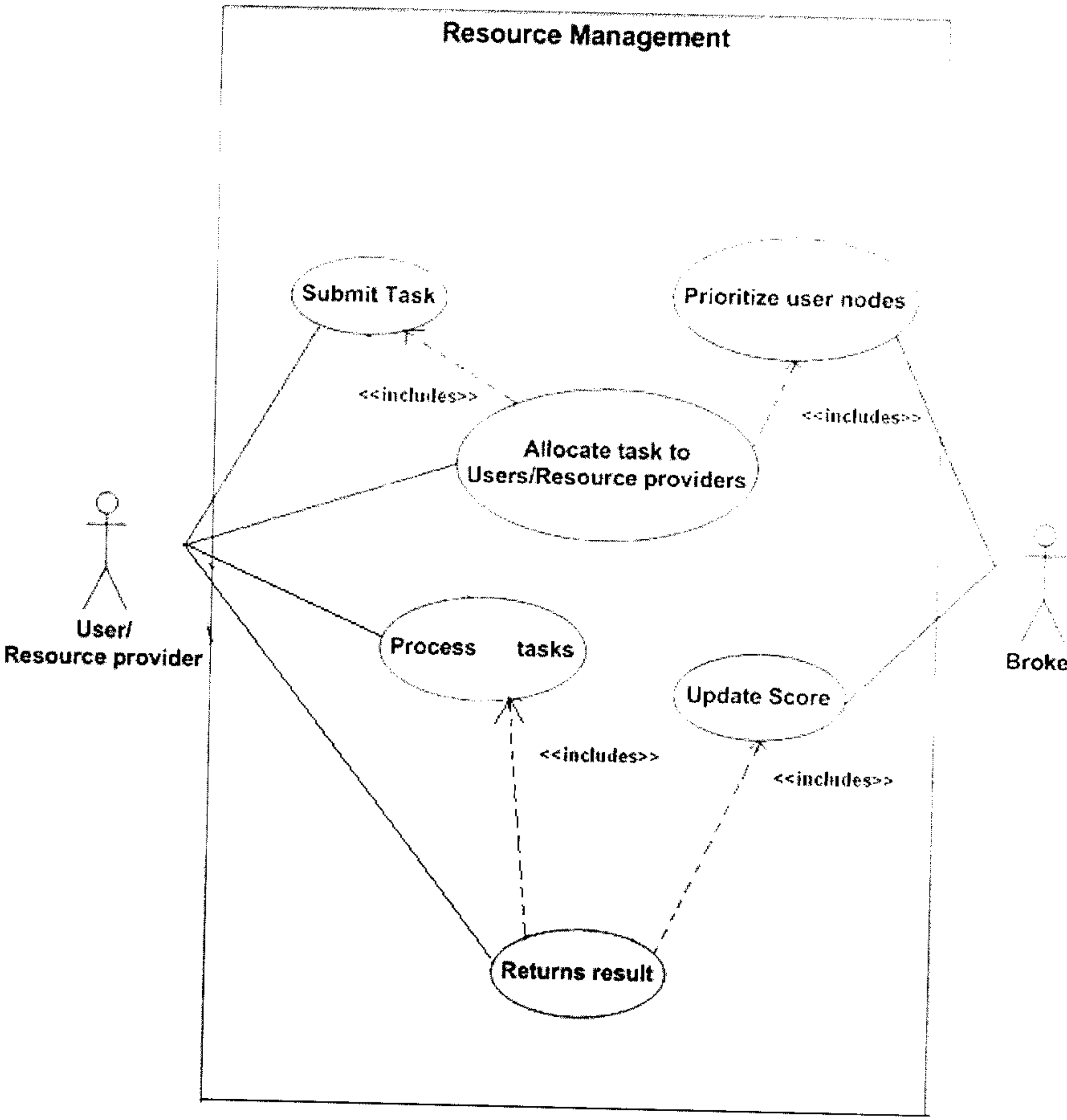


Fig 2.2 Resource Management

2.2.2 ACTIVITY DIAGRAM

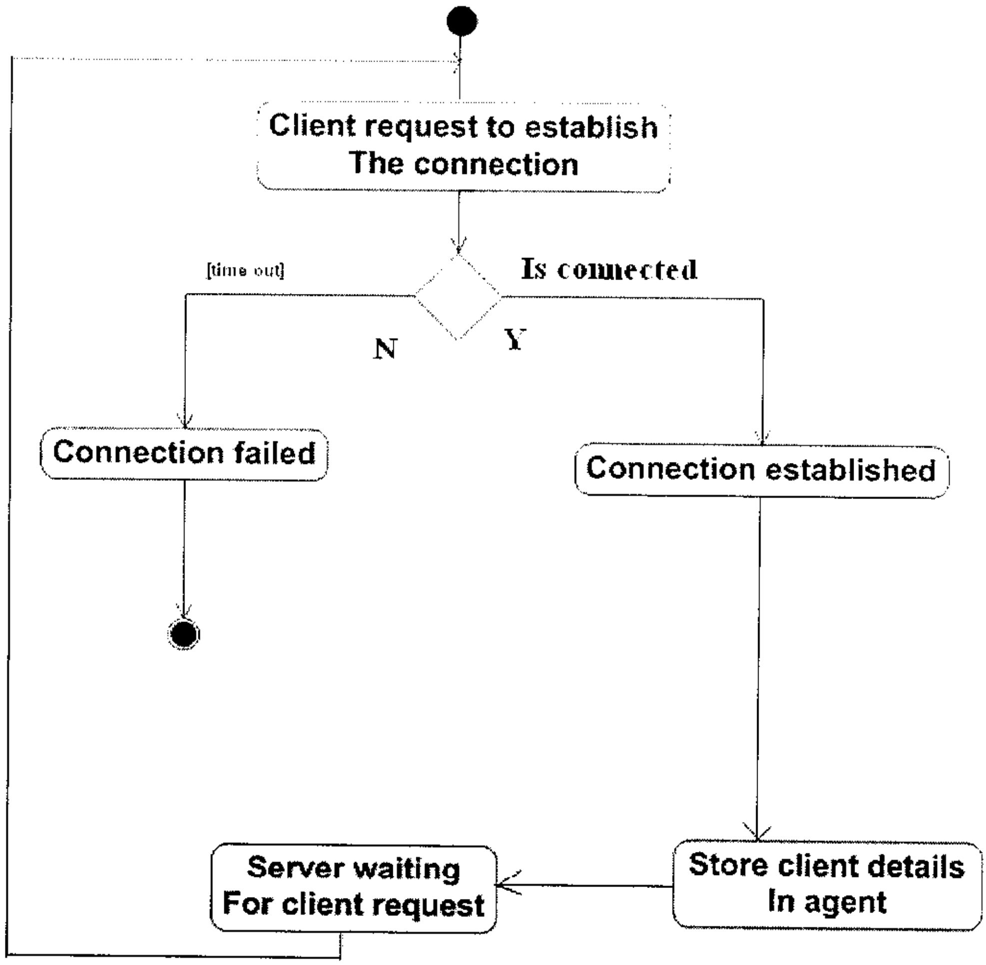
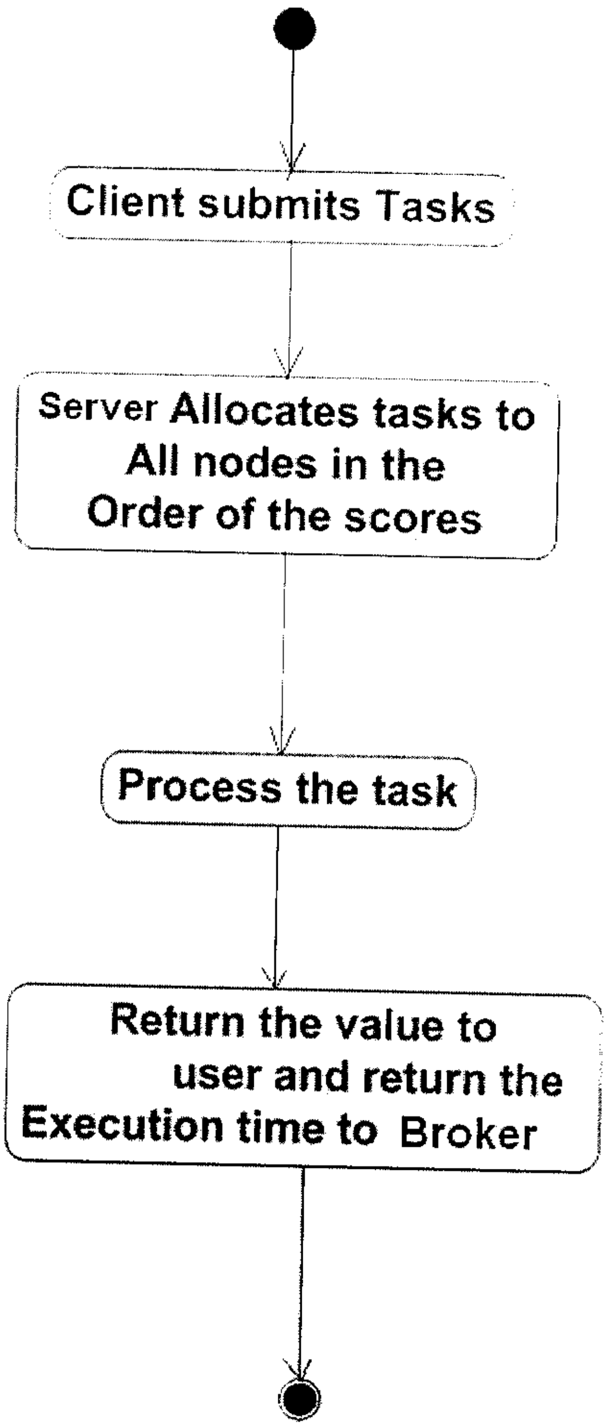


Fig 2.3 The clients establishing connection with the server



Resource Management

2.3 TEST PLAN

2.3.1 UNIT TESTING

A program represents the logical elements of a system. For a program to run satisfactorily, it must compile and test data correctly and tie in properly with other programs. Achieving an error free program is the responsibility of the programmer. Program testing checks for two types of errors: syntax and logical. Syntax error is a program statement that violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax errors. These errors are shown through error message generated by the computer. For Logical errors the programmer must examine the output carefully.

When a program is tested, the actual output is compared with the expected output. When there is a discrepancy the sequence of instructions must be traced to determine the problem. The process is facilitated by breaking the program into self-contained portions, each of which can be checked at certain key points .The idea is to compare program values against desk-calculated values to isolate the problems.

Test case no	Description	Expected result
1	Test for application window properties	All the properties of the windows are to be properly aligned and displayed
2	Test for mouse operations	All the mouse operations like click, drag, etc. must perform the necessary

2.3.2 FUNCTIONAL TESTING

Functional testing of an application is used to prove that application delivers correct results, using enough inputs to give an adequate level of confidence that will work correctly for all sets of inputs. The functional testing will need to prove that the application works for each client type and that personalization functions work correctly.

Test case no	Description	Expected result
1	Test for all clients	All clients should produce the result after execution.
2	Test for various input values	The result after execution should give the accurate result.

2.3.3 NON-FUNCTIONAL TESTING

This testing is used to check that an application will work in the operational environment.

Non-functional testing includes:

- Load testing
- Performance testing
- Usability testing

LOAD TESTING

Test case no	Description	Expected result
1	It is necessary to ascertain that the application behaves correctly under loads when 'server busy' response is received.	Should designate another active node as a Server.

PERFORMANCE TESTING

Test case no	Description	Expected result
1	This is required to assure that an application performed adequately, has the capability to handle any workload, delivers its results in expected time and uses an acceptable level of resource and it is an aspect of operational management.	Should handle large input values, and produce accurate results in an expected time

WHITE BOX TESTING

White box testing, sometimes called glass-box testing is a test case design method that uses the control structure of the procedural design to derive test cases.

Using white box testing method, the software engineer can derive test cases.

Test case no	Description	Expected result
1	Exercise all logical decisions on their true and false sides	All the logical decisions must be valid
2	Execute all loops at their boundaries and within their operational bounds.	All the loops must be finite
3	Exercise internal data structures to ensure their validity.	All the data structures must be valid

BLACK BOX TESTING

Black box testing, also called behavioral testing, focuses on the functional requirements of the software. That is black box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black box testing is not as alternative to white box testing . Rather it is a complementary approach that is likely to uncover a different class of errors than white box methods. Black box testing attempts to find errors in the following categories.

Test case no	**Description**	**Expected result**
1	To check for incorrect or missing functions	All the functions must be valid
2	To check for interface errors	All the interface must function normally
3	To check for errors in a data structure or external data base access.	The database updation and retrieval must be done
4	To check for initialization and termination errors.	All the functions and data structures must be initialized properly and terminated normally

3. SYSTEM DESIGN

SYSTEM DESIGN

3.1 ARCHITECTURAL DESIGN

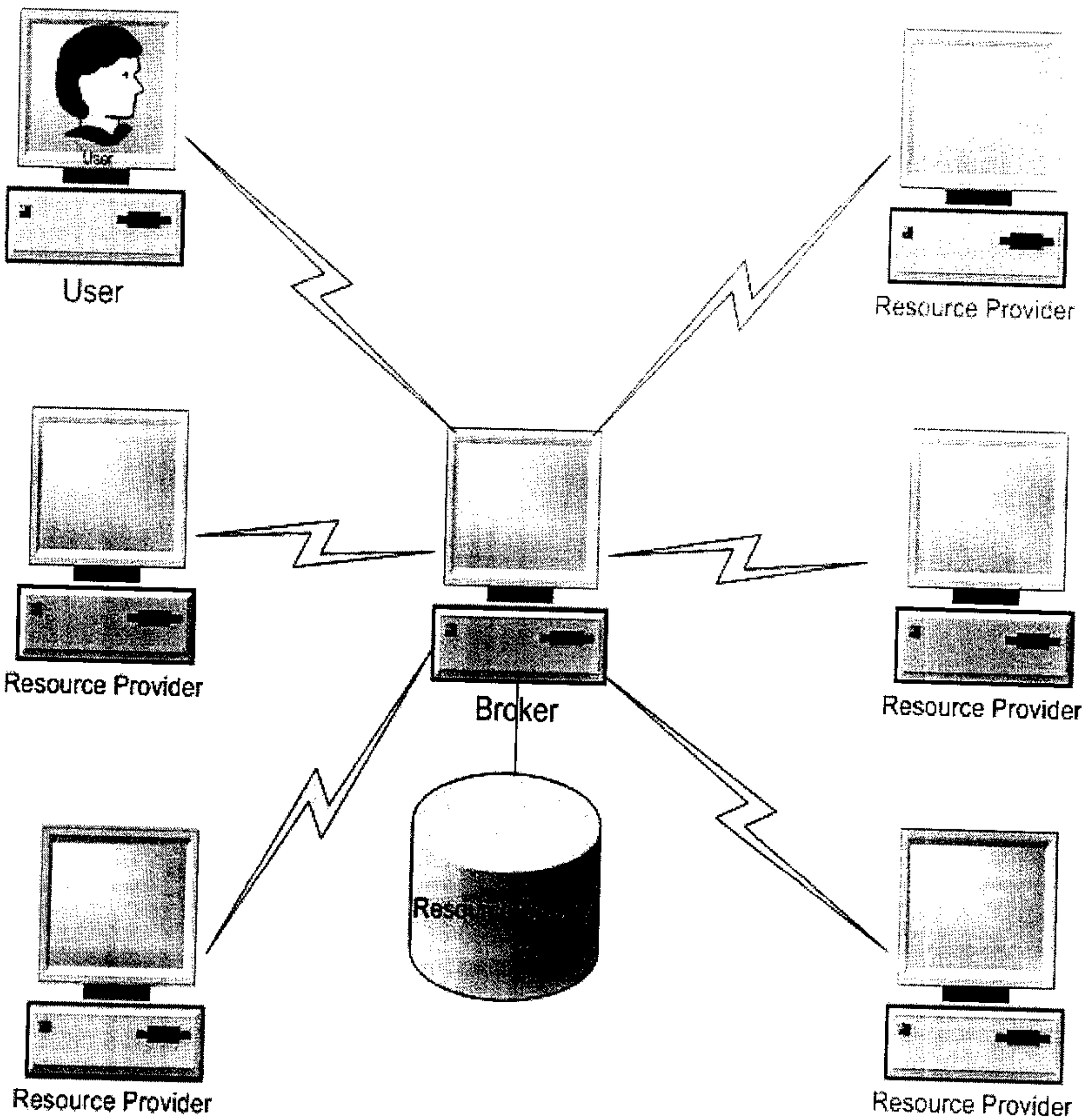
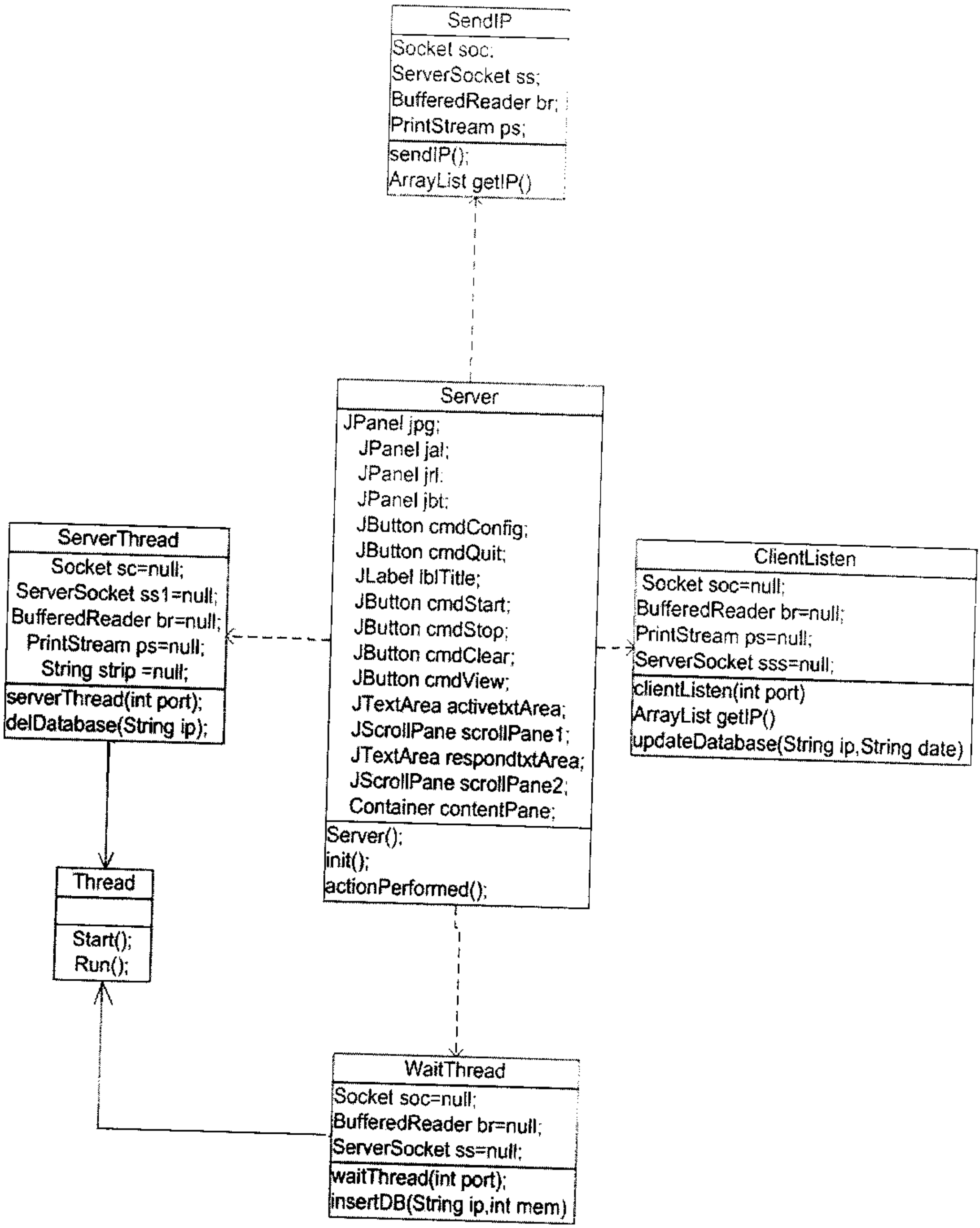


Figure 3.1 Architecture Design

3.2 STRUCTURAL DESIGN

3.2.1 CLASS DIAGRAM



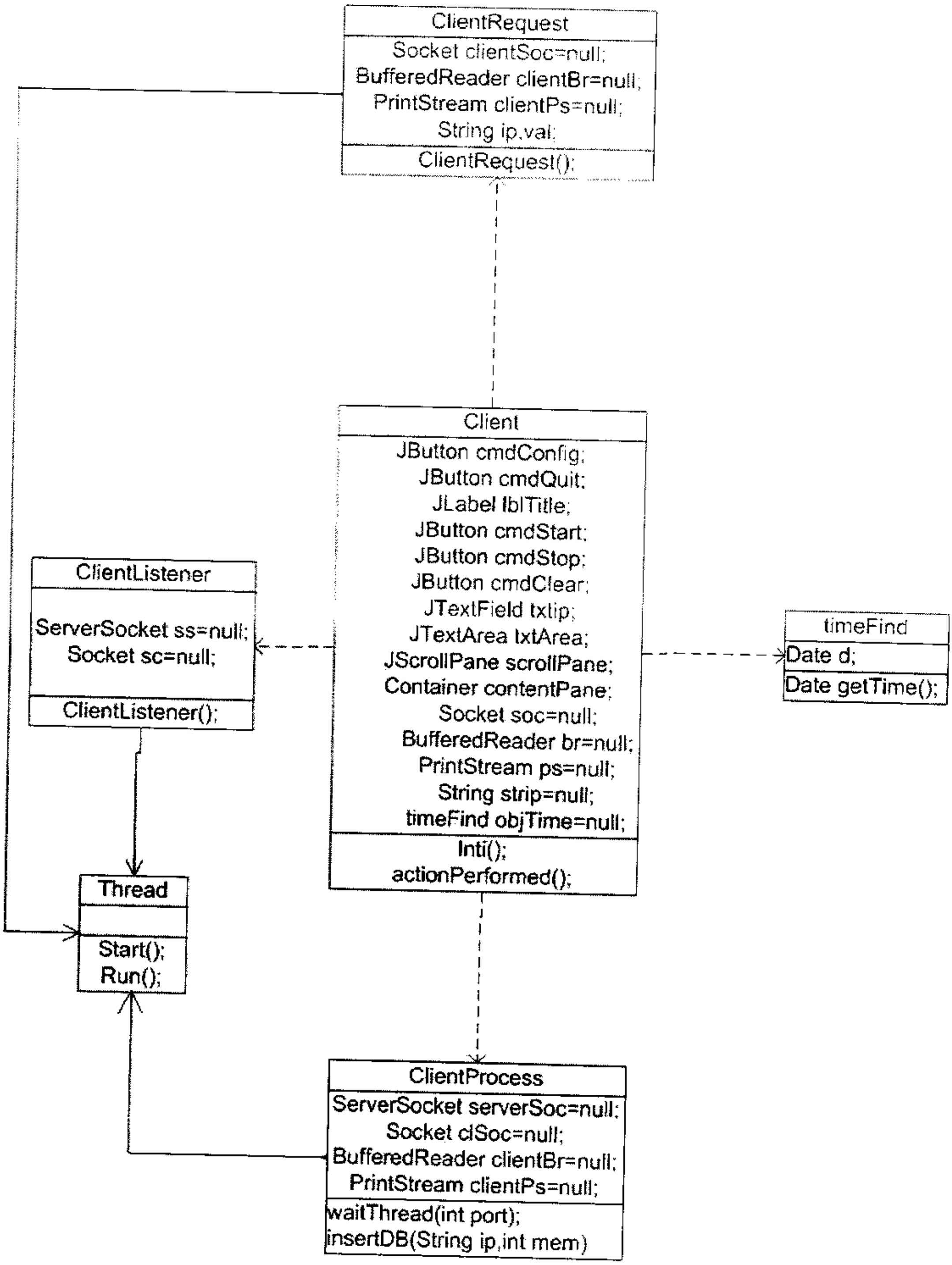


Fig 3.3 Client Class Diagram

3.2 PROCESS DESIGN

3.2.1 SEQUENCE DIAGRAM

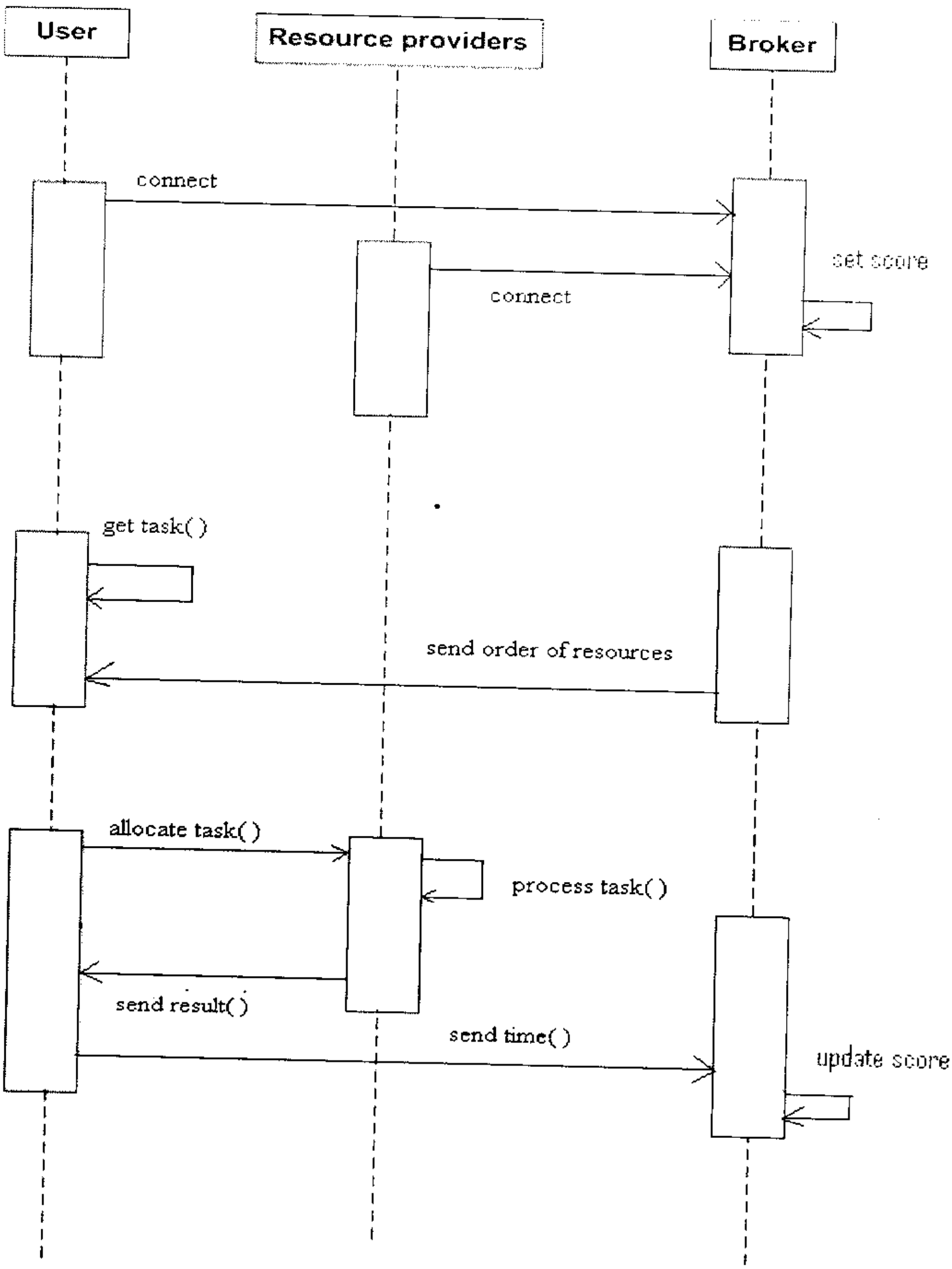


Fig3.4 Both the Grid creation and Resource Management

3.4 DEPLOYMENT DESIGN

3.4.1 DEPLOYMENT DIAGRAM

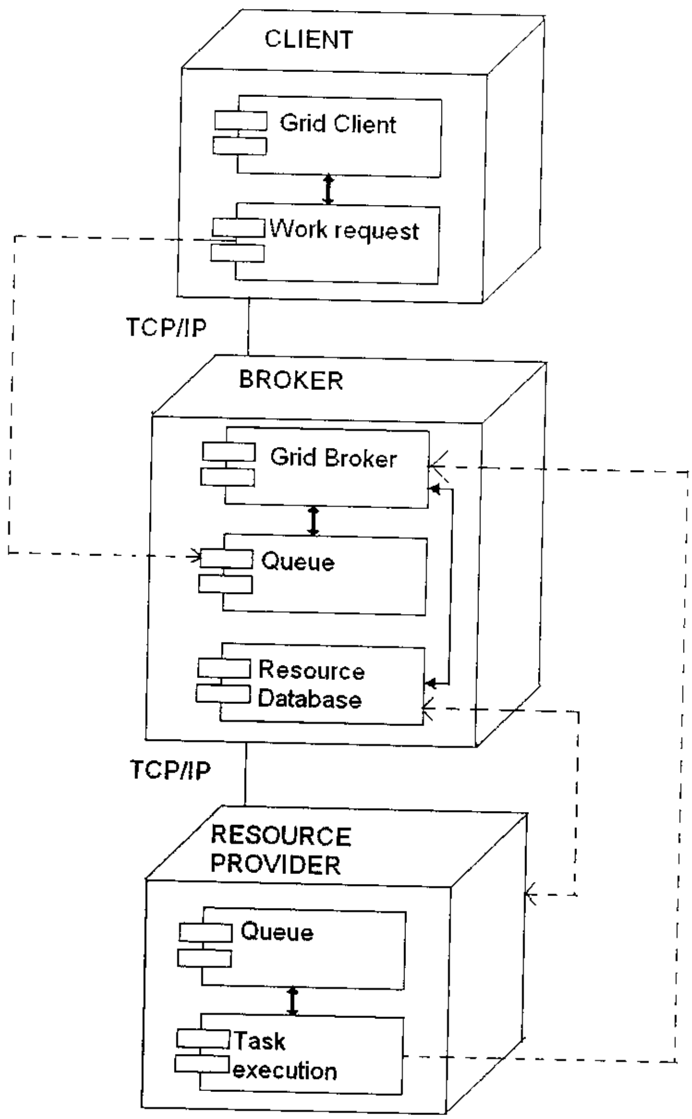


Fig 3.5 Deployment Diagram

3.5 DATA DESIGN

3.5.1 TABLE DESIGN

FIELD NAME	DATA TYPE	DESCRIPTION
Clientip	text	The clientip is used to store the ip address of the client connected to the grid
score	Number	The score is used for reinforcement learning and is updated whenever the resource providers complete their task
TotTime	Number	This is used to store the time in milli seconds which later can be used for metered pricing
Starttime	Date/Time	This is used to store the date and time when the client actually comes into the grid

Fig 3.6 Tabular design

4. FUTURE ENHANCEMENTS

FUTURE ENHANCEMENTS

A Grid resource allocation mechanism must be adaptive to the changing policy environment, as policy satisfies the way in which a task is completed. This means that individual agents will not share the same view of the environment. Different agents should have different sets of resources under consideration and have different experiences of resource performance due to differential policies (priorities).

In future this mechanism can be implemented as a multi agent system by providing the agent for every resource connected in the grid. Another significant challenge for users and brokers is co-allocation. The resources may be heterogeneous and the co-allocating agent must coordinate interactions with multiple providers where no pre-existing coordination can be assumed.

By exploiting quasi-transactional mechanisms for resource allocation, i.e., advance reservation and agreement , we believe that the learning agent may be able to decompose the co-allocation problem into a set of simpler independent operation types with separate learning states

5. RESULT ANALYSIS

RESULT ANALYSIS

NUMBER OF MACHINES VS TIME IN ms

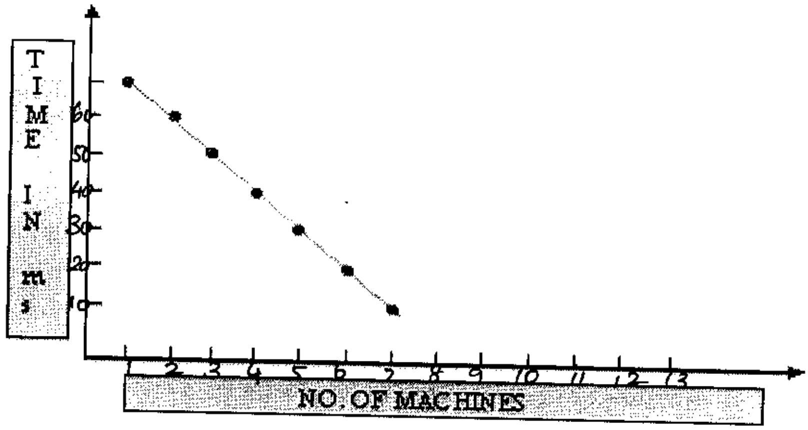


Fig 5.1

From the result analysis we have concluded that

1. If the number of machines increase then the time taken to complete the Job is decreased i.e. efficiency will be high.
2. More complex jobs can be easily done by dividing the complex jobs into various sub-tasks.

   If there is a single machine which implies work load is high and time taken to complete the job is also very high.

   If there are more than two machines which implies work load is low and time taken to complete the job is also less.
3. The Fig 5.1 shows that the graph represents a slope line which indicates the importance of Grid computing in order to perform complex applications with higher efficiency in the result.

6. CONCLUSION

CONCLUSION

Our approached of the resource allocation system agents use reinforcement learning based on local observations to adapt to changing resource loads. However, they assumed that the capacity of the resource is evenly distributed over the jobs, hence no queueing occurred. Although this difference seems to be a minor, it actually has a significant impact on the reinforcement signal received by the agents. Indeed, even within the FCFS scheduling scheme considered in this project, the wait time for a job submitted by an agent might vary by orders of magnitude depending on its position in the queue due to the wide dispersion in job sizes.

The benefit we have observed for the RL algorithm over random selection already an improvement over existing Grid meta-scheduling strategies, many of which, while performing substantial planning of job sequences etc., make random or otherwise uniform distribution decisions to spread work among several (or many) large-scale resources .Even when meta-schedulers attempt to use environmental information, such as load levels, our results suggest that the RL algorithm can provide better adaptive behavior because each meta-scheduler would learn from the environment's responses to its own queries.

7. APPENDIX

APPENDIX

8.1 SAMPLE CODES

SERVER SIDE PROGRAM:

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
import java.sql.*;
import java.util.Date;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

class Server extends JFrame implements ActionListener {

	class serverThread extends Thread {
		Socket sc=null;
		ServerSocket ss1=null;
		BufferedReader br=null;
		PrintStream ps=null;
		String strip =null;

		serverThread(int port) {
			try {
				ss1=new ServerSocket(port);
				start();
			}
			catch(Exception ex) {
				System.out.println("In serverThread "+ex);
			}
		}

		public void run() {
			while(true) {
				try {
					System.out.println("Waiting to Del");
					sc=ss1.accept();
```

```
                              br=new BufferedReader(new
InputStreamReader(sc.getInputStream()));
                              ps=new
PrintStream(sc.getOutputStream(),true);
                              System.out.println("Going to Delete the
record");
                              if(br.readLine().equals("0")) {
                                   String
temp=sc.getRemoteSocketAddress().toString();
                                   String
strip=temp.substring(1,temp.indexOf(":"));
                                   delDatabase(strip);
                              }
                              ps.close();
                              br.close();
                              sc.close();
                         }
                         catch(Exception ex) {
                              System.out.println("ClientListen "+ex);
                         }
                    }
               }
               void delDatabase(String ip) {
                    Connection cn=null;
                    Statement stmt=null;
                    try {
                         System.out.println("Dele record");

     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                         cn =
DriverManager.getConnection("jdbc:odbc:agent","","");
                         stmt
=cn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_UPDATABLE);
                         ResultSet rs=stmt.executeQuery("select clientip
from client where clientip='"+ip+"'");
                         if(rs.next()) {
                              rs.absolute(1);
                              rs.deleteRow();
                              System.out.println("Deleted");
                         }
                         rs.close();
```

```
                    cn.close();
                }
                catch(Exception ex) {
                    System.out.println("deldb"+ex);
                }
            }
        }

        class sendIP extends Thread {
            Socket soc=null;
            BufferedReader br=null;
            PrintStream ps=null;
            ServerSocket ss=null;
            sendIP(int port) {
                try {
                    ss=new ServerSocket(port);
                    start();
                }
                catch(Exception ex) {
                    System.out.println("ClientListen "+ex);
                }
            }
            public void run() {
                while(true) {
                    try {
                        soc=ss.accept();
                        ps=new
PrintStream(soc.getOutputStream(),true);
                        System.out.println("connectd1");
                        ArrayList ip=getIP();
                        for(int i=0;i<ip.size();i++) {
                            System.out.println(ip.get(i));
                            ps.println(ip.get(i));
                        }
                        ps.println("END");
                    }
                    catch(Exception ex) {
                        System.out.println("ClientListen "+ex);
                    }
                }
            }
```

```
                    ArrayList obj=new ArrayList();
                    try {

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                            Connection cn =
DriverManager.getConnection("jdbc:odbc:agent","","");
                            Statement stmt =cn.createStatement();
                            ResultSet rs=stmt.executeQuery("select clientip
from client order by score desc");
                            while(rs.next()) {
                                    System.out.println("---
"+obj.add(rs.getString(1)));
                            }
                            stmt.close();
                            cn.close();
                    }
                    catch(Exception ex) {
                            System.out.println("get IP "+ex);
                    }
                    return obj;

            }


    }


    class waitThread extends Thread {
            Socket soc=null;
            BufferedReader br=null;
            ServerSocket ss=null;
            waitThread(int port) {
                    try {
                            ss=new ServerSocket(port);
                            start();
                    }
                    catch(Exception ex) {
                            System.out.println("ClientListen "+ex);
                    }
            }
            public void run() {
```

```
            System.out.println("Waiting in waitThread ");
                                soc=ss.accept();
                                System.out.println("connectd1");
                                br=new BufferedReader(new
InputStreamReader(soc.getInputStream()));
                                String strIP=br.readLine();
                                int mem=Integer.parseInt(br.readLine());
                                //String mem=br.readLine();
                                System.out.println(strIP+" == "+mem);
                                insertDB(strIP,mem);
                        }
                        catch(Exception ex) {
                                System.out.println("ClientListen "+ex);
                        }
                }
        }
        void insertDB(String ip,int mem) {
                Connection cn=null;
                Statement stmt=null,stmt1=null,stmt2=null,st=null;
                try {

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        cn =
DriverManager.getConnection("jdbc:odbc:agent","","");
                        st =cn.createStatement();
                        stmt =cn.createStatement();
                        stmt1=cn.createStatement();
                        stmt2=cn.createStatement();
                        int x;
                        x=st.executeUpdate("update client set
TotTime="+mem+" where clientip='"+ip+"'");
                        System.out.println("Updated Row : "+x);
                        ResultSet rs1=stmt.executeQuery("select
count(*) from client");
                        int trow=0;
                        if(rs1.next())
                                trow=rs1.getInt(1);

                        System.out.println("Total Rows :"+trow);
                        rs1.close();
                        //ResultSet rs=stmt1.executeQuery("select
clientip,score from client order by TotTime");
```

```
                    ResultSet rs=stmt1.executeQuery("select score
from client where clientip='"+ip+"'");
                    while(rs.next()) {
                        //String t1=rs.getString(1);
                        //x=rs.getInt(2)+trow;
                        x=rs.getInt(1);
                        x++;
                        System.out.println("xx "+x+" : "+ip);
                        x=stmt2.executeUpdate("update client set
score="+x+" where clientip='"+ip+"'");

    activetxtArea.setText(activetxtArea.getText()+"\n"+"IP
Address:"+ip+"  UpdatedScore:"+x+"  TimeTaken:"+mem);
                        System.out.println("Updated Row1 :
"+x);

                        trow--;
                    }
                    rs.close();
                    stmt1.close();
                    stmt.close();
                    stmt2.close();
                    st.close();
                    cn.close();
                }
                catch(Exception ex) {
                    System.out.println("insertDB "+ex);
                }
            }

        }
        class clientListen extends Thread {
            Socket soc=null;
            BufferedReader br=null;
            PrintStream ps=null;
            ServerSocket sss=null;
            clientListen(int port) {
                try {
                    sss=new ServerSocket(port);
                    start();
                }
                catch(Exception ex) {
```

```
		}

		public void run() {
			while(true) {
				try {
					System.out.println("Waiting");
					soc=sss.accept();
					System.out.println("connectd");
					br=new BufferedReader(new
InputStreamReader(soc.getInputStream()));
					ps=new
PrintStream(soc.getOutputStream(),true);
					String
temp=soc.getRemoteSocketAddress().toString();
					String
strip=temp.substring(1,temp.indexOf(":"));
					Date date=new Date();

	activetxtArea.setText(activetxtArea.getText()+"\n"+strip+"
Connected at "+date);
					updateDatabase(strip,date.toString());
					ArrayList ip=getIP();
					for(int i=0;i<ip.size();i++) {
						System.out.println(ip.get(i));
						ps.println(ip.get(i));
					}
					ps.println("END");
				}
				catch(Exception ex) {
					System.out.println("ClientListen "+ex);
				}
			}
		}
		ArrayList getIP() {
			ArrayList obj=new ArrayList();
			try {

	Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
				Connection cn =
DriverManager.getConnection("jdbc:odbc:agent","","");
				Statement stmt =cn.createStatement();
				ResultSet rs=stmt.executeQuery("select clientip
```

```
                        while(rs.next()) {
                                System.out.println("---
"+obj.add(rs.getString(1)));
                        }
                        stmt.close();
                        cn.close();
                }
                catch(Exception ex) {
                        System.out.println("get IP "+ex);
                }
                return obj;

        }
        void updateDatabase(String ip,String date) {
                Connection cn=null;
                Statement stmt=null;
                try {

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        cn =
DriverManager.getConnection("jdbc:odbc:agent","","");
                        stmt =cn.createStatement();
                        ResultSet rs=stmt.executeQuery("select clientip
from client where clientip='"+ip+"'");
                        if(rs.next()){
                        }
                        else {
                                String query="INSERT INTO client
values('"+ip+"',"+"0,0"+",'"+date+"')";
                                System.out.println(query);
                                stmt.execute(query);
                        }
                        stmt.close();
                        cn.close();
                }
                catch(Exception ex) {
                        System.out.println("ClientListen "+ex);
                }
        }
    }
  Server() {
    super("Realtime System Scheduling Server")
```

```
    jal = new JPanel();
    jrl = new JPanel();
    jbt = new JPanel();
            cmdQuit= new JButton( "Quit" );
    cmdStart = new JButton(" Start ");
    cmdStop = new JButton(" Stop ");
    cmdClear = new JButton("Clear Log");
    cmdView=new JButton("View");
    activetxtArea = new JTextArea(30, 40);

    activetxtArea.setFont(new Font("Serif", Font.BOLD, 16));
    scrollPane1 = new JScrollPane(activetxtArea);
    init();
}

public void init()
{
    contentPane = getContentPane();
    jpg = new JPanel();
    jbt.add(cmdStart);
    jbt.add(cmdStop);
    cmdStart.setEnabled(true);
    cmdStop.setEnabled(false);
    cmdClear.setEnabled(false);
    jbt.add(cmdClear);
    jbt.add(cmdView);
    jbt.add(cmdQuit);
    jal.add(scrollPane1);
    jpg.add(jbt);
    jpg.add(jal);
    jpg.add(jrl);
    contentPane.add(jpg);
    activetxtArea.setEditable(false);
    setLocation(0, 0);
    setSize(500, 570);
    setVisible(true);
    setResizable(false);
    cmdQuit.addActionListener(this);
    cmdStart.addActionListener(this);
    cmdStop.addActionListener(this);
    cmdClear.addActionListener(this);
```

```
public void actionPerformed(ActionEvent actionevent)
{
   if(actionevent.getSource().equals(cmdQuit))
   {
      System.out.println("Logout");
      System.exit(0);
   }
   if(actionevent.getSource().equals(cmdStart))
      try
      {
         cmdStart.setEnabled(false);
         cmdStop.setEnabled(true);
         cmdClear.setEnabled(true);
         System.out.println("Server Starts...");
                         new clientListen(6000);
                         new waitThread(6001);
                         new sendIP(6002);
                         new serverThread(6500);
   activetxtArea.setText(activetxtArea.getText()+"Grid Connectd ...");
        System.out.println("Server Listen");
     }
     catch(Exception exception)
     {
        System.out.println("XXXXX"+exception);
     }
  if(actionevent.getSource().equals(cmdStop))
     try
     {
        System.out.println("Stop");
        cmdStart.setEnabled(true);
        cmdStop.setEnabled(false);
     activetxtArea.setText(activetxtArea.getText()+"\n"+"Grid Stop");
     }
     catch(Exception exception1)
     {
        System.out.println("X2"+exception1);
     }
if(actionevent.getSource().equals(cmdClear))
{
   System.out.println("Clear Log");
   activetxtArea.setText("");
```

```
      if(actionevent.getSource().equals(cmdView)) {
                    try {
                         int cnt=0;

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                         Connection cn =
DriverManager.getConnection("jdbc:odbc:agent","","");
                         Statement stmt =cn.createStatement();
                         ResultSet rs=stmt.executeQuery("SELECT
COUNT(*) FROM client");
                         while(rs.next()){
                              cnt=rs.getInt(1);
                         }
                         System.out.println("Count :"+cnt);
                         viewTable objrt=new viewTable(cnt);
                    }catch(Exception ex) {
                         System.out.println("In View Table"+ex);
                    }
               }
  }
  public static void main(String args[])
  {
    new Server();
  }
  JPanel jpg;
  JPanel jal;
  JPanel jrl;
  JPanel jbt;
  JButton cmdConfig;
  JButton cmdQuit;
  JLabel lblTitle;
  JButton cmdStart;
  JButton cmdStop;
  JButton cmdClear;
  JButton cmdView;
  JTextArea activetxtArea;
  JScrollPane scrollPane1;
  JTextArea respondtxtArea;
  JScrollPane scrollPane2;
  Container contentPane;
}
```

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

class Client extends JFrame implements ActionListener {

        Socket soc=null;
        BufferedReader br=null;
        PrintStream ps=null;
        String strip=null;
        timeFind objTime=null;
        ArrayList alSum = new ArrayList();

  Client()
  {
    super("Realtime System Scheduling ");
    init();
  }

  public void init()
  {
    txtip=new JTextField(15);
            cmdQuit= new JButton( "Quit" );
    cmdStart = new JButton(" Connect ");
    cmdStop = new JButton(" Disconnect ");
    cmdClear = new JButton("Clear Log");

            txtArea = new JTextArea(20, 50);
            JScrollPane scrollPane =
                   new JScrollPane(txtArea,

     JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,

     JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
          txtArea.setFont(new Font("Serif", Font.BOLD, 16));
          txtArea.setLineWrap(t
```

```
                txtArea.setEditable(false);
                cmdStop.setEnabled(false);

                getContentPane().setLayout(new
 FlowLayout(FlowLayout.CENTER));
        getContentPane().add(txtip);
        getContentPane().add(cmdStart);
        getContentPane().add(cmdStop);
        getContentPane().add(cmdClear);
        getContentPane().add(cmdQuit);
        getContentPane().add(scrollPane);

                setLocation(200, 200);
                setSize(625, 550);
                setVisible(true);

        cmdQuit.addActionListener(this);
        cmdStart.addActionListener(this);
        cmdStop.addActionListener(this);
        cmdClear.addActionListener(this);

    }

    public void actionPerformed(ActionEvent actionevent)
    {
        if(actionevent.getSource().equals(cmdQuit))
        {
            System.out.println("Logout");
            System.exit(0);
        }
        if(actionevent.getSource().equals(cmdStart)) {
            try
            {
                System.out.println("Server Starts...");
                if(txtip.getText().trim().equals(""))
                    JOptionPane.showMessageDialog(this, "   Enter Valid Grid
IP", "M E S S A G E", 1);
                else {
                                try {
                                        strip=txtip.getText().trim();
                                        soc=new Socket(strip,6000);
```

```
                                        cmdClear.setEnabled(true);
                                        txtip.setEnabled(false);
                        txtArea.setText("Connected to "+txtip.getText());
                              System.out.println("Client Listen");
                                        new clientListener();
                                        new clientProcess();
                                  }
                                  catch(Exception ex) {
                                  //System.out.println("ClientListen "+ex);

        txtArea.setText(txtArea.getText()+"\n"+ex);
                                  }
                            }
        }
        catch(Exception exception)
        {
          System.out.println("XXXXX"+exception);
        }
             }
      if(actionevent.getSource().equals(cmdStop)) {
        try
        {
          System.out.println("Stop");
          Socket socend=new Socket(strip,6500);
          PrintStream psend=new
PrintStream(socend.getOutputStream(),true);
          psend.println("0");
          psend.close();
          socend.close();
          cmdStart.setEnabled(true);
          cmdStop.setEnabled(false);
          txtip.setEnabled(true);
          txtArea.setText(txtArea.getText()+"\n"+strip+"
Disconnected");
        }
        catch(Exception exception1)
        {
          System.out.println("X2"+exception1);
        }
             }
      if(actionevent.getSource().equals(cmdClear))
      {
```

```
            txtArea.setText("");
        }
    }

    class clientListener extends Thread {
            ServerSocket ss=null;
            Socket sc=null;

            clientListener() {
                    try {
                            ss=new ServerSocket(7000);
                            start();
                    }
                    catch(Exception ex) {
                                    System.out.println("Client Process"+ex);
                    }
            }

            public void run() {
                    objTime=new timeFind();
                    while(true) {
                            try{
//txtArea.setText(txtArea.getText()+"\nNow this System is waiting for
job");

                                    sc=ss.accept();
                                    System.out.println("Connected");
                    txtArea.setText(txtArea.getText()+"\nJob Accepted");
                                    BufferedReader br=new
BufferedReader(new InputStreamReader(sc.getInputStream()));
                                    PrintStream printJob=new
PrintStream(sc.getOutputStream(),true);
                            FileReader fr=new FileReader("ip.txt");
                    //BufferedReader bread=new BufferedReader(fr);
                                    String temp;
                                    ArrayList objIP=new ArrayList();
                                    Socket skforIP=new Socket(strip,6002);
                                    BufferedReader brIP=new
BufferedReader(new InputStreamReader(skforIP.getInputStream()));


    txtArea.setText(txtArea.getText()+"\nAvailble Clients");
```

```
txtArea.setText(txtArea.getText()+"\n"+temp);
                                objIP.add(temp);
                        }
                        txtArea.setText(txtArea.getText()+"\n");

            brIP.close();
            skforIP.close();

            txtArea.setText(txtArea.getText()+"\n Input : ");
                        ArrayList objInput=new ArrayList();

                        int inp=0;

      while(!(temp=br.readLine()).equals("END")) { // Read from Socket
(Input)
                  txtArea.setText(txtArea.getText()+"\n"+temp);
                              //objInput.add(temp);
                              inp = Integer.parseInt(temp);
                        }
                        String t = "";
                        int x1;
                        int x2;
                        x1 = 1;
                        do {
                              x2 = x1+99;
                              if(inp>x2)
                                    t = x1 + "-" + x2;
                              else
                                    t = x1 + "-" + inp;
                              objInput.add(t);
                              System.out.println(t);

                              if(inp<=x2)
                                    break;
                              x2++;
                              x1 = x2;

                        }
                        while(true);
```

```
        txtArea.setText("\n"+txtArea.getText()+"\nJob Allocated in
following Order ");
                                int diff=objInput.size()-objIP.size();
                                int len=0;
                                boolean flag=false;

                                if(diff>=0) {
                                        len=objIP.size();
                                        flag=true;
                                }
                                else if(diff<0) {
                                        len=objInput.size();
                                        flag=false;
                                }
                System.out.println(objInput.size()+"  :  "+objIP.size());
                                int i=0,j=0,x=objInput.size();
                                while(true) {
                        //long l1=System.currentTimeMillis();
                while(j<len) {
                                System.out.println("inside");
                                System.out.println("i="+i+",j="+j);
                                String strip1=(String)objIP.get(j);
                                String strval=(String)objInput.get(i);
                System.out.println("in ClientListener : "+strip1+" : "+strval
                txtArea.setText(txtArea.getText()+"\n"+"REQUEST   :
"+objTime.getTime()+ " : "+ strip1+" : "+strval);
                clientRequest clientObj=new clientRequest(strip1,strval);
                                                i++;
                                                j++;
                                        }

                        //long l2=System.currentTimeMillis();

        //txtArea.setText(txtArea.getText()+"\n"+"Time taken for this Job :
"+(l2-l1)/1000);


                                if(flag==false)
                                        break;
                                        else {
                                                diff=x-objIP.size();
```

```
                                    System.out.println("in loop "+diff);
                                        j=0;
                                        //Thread.sleep(5000);
                                        if (diff==0)
                                            break;
                                        else {
                                        //Thread.sleep(10000);
                                            objIP.clear();
                              skforIP=new Socket(strip,6002);
 brIP=new BufferedReader(new
 InputStreamReader(skforIP.getInputStream()));
      txtArea.setText(txtArea.getText()+"\nAvailble Clients");

 while(!(temp=brIP.readLine()).equals("END")) {

 txtArea.setText(txtArea.getText()+"\n"+temp);

 objIP.add(temp);
                                            }
                                            continue;
                                        }
                                    }
                              }
                              Thread.sleep(1000);
                         //System.out.println("al size " + alSum.size());
                              BigInteger b = new BigInteger("1");
                              for(int i1 = 0; i1 < alSum.size(); i1++) {
                              //System.out.println(alSum.get(i1));
      b = b.multiply(new BigInteger(alSum.get(i1).toString()));
                                   //System.out.println("b = " + b);
                              }

txtArea.setText(txtArea.getText()+"\nResult = " + b);

                         }
                         catch(Exception ex){
                         System.out.println("In clientListener : "+ex);
                         }
                    }
          }
}
```

```
class timeFind {
        Date d;
        Date getTime(){
                d=new Date();
                return d;
        }
}

class clientRequest extends Thread {
        Socket clientSoc=null;
        BufferedReader clientBr=null;
        PrintStream clientPs=null;
        String ip,val;
        clientRequest(String ip,String val) {
                try {
                        clientSoc=new Socket(ip,7001);
                        this.val=val;
                        this.ip=ip;
                        System.out.println("Thread going to Start");
                        start();
                        //join();
                }
                catch(Exception ex) {
                                System.out.println("Client Process"+ex);
                }
        }
        public void run() {
                objTime=new timeFind();
                try {
                        System.out.println("Coming for Processing");
                        clientBr=new BufferedReader(new
InputStreamReader(clientSoc.getInputStream()));
    clientPs=new PrintStream(clientSoc.getOutputStream(),true);

                        //long l1=System.currentTimeMillis();
                        clientPs.println(val);
                        String temp,strmem;
                        temp=clientBr.readLine();
                        String strtime=clientBr.readLine();
                        //long l2=System.currentTimeMillis();
```

```
        System.out.println("In ClientRequset value is :"+val+" = "+temp);

        txtArea.setText(txtArea.getText()+"\n"+"RESPONSE :
"+objTime.getTime()+ " : "+ip+" : "+val+" : "+temp+" : "+strtime);
                                //sum.add(new BigInteger(temp));
                                alSum.add(temp);
                                System.out.println("added");

                                Socket serSoc=new Socket(strip,6001);
                                PrintStream serPs=new
PrintStream(serSoc.getOutputStream(),true);
                                serPs.println(ip);
                                //serPs.println(strmem);
                                serPs.println(strtime);
                                serPs.close();
                                serSoc.close();
                                stop();
                        }
                        catch(Exception ex) {
                                System.out.println("ClientRequest "+ex);
                        }
                }
        }

    public static void main(String args[])
    {
      new Client();
        }

    JButton cmdConfig;
    JButton cmdQuit;
    JLabel lblTitle;
    JButton cmdStart;
    JButton cmdStop;
    JButton cmdClear;
    JTextField txtip;
    JTextArea txtArea;
    JScrollPane scrollPane;
    Container contentPane;
```

8.2 SCREENSHOTS

SERVER:

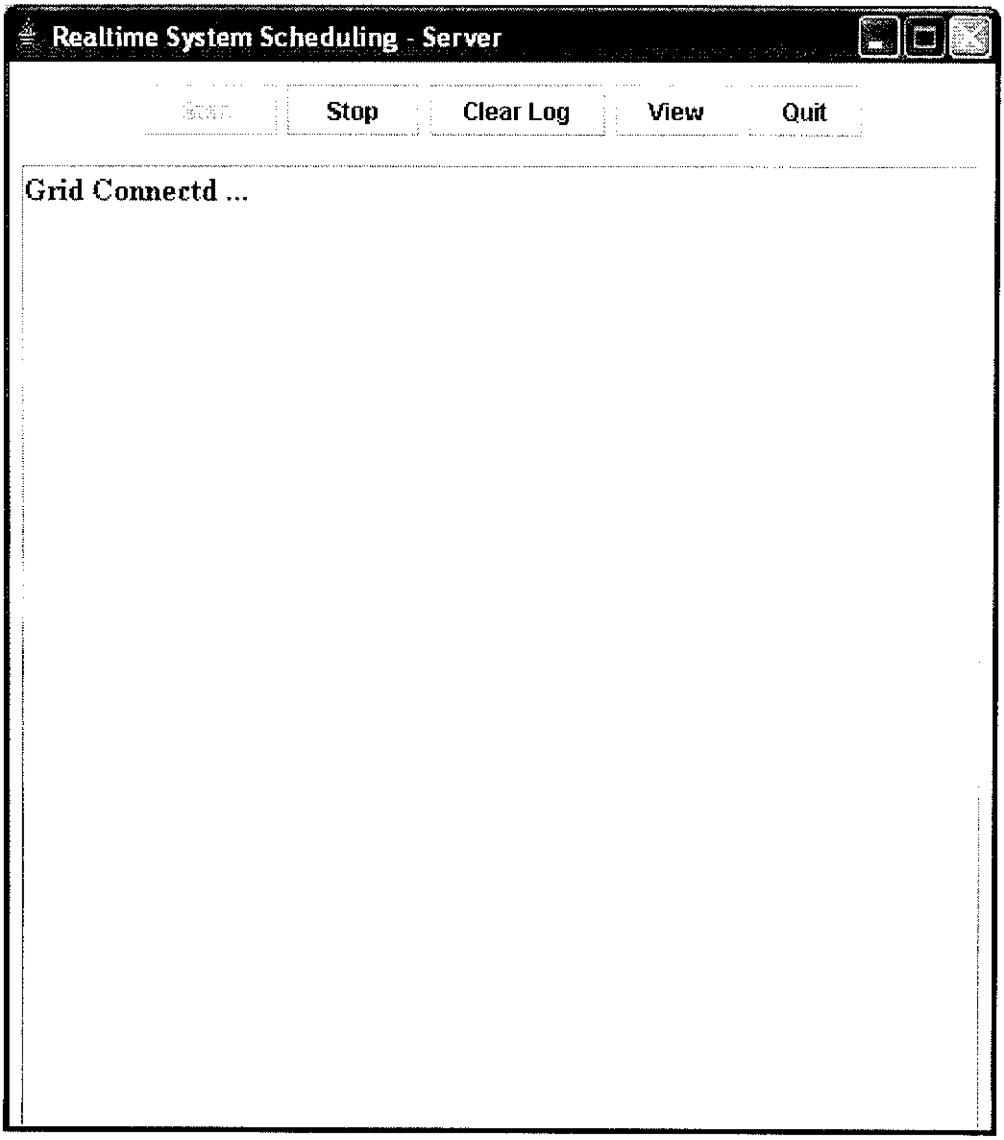
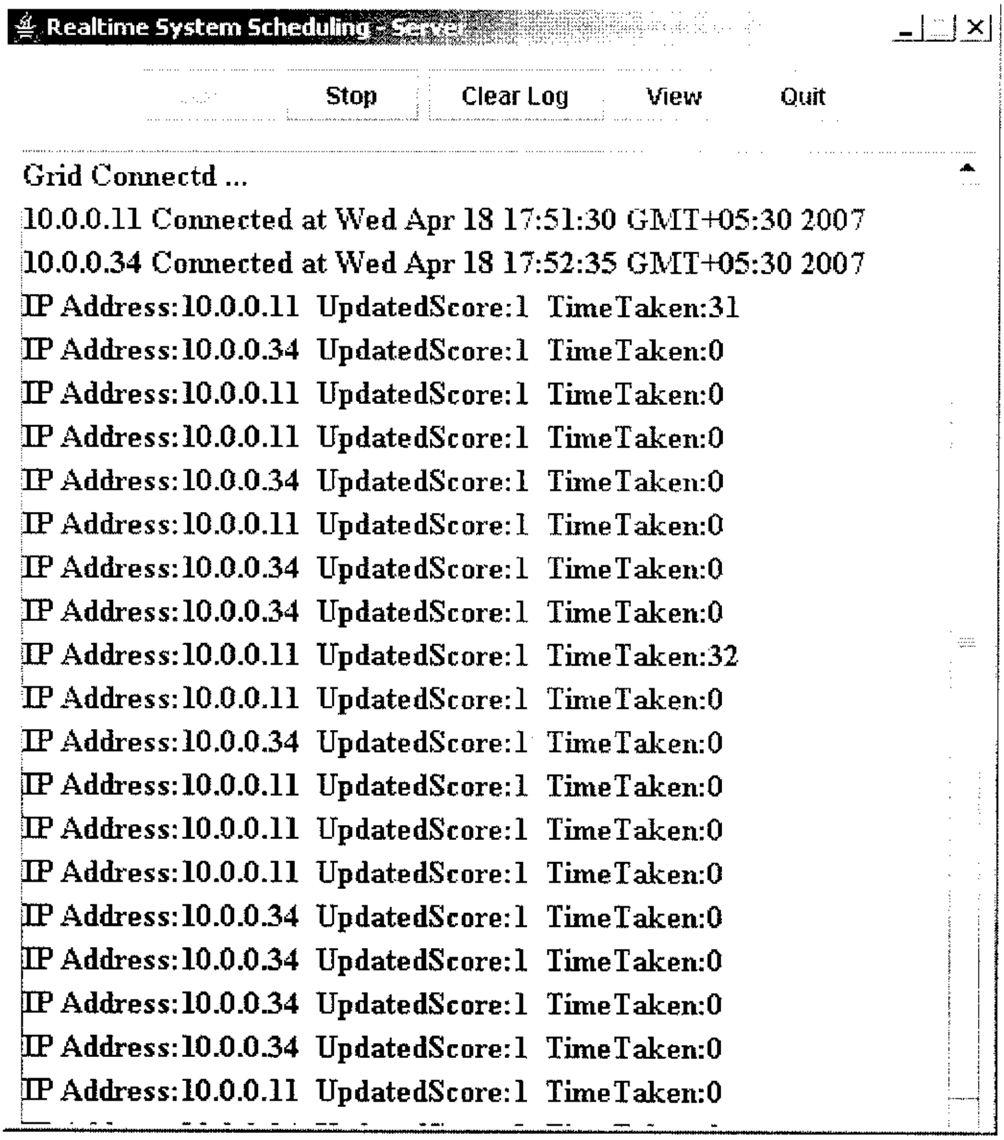


Fig 3.9 Server Application window



3.11 Server window showing the details of the tasks done by various clients connected to the Grid environment

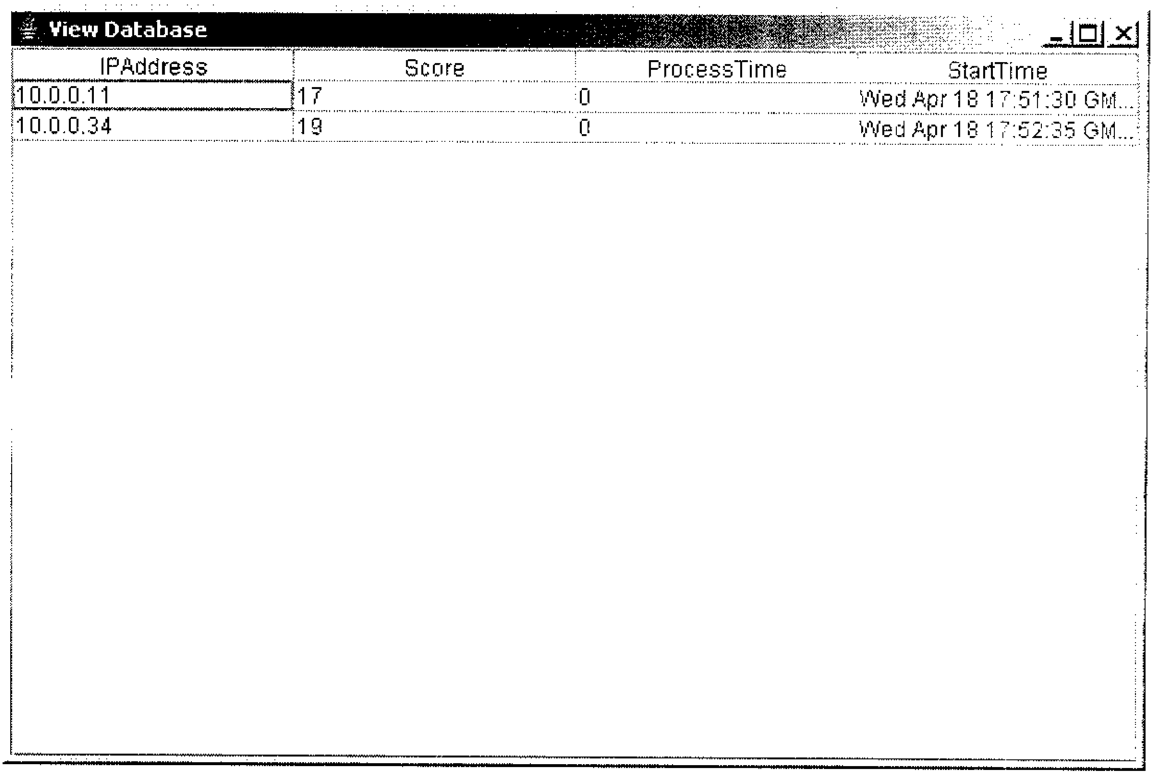


Fig 3.14 Server window showing the database

CLIENTSIDE:

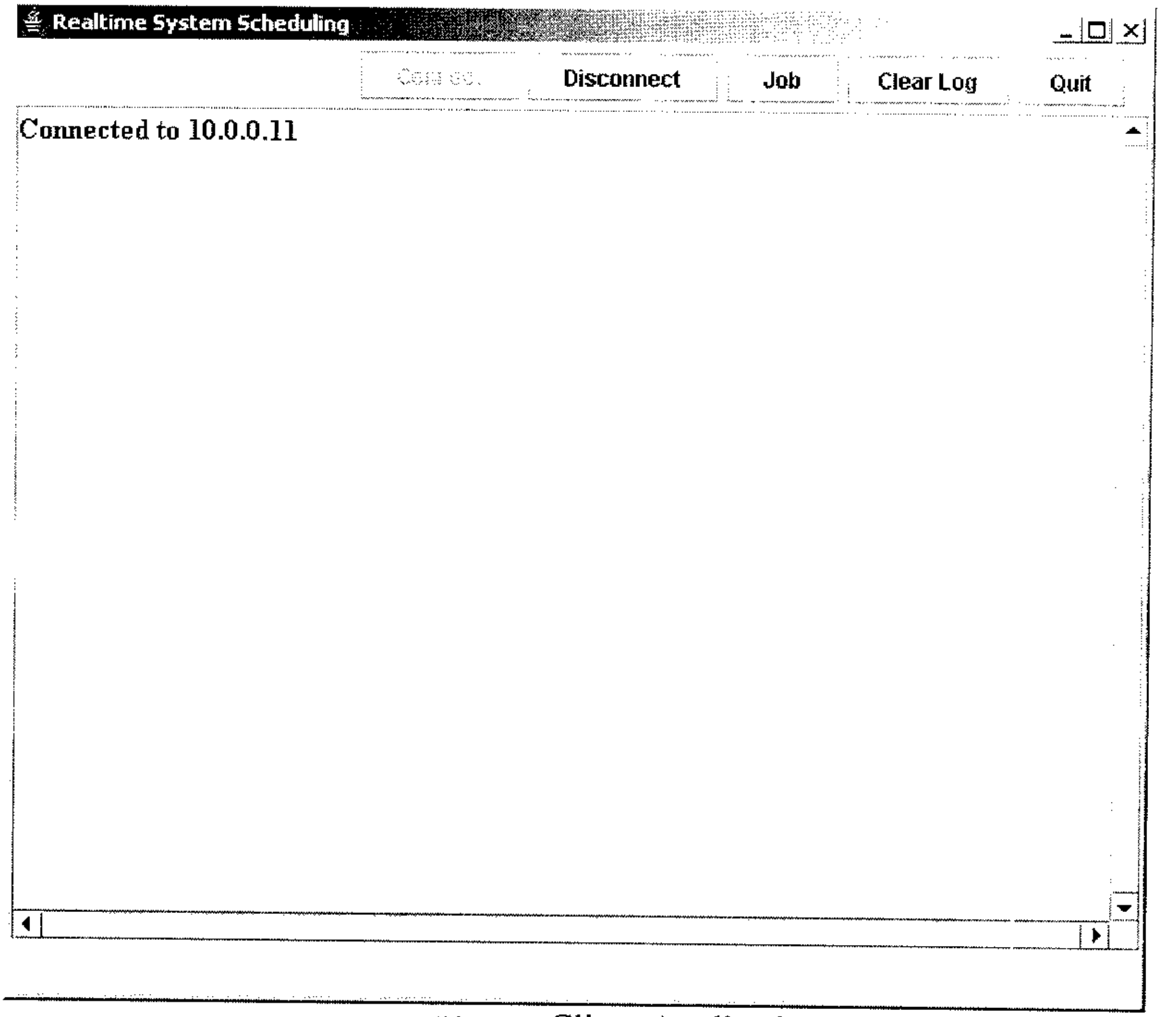
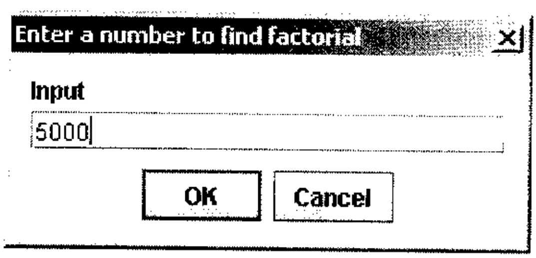


Fig 3.10 Client Application window



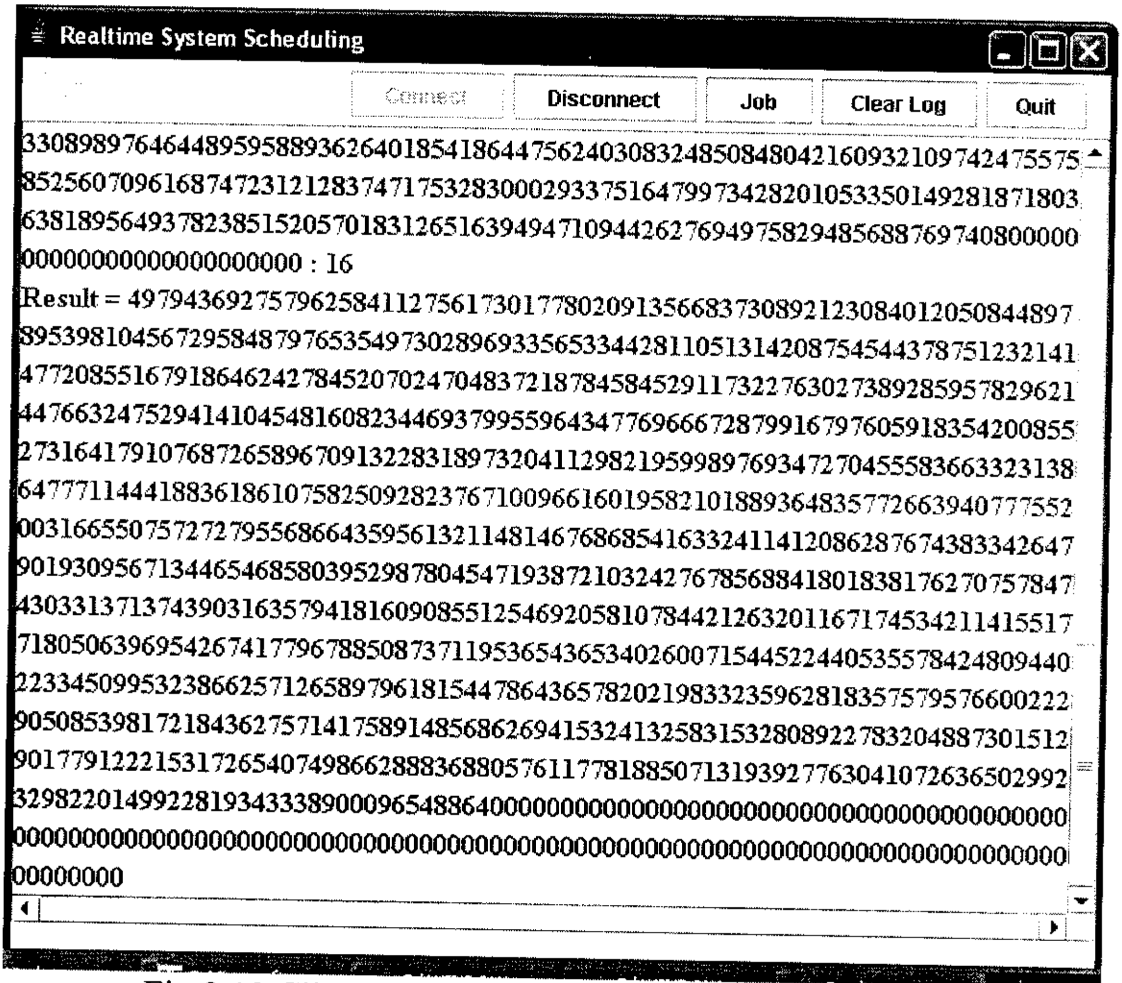


Fig 3.12 Client window showing the work done details

8. REFERENCES

REFERENCES

1. Herbert Schildt (2002), *The Complete Reference Java 2.* Fourth Edition Tata McGraw-Hill. 2002

2. *Grid Computing A research Monograph* ,D.janakiram 2005, Tata Mc Graw-Hill

3.computer networks, Andrew Tanenbaum Fourth Edtion 2005, Prentice-Hall of India Pvt

WEBSITES

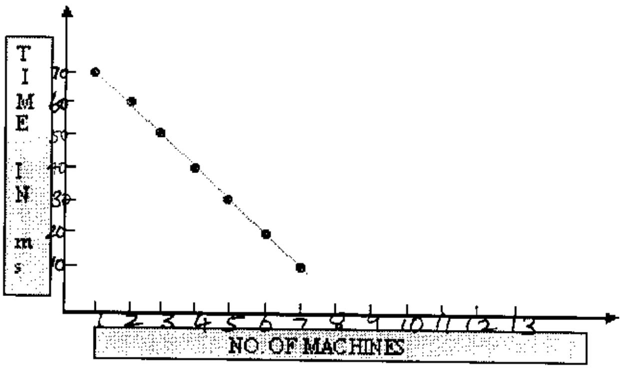
1.www.grid computing.org

2.www.globus.org

3.www.gryphyn.org

4. www.network.com

RESULT ANALYSIS



From the result analysis we have concluded that

1. If the number of machine increases then the time taken to complete the job is decreased i.e. efficiency will be high.
2. More complex jobs can be easily done by dividing the complex jobs into various sub-tasks.
   If there is a single machine -> work load is high and time taken to complete the job is very high.
   If there is more than two machines -> work load is low and time taken to complete the job is less.
3. The graph represent the slope line which indicates the importance of Grid computing in order to perform the complex application with higher efficiency in the result.