P-1495

LOAD BALANCING FOR DISTRIBUTED COMPUTING

A PROJECT REPORT

Submitted by

REMYA SHANKAR (71201104042)
RAJASOWMIYA.M (71201104037)
SIVAKUMAR.P (71201104059)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600025

APRIL 2005

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report **"LOAD BALANCING FOR DISTRIBUTED COMPUTING"** is the bonafide work of **"REMYA SHANKAR, RAJASOWMIYA.M and SIVAKUMAR.P"** who carried out the project work under my supervision.

[illegible]

SIGNATURE

[illegible] 15·4·05

SIGNATURE

Dr. S. Thangasamy
HEAD OF THE DEPARTMENT
Computer Science and Engineering
Kumaraguru College of technology,
Chinnavedapatti,
Coimbatore- 641006

Ms.S.Rajini
SENIOR LECTURER
Computer Science and Engineering
Kumaraguru College of technology
Chinnavedapatti,
Coimbatore- 641006

Submitted for the Viva-voce Examination held on 20-04-05

[illegible] 20/4

Internal Examiner

[illegible] 20/4

External Examiner

EVALUATION CERTIFICATE

ANNA UNIVERSITY : CHENNAI 600 025

EVALUATION CERTIFICATE

College : **KUMARAGURU COLLEGE OF TECHNOLOGY**

Branch : **COMPUTER SCIENCE AND ENGINEERING**

Semester : **EIGHT (08)**

S.No	Name of the Students	Title of the Project	Name of Supervisor
1	SIVAKUMAR.P RAJASOWMIYA.M REMYASHANKAR	Load balancing for distributed computing	Ms.S.Rajini Senior Lecturer

The report of the project work submitted by the above student in partial fulfillment for the award of BACHELOR OF ENGINEERING degree in COMPUTER SCIENCE AND ENGINEERING of Anna University were evaluated and confirmed to be the report of the work done by the above student and then evaluated.

(INTERNAL EXAMINER) (EXTERNAL EXAMINER)

DECLARATION

DECLARATION

We hereby declare that the project entitled " LOAD BALANCING FOR DISTRIBUTED COMPUTING", is a record of original work done by us and 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 the Degree of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place: Coimbatore

Date: 12-04-05

[illegible]
(RAJASOMIYA.M)

[illegible]
(REMYASHANKAR)

[illegible]
(SIVAKUMAR.P)

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We thank Almighty God, the guiding light of my life, for granting me the strength and courage to complete this project work successfully.

We express our immense gratitude to **Dr. K.Arumugam**, Correspondent, Kumaraguru College of Technology, Coimbatore for giving me an opportunity to study in their prestigious institution and to take up the project in partial fulfillment of the regulations for the B.E. program.

We are thankful to **Dr. K.K.Padmanabhan**, Principal, Kumaraguru College of Technology, Coimbatore, for the facilities made available during the course.

We convey our heartfelt thanks to **Dr.S.Thangasamy, B.E. (Hons), Ph.D.,** Professor and Head, Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore, for his guidance and enthusiasm that went a long way in the successful completion of the project. I also thank **Ms.S.Rajini**, B.E, project guide, Kumaraguru College of Technology, Coimbatore, for the invaluable advice and guidance given to me to make this endeavor a successful one.

Last but not the least, we would like to thank all the staff members of the Computer Science department and all those who have directly or indirectly assisted me in successfully completing this project.

ABSTRACT

ABSTRACT

A distributed system is a collection of autonomous processors, which can communicate with each other, which allows resource sharing. The purpose of such a system is to provide a mechanism to solve a huge task in a limited amount of time with efficiency. The autonomous processors can communicate with each other and co-operate in a consistent and a stable manner.

In a distributed system, the project or the problems are divided into various modules and provided to the processors for processing. In course of time, some of the processors are heavily loaded and others are lightly loaded and some are even idle. The load balancing mechanism tries to balance the total system by transparently transferring the workload from heavily loaded nodes to lightly loaded nodes in the attempt to ensure good overall performance.

Load balancing mechanism involves two phases

- Process scheduling
- Process migration

Process scheduling involves the process of finding which process to migrate and to which processor it should migrate. Process migration is the actual transfer of the process from the source processor to the destination processor.

This load balancing algorithm strives to balance the load on all nodes at all times. A processor may be in any of the three states: heavy load, normal load or light load. A processor in light load may accept a migrant process, while a processor in heavy load can have its processes migrate to another processor. A processor in normal load needs no balancing effort.

CONTENT

CONTENTS

INTRODUCTION

1. INTRODUCTION

1.1 Existing system and its limitations

Existing System

At first round robin was considered the standard in sharing the load among the servers. Super computers are used by some organizations for finishing their job fastly. Later some of the software load balancing packages came in to use, they are

- BEOWULF
- COW (Clustering of Workstations)
- HEARTBEAT
- OSCAR
- VLS
- MOSIX

Limitations of the Existing System

- When building in sequence, it is possible that a node receives a task that will take it much longer than the other nodes to complete. It is also possible that as other nodes finish their jobs faster, the node, which has been slowed down is handed another job. When performing large parallel builds, eventually very slow machines will stall the entire build, as they are attempting to compile many objects at once, and are usually at this point near-death from swapping.

- In Round Robin a long process had to wait for long time.
- A new job had to wait for a long time till all the assigned jobs got completed.
- There are large numbers of commercial balancing and queuing systems available such as CODINE from Genias software, Load Lever from IBM. All these allow the presence of traditional batch system that might be found on an IBM mainframe.
- The software load-balancing package is used only for specific applications.
- Very traditional batch systems require some active tools. These are very good tools, but they add greatly to the cost of the cluster as their cost is usually tied directly to the number of nodes to run the batch systems.
- Many who have a need to harness super computing power don't buy super computers because they can't afford them. And then there is the upgrade problem.
- By the time your machine is delivered, it is often out of date and a newer model will have taken its place. If you would like to upgrade your machine the manufacturer gives you limited options.

1.2 Proposed System

Objectives of the proposed system

Load balancing can be used for all parallel programming applications, which should have no interaction in between. Fault tolerance and high availability is the distinct advantage of load balancing. One of the important features of cluster will be the ability to set up systems by which the users can run jobs non-interactively. It is a group of systems, which functions as super computer.

Advantages of the Proposed System

1. It is simple, easy and fast to build.

2. It is very simple to build using components available from a lot of sources unlike commercial products; they are independent of a single vendor or a single source for equipment.

3. Clusters are a cheap and easy way to take off-the-shelf components and combine them in to a single super computer. The clusters are actually faster than a commercial super computer.

4. Fault tolerance and high availability.

5. Parallel Linux clusters can offer some advantages over a commercial super computer that go beyond simple price comparison. Unlike a

commercial super computer, a cluster can be made more powerful, increase its performance by adding new nodes.

6. It saves the waiting time (zero down time)

7. Upgrading a Linux cluster is very easy. It is as easy as buying whatever the “latest, greatest” off-the-shelf system and it in to the cluster, the cluster can be upgraded as a whole.

8. An excellent use for Linux clusters is the creation of high availability web servers and other mission critical availability system.

PROPOSED LINE OF ATTACK

2. PROPOSED LINE OF ATTACK

Load balancing distributes traffic efficiently among networks servers so that no individual server is overburdened. This vendor-neutral guide to the concepts and terminology of load balancing offers practical guidance to planning and implementing the technology in most environments. This helps to insulate the individual machines within the cluster, and allows the cluster, and allows the cluster to dole out traffic to the least busy servers. The load balancing is called as the massively parallel processing machine. It can be used for all parallel programming applications. The effective form of load balancing is by assigning a single host name to several IP addresses (different machines)

Cluster computers provide a low-cost alternative to multiprocessor systems for many applications. Building a cluster component is within the reach of any computer user with solid C programming skills and a knowledge of operating systems, hardware and networking. A cluster computer is a multi computer; network of node computers running distributed software that makes them work together as a team. Distributed software turns a collection of network computers into a distributed system. It presents the user with a single-system image and gives the system its personality. Software can turn a network of computers into a transaction processor, a supercomputer, or even a novel design of your own. By combining the power of many workstation-class or server-class machines, performance levels can be made to reach supercomputer levels, for a much lower price than traditional super computer can offer.

By breaking the problem down into tasks that can be done in parallel, the computers in a high performance cluster can share the load and complete the problem more quickly. Performance clustering works in a similar manner to traditional Symmetric Multiprocessor (SMP) servers.

PROPOSED METHODOLOGY

3. PROPOSED METHODOLOGY

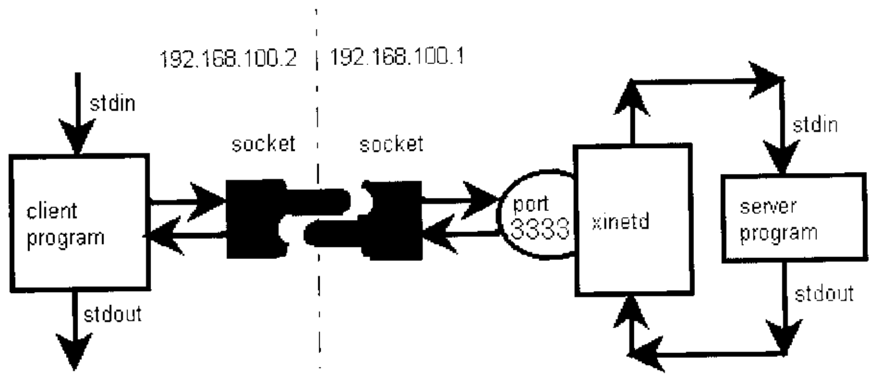
Load balancing mechanism involves two phases

- Process scheduling
- Process migration

Process scheduling involves the process of finding which process to migrate and to which processor it should migrate. Process migration is the actual transfer of the process from the source processor to the destination processor.

This load balancing algorithm strives to balance the load on all nodes at all times. A processor may be in any of the three states: heavy load, normal load or light load. A processor in light load may accept a migrant process, while a processor in heavy load can have its processes migrate to another processor. A processor in normal load needs no balancing effort.

CONNECTION ESTABLISHMENT



The stdin is the input (the set of commands) to the first processor. The connection is established through the sockets of the other processors with which the load is to be shared. The connection is established by using their IP address. The connection takes place through the port no. 3333 and the commands are taken there for execution. The commands are executed and the result is sent back to the processor as stdout.

SYSTEM DESIGN

4. SYSTEM DESIGN

Design is a meaningful engineering representation of something that is to be built. It can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria for good design. In software engineering context design focuses on four major areas of concern.

1. **Architecture**

   The architectural design defines the relationships between the major structural elements.

2. **Interfaces**

   The interfaces design describes how the software communicates within itself, with systems that interoperate with it, and also with the human beings who use it.

3. **Components**

   The component level design transforms structural elements of the software architecture to procedural description of the software components.

Design is the process through which the requirements are translated into blueprint for constructing the software. The design must have some characteristics

.

1. The design must implement all of the explicit requirements contained in the analysis model and it must accommodate all the implicit requirements.

2. The design must be a readable, understandable guide for those who generate code and for those who test & subsequently support the software.

The design should provide a complete picture of the software, functional and behavioral domains from an implementation perspective.

PROGRAMMING ENVIRONMENT

5. PROGRAMMING ENVIRONMENT

The hardware and software configurations that were used to develop this system are given below.

5.1 Hardware Requirements

Server Side Configuration

<u>Front Server Machine</u>

Processor	: Intel Pentium III Processor @ 2GHz
Memory	: 128 MB RAM
Hard Disk	: 10 GB
Keyboard	: Linux Compatible
Floppy	: 3 1/2 inch Floppy Drive
Mouse	: PS/2 Port Mouse
Monitor	: 15" Color Display
NIC	: Ethernet Card

Client Side Configuration

<u>Real Server Machine</u>

Processor	: Intel Pentium III Processor @ 900 MHz
Memory	: 128 MB RAM
Hard Disk	: 10 GB
Keyboard	: Linux Compatible

Floppy	: 3 1/2 inch Floppy Drive
Mouse	: PS/2 Port Mouse
Monitor	: 15" Color Display
NIC	: Ethernet Card – 2 Nos.

<u>Network Setup Hardware</u>

Patch Cable	: CAT 5E UTP
NIC	: Ethernet Hub / Switch – 8/16 ports

5.2 Software Requirements

Operating System	: Red Hat Linux 9.0
Language	: "C" with Shell Scripting.

DETAILED DESIGN

6. DETAILED DESIGN

Load balancing is a concept in which the group of processors shares their jobs. It is the ability to preempt the process on one machine and reactivate the process on another machine. Load balancing is a service on one IP address over multiple servers without generating a single point of failure. They take traffic destined for a machine from the server and route it to an available server based on the configuration you require. Traffic destined on our server is initially forwarded to one or more servers with highest priority setting. When all servers at this level have failed, traffic is forwarded to the server(s) with the next highest priority. In the event of a highest priority machine returning to service, traffic is forwarded to this machine and lower priority machine returns to a dormant state.

By combining the power of many workstation-class or server class machines, performance levels can be made to reach supercomputer levels. For a much lower price than a traditional super computer can offer, which require expensive specialized hardware and software. Think of a high traffic web site, they are not really a single server. There will be a default gateway and many other servers connected to it. The client communicates through the default gateway. At its core, clustering technology has two basic parts.

In load balancing there is a master node to which many compute nodes or slaves are connected. There can be many requests to a particular node and the server responds correspondingly. The first component, made up of a customized operating system (such as kernel modifications made to Linux), special compilers and applications allows programs to take full advantage of clustering. The second component is the hardware inter connect between machines (nodes) in the server cluster. These interconnects are often highly specialized interfaces.

This can be implemented in three modules. They are

1. **Job Server**
2. **Job Dispatcher**
3. **Client module**

Job Server

In Job server there are two primary programs, they are

1. **Job Scheduler**
2. **Job Splitter**

Job Scheduler:

It is used to take the traffic destined for the server. It gets the machine data at first. It keeps track of the number of nodes and the IP addresses of the nodes. It is also possible to schedule these processes on the remote machines, so that not more than one process per machine is active at any one time.

This was designed to combat problems with using sequences for parallel builds. When building in sequence, it is possible that a node receives a task that will take much longer time than for the other nodes to complete. It is also possible that as other nodes finish their jobs faster, the node which has been bogged down, is handed another job. When performing larger parallel builds, very slow machines will stall at entire build, as they attempt to compile many objects at once, and usually at this point swapping is performed.

The job scheduling in cluster can prevent this in two ways. The job scheduling will not allow a node to process any more than one command at the same time. If more commands than nodes are requested, the excess commands will block until a node has been freed.
Second the scheduler has the ability to register a benchmark number of some sort for each node. This allows the scheduler to always give out the fastest of the remaining nodes whenever one is requested. This allows a parallel build to more efficiently utilize a heterogeneous cluster.

Job Splitter

Traffic is evenly split between servers depending on a predetermined load ratio. The master node gets the jobs and these jobs are split according to the number of slave nodes or computer nodes. In this the traffic from the different clients is split between different nodes, connected to the master node. The different compute nodes according to the jobs split among slave nodes carry out the processing of each job. If the number of jobs is more than one in a node the excess jobs are in a dormant sate until the node is free.

Job Dispatcher

The jobs after processing are stored in the master node as files with job name as job combined with the IP address of corresponding compute node, where the job is carried out. When we view the jobs file we may get the information of what are the jobs carried out by the respective compute nodes. We have all information about the different nodes connected and the IP addresses of the compute nodes and what are the jobs performed by the respective compute nodes. It also has the information of where the output for the job is stored.

Client module

After the processing if a client wants the output from a particular compute node to be viewed, the name of the job file can be specified to view the jobs. The output of the job will be in the respective compute node, where the client had requested. These modules are implemented in the Linux platform with the help of C programming language and shell scripting.

Good programmers optimized their programs for speed because computer power is a valuable commodity. There is a limited amount of speed available in each computer, and not a lot of ways to connect them together to allow groups of computers to accomplish a task cooperatively.

Clusters are in fact quite simple. They are a bunch of computers tied together with a network working on many problems. Parallelism is the quality that allows something to be done in parts that work independently rather than a job that has a so many interlocking dependencies that it cannot be further broken down.

With load balancing clustering model, the number of users (or the number of transactions) can be allowed across a number of application instances, so as to increase transaction throughput. This model easily scales up as the overall volume of users and transactions goes up. The scale-up happens through simple applications replication only, requiring little or no application modification or alteration.

One of the most important features of load balancing will be the ability to set up the systems by which the users can run the job non-interactively. And, since this is a clustered environment, it is also necessary that those non-interactive jobs be delivered to lightly loaded nodes in the cluster or even to split them up among nodes.

There are lots of commercial load balancing and queuing systems but they require some tools such as, Load Lever for IBM computers and Cluster from Active tools. All these allow users to have what could be considered a traditional batch system that could be used for some particular organization or some particular systems. For example, Load Lever is a traditional batch system that might be found on an IBM mainframe. So we have to build a fully functional cluster, a system that can support a fully functional batch system.

Cluster technology helps to bring availability and performance to the traditional centralized server model. Formalized service levels within and between businesses are becoming standard practices. Availability, performance and reliability are chief among the metrics measured. Continuous availability on a technical level tends to reduce business-planning costs.

FUTURE ENHANCEMENTS

7. FUTURE ENHANCEMENTS

The project "Load Balancing" has been programmed in "C" and developed under Red Hat Linux. The clustering can be implemented in other Linux packages like SuSe, Debian, Mandrake and other available packages. This can be further developed in case of high availability clusters, which could be used to solve all the big business needs.

- If an application error occurs, the application restarts itself on the same node and corrects any potential cause for error (such as corrupt control data).

- If an application performs some amount of checkpoint-restart processing, the application shall be viewed as if it is close to the point of failure.

- If a system outage occurs, the application is restarted on a backup, and it can be viewed as though it had started for the first time.

CONCLUSION

8. CONCLUSION

Load balancing technology helps to achieve availability, high and better performance to the traditional data center model. Formalized service levels within corporate and between businesses will become standard practice. Availability, performance and reliability are chief among the metrics measured. Continuous availability on a technical level tends to reduce business-planning costs. This would include application availability, batch versus backup contention, decision support systems, use of assets for something other than just disaster recovery and point-in-time reconstruction. These issues lead to cost avoidance.

Load balancing distributes traffic efficiently among network servers so that no individual server is overburdened. This vendor-neutral guide to the concepts and terminology of load balancing offers practical guidance to planning and implementing the technology in most environments. Clustering elegantly solves the problem of integrating servers along with centralized network and storage resources. Clustering technologies provide benefits far beyond the mere historical challenge of high-end scalability. These benefits include extreme availability, redundancy/resiliency/replication, backup, and fail over, automated cross-platform systems management, parallel processing, resource sharing.

REFERENCES

9. REFERENCES

1. David HM Spector, ***Building Linux Clusters***, O'Reilly Publications

2. Alex Vrenios, ***Linux Cluster Architecture***, Sam's Publications

3. Charles Bookman, ***Linux Clustering: Building and Maintaining Linux Clusters***, New Riders Publication.

4. Elis Awad, ***System Analysis and Design***, Galgotia Publication.

5. Maurice J. Bach, ***The Design of the Unix Operating System***, O'Reilly Publications

6. www.linux.org
7. www.linuxvirtualserver.org
8. www.coyotepoint.com
9. www.linuxhowto.com
10. www.ctssn.org

APPENDICES

10.1 SAMPLE CODES

Client.c

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<errno.h>

extern int errno;
#define QKEY (key_t)0100
#define QPERM 0666
#define MAXOBN 500
#define MAXPRIOR 10

struct q_entry {
  long mtype;
  char mtext[MAXOBN+1];
};


int init_queue() {
  int queue_id;
  if(queue_id = msgget(QKEY, IPC_CREAT | QPERM) == -1) {
    perror("msgget failed");
    return(queue_id);
  } }

warn(char *s) {
  fprintf(stderr, "client (client.c) Warning: %s\n", s);
}

static int s_qid = -1;
enter(char *objname, int priority) {
  char *strncpy();
  struct q_entry s_entry;
  int len;
```

```
 if((len = strlen(objname)) > MAXOBN) {
  warn("Name too Long...");
  return -1;
 }

 if(priority > MAXPRIOR || priority < 0) {
  warn("Invalid priority Level");
  return -1;
 }
 if(s_qid == -1 && (s_qid = init_queue()) == -1)
  return -1;

 s_entry.mtype = (long)priority;
 strncpy(s_entry.mtext, objname, MAXOBN);
 printf("message is %s\n", s_entry.mtext);
 if(msgsnd(s_qid, &s_entry, len, 0) == -1) {
  perror("Msgsnd Failed");
  return(-1);
 }
 else    return 0;
}

main(int argc, char *argv[]) {
 int priority;
 if(argc != 3) {
  fprintf(stderr, "USAGE: %s command priority!\n", argv[0]);
  exit(1);
 }

 if((priority = atoi(argv[2])) <= 0 || priority > MAXPRIOR) {
  warn("Invalid priority");
  exit(2);
 }

 if(enter(argv[1], priority) < 0) {
  warn("Enter Failure");
  exit(3);
 }

 exit(0); }
```

ClusterManager.sh

```
printChar() {
 num=0;
 while [ $num -lt $2 ]
 do
   echo -n $1
   num=`expr $num + 1`
 done
 echo
}

printLine() {
 num=0;
 while [ $num -lt $1 ]
 do
   echo
   num=`expr $num + 1`
 done
 echo
}

function mainMenu() {
 clear
 printLine 5
 printChar \# 79
 printf "\n\n\t\t\t1. Add Cluster Member(s)\n";
 printf "\t\t\t2. Submit job(s)\n";
 printf "\t\t\t3. Split Jobs\n";
 printf "\t\t\t4. Start ClusterServer\n";
 printf "\t\t\t5. Quit\n\n";
}

function quitApp() {
 printChar \# 79
 echo -e "\t\t\tThank you for using Cluster....\n\n\n";
}

printChar \# 79
printf "Starting Cluster system please wait.....\n"
```

```
sleep 2


while true
do
 mainMenu
 echo -e "\nWhat do you want to perform.............?\c: "
 read opt
 case $opt in
  1)
        sh AddNodes.sh;;
  2)
        sh JobReceiver.sh;;
  3)
        sh JobSplitter.sh;;

  4)
        sh StartServer.sh;;

  5)
        quitApp;
        exit;;

  *)
        echo "Sorry I couldn't get that? ";
 esac
  echo -n "Do you want to continue...say (y/n): "
  read reply
  if [ "$reply" != "y" ];then
    quitApp
    break;
  fi
 done
```

NFS Configurator.c

```
//
// This program configures NFS Server
//

#include<stdio.h>

void nfsConfigurator() {
 FILE *fp = fopen("/etc/exports", "a+");
 char shareName[300], ipAddress[50];
 char sharePermissions[3];

 printf("\n\nEnter the Share Name: ");
 scanf("%s", shareName);
 printf("\n\nEnter the IP range where it has to shared ");
 printf("\nFormat <ip>/<netmask): sample
192.168.200.0/255.255.255.0\n");
 scanf("%s", ipAddress);
 printf("\nEnter Share permissions (rw, ro): ");
 scanf("%s", sharePermissions);

 fprintf(fp, "%s\t\t%s(%s)\n", shareName, ipAddress, sharePermissions);

 // Close the file
 fclose(fp);
 printf("\nPlease wait setup is initializing....");
 printf("\nThis may take few seconds....\n");
 system("service nfs restart &> /dev/zero");
 system("echo YES boss its ready for use....!!!");
} // end nfsConfigurator

int main() {
 nfsConfigurator();
 return 0;
}
```

ClustClient.c

```
#include     "unp.h"
int main(int argc, char **argv)
{
      FILE *fp;
     int sockfd, n;
      int priority = 4;
      char cmdString[15];
      char recvline[MAXLINE + 1];
      struct sockaddr_in servaddr;

      if (argc != 2)
              err_quit("usage: a.out <IPaddress>");

      if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
              err_sys("socket error");

      bzero(&servaddr, sizeof(servaddr));
      servaddr.sin_family = AF_INET;
      servaddr.sin_port   = htons(13);/* server */
      if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
              err_quit("inet_pton error for %s", argv[1]);

      if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
              err_sys("connect error");

      while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
              recvline[n] = 0;     /* null terminate */

              // THIS following line makes the commands to be available in
              // a file named 'commands'
           fp=fopen("commands", "w");
              fprintf(fp, "%s", recvline);
              sprintf(cmdString, "%s", "sh commands");
           system(cmdString);
      }
      if (n < 0)
              err_sys("read error");
      exit(0); }
```

Server.c

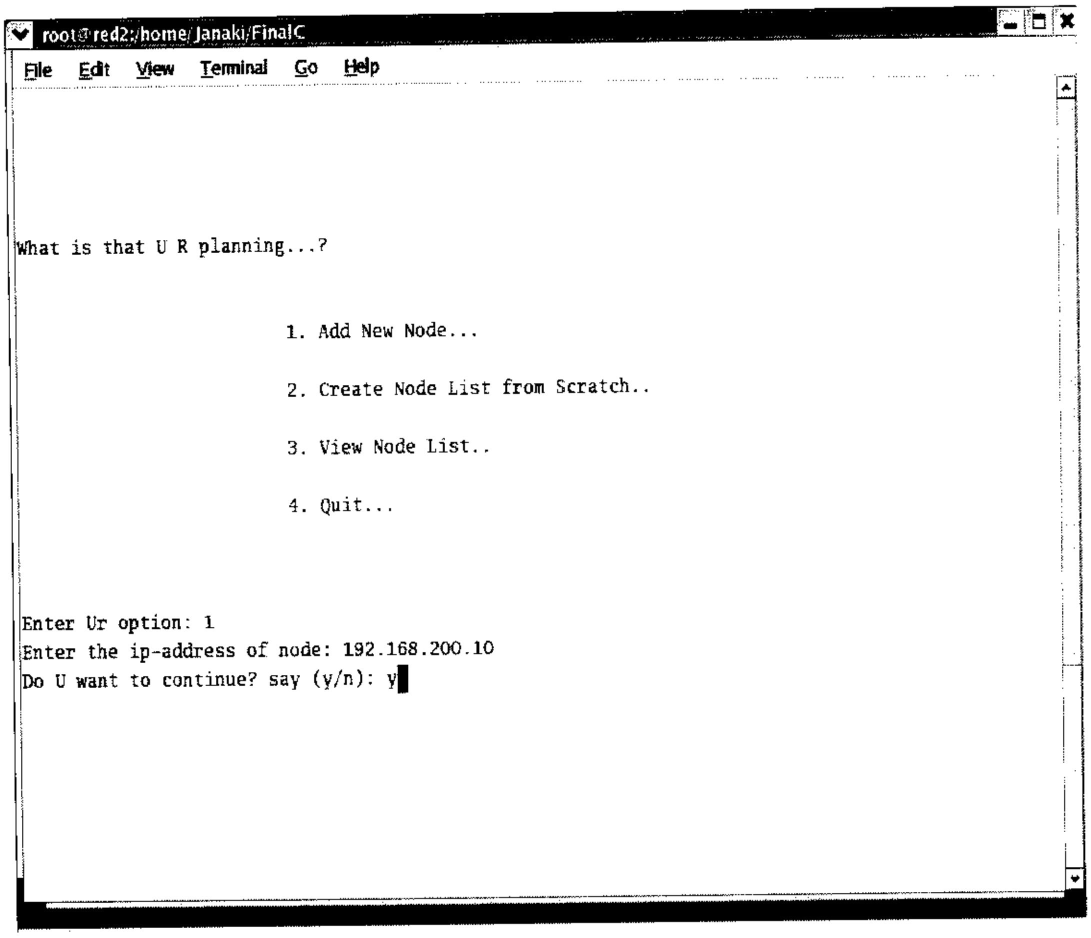
```
#include<stdio.h>
#include<fcntl.h>
#include<signal.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/errno.h>


#define QKEY (key_t)0100
#define QPERM 0666
#define MAXOBN 500
#define MAXPRIOR 10

struct q_entry {
  long mtype;
  char mtext[MAXOBN+1];
};

static int r_qid = -1;
int fd;

void usr1() {
  pause();
}

void usr2() {
  signal(SIGUSR1, usr1);
  signal(SIGUSR2, usr2);
}

server() {
  long pro_type;
  struct q_entry r_entry;
  int mlen, fd;
  char buf[10];
  fd = creat("ser.txt", O_WRONLY | O_CREAT | O_EXCL | 0644);
  if(fd < 0) {
```

```
  printf("Server already installed!!\n");
  exit(0);
 } else {
  printf("Server process PID is %d\n", getpid());
 }

 sprintf(buf, "%d", getpid());
 write(fd, buf, sizeof(int));
 chmod("ser.txt", 0400);

 if(r_qid == -1 && (r_qid = init_queue()) == -1) {
   return -1;
 }
 signal(SIGUSR1, usr1);
 signal(SIGUSR2, usr2);

 for(;;) {
   if((mlen = msgrcv(r_qid, &r_entry, MAXOBN, 0, MSG_NOERROR)) ==
-1) {
     perror("msgrcv failed");
     return -1;
   } else {
     r_entry.mtext[mlen] = '\0';
     proc_command(r_entry.mtext);
   }
  }
}

proc_command(char *msg) {
//  printf("command is '%s\n", msg);
  system(msg);
}



init_queue() {
  int queue_id;
  /* Attempting to Create Message Queue */
  if(queue_id = msgget(QKEY, IPC_CREAT | QPERM) == -1) {
    perror("msgget failed");
    return queue_id;
  }
}

void warn(char *s) {
  fprintf(stderr, "Warning: %s\n", s);
}

main() {
  int pid = fork();

  switch(pid) {
    case 0:
      setpgrp();
      server();
      break;

    case -1:
      warn("Fork to start server failed!!!\n");
      break;
  } // end switch
  exit(pid != -1 ? 0 : 1);
} // end main
```

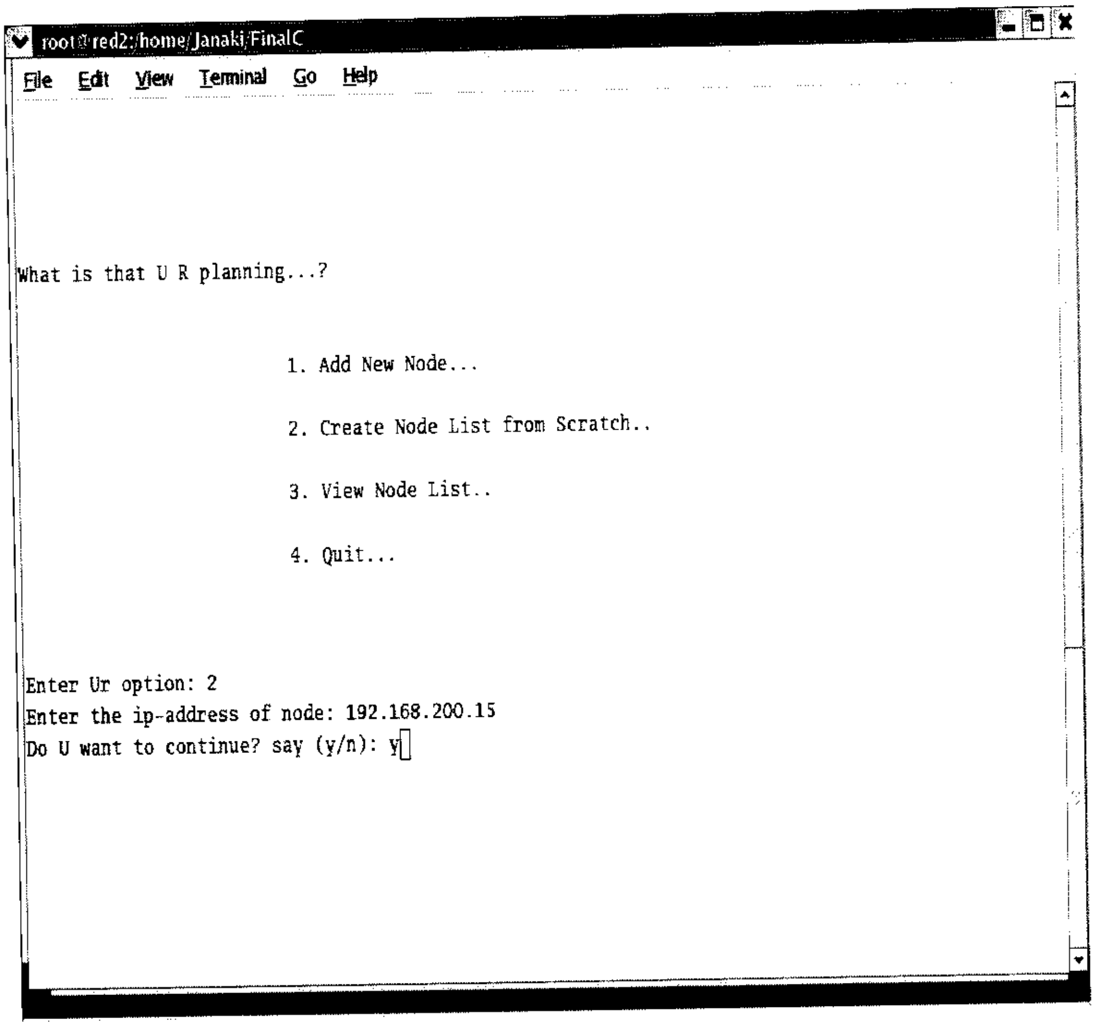
JobReceiver.sh

```
printChar() {
 num=0;
 while [ $num -lt $2 ]
 do
  echo -n $1
  num=`expr $num + 1`
 done
 echo
}

submitJob() {
 rm -f scheduled
 lineNumber=1
 totalLines=`wc -l jobFile | tr -s " " | cut -f2 -d" "`;
  while [ $lineNumber -le $totalLines ]
 do
  command=`sh PrintLine.sh $lineNumber jobFile`;
  cmd=`echo $command | cut -f1 -d" "`
  echo "$command &> /mnt/clusterOutput/$cmd.output" >> scheduled
  lineNumber=`expr $lineNumber + 1`;
 done }
printChar + 79
echo
rm -f cmdFile
echo "Start entering commands.....<type 'stop' when you are done with
input>.."
while true
do
 read cmdStr
 if [ "$cmdStr" = "stop" ]
 then
  break;
 fi
 echo $cmdStr >> cmdFile
done
cat cmdFile | cut -f1- -d" " > jobFile

submitJob
```

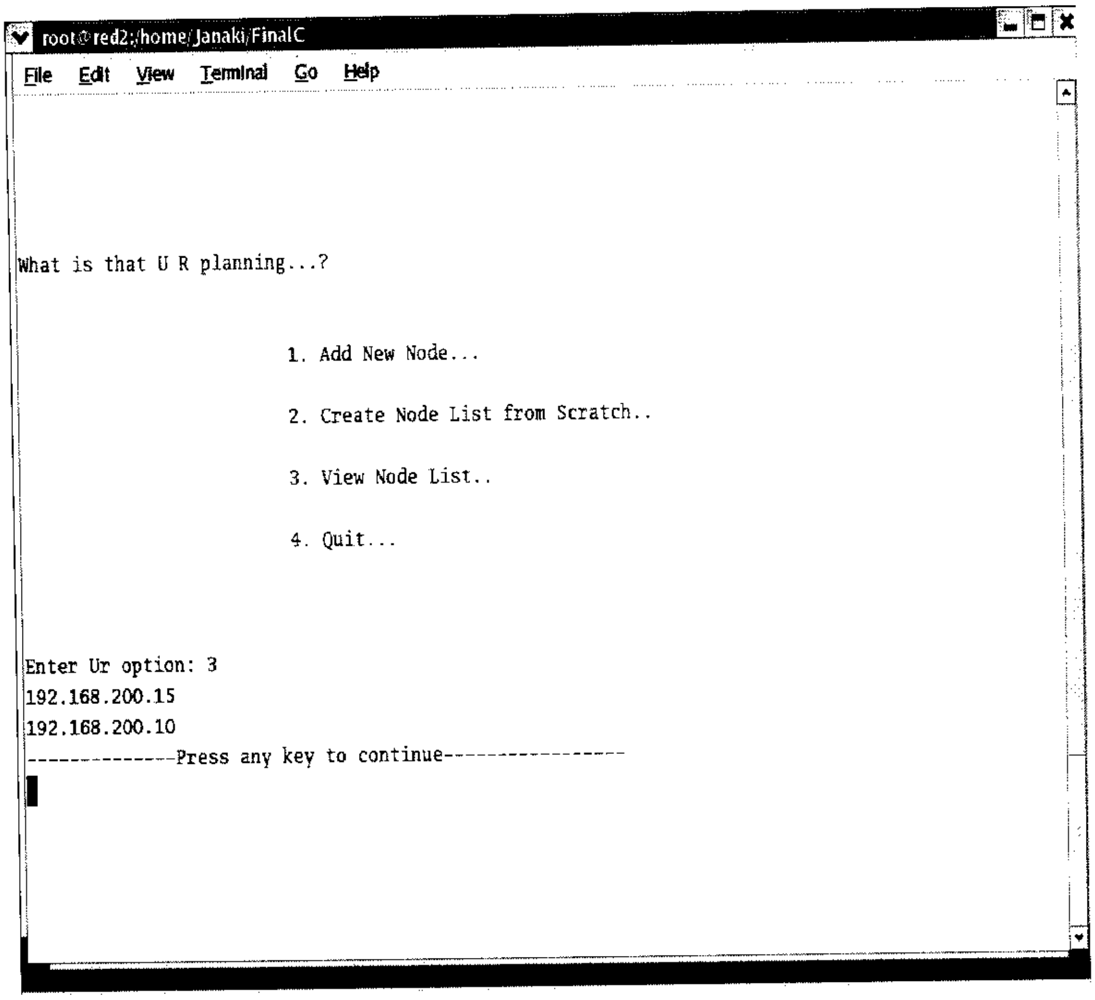
10.2 RESULTS


root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help

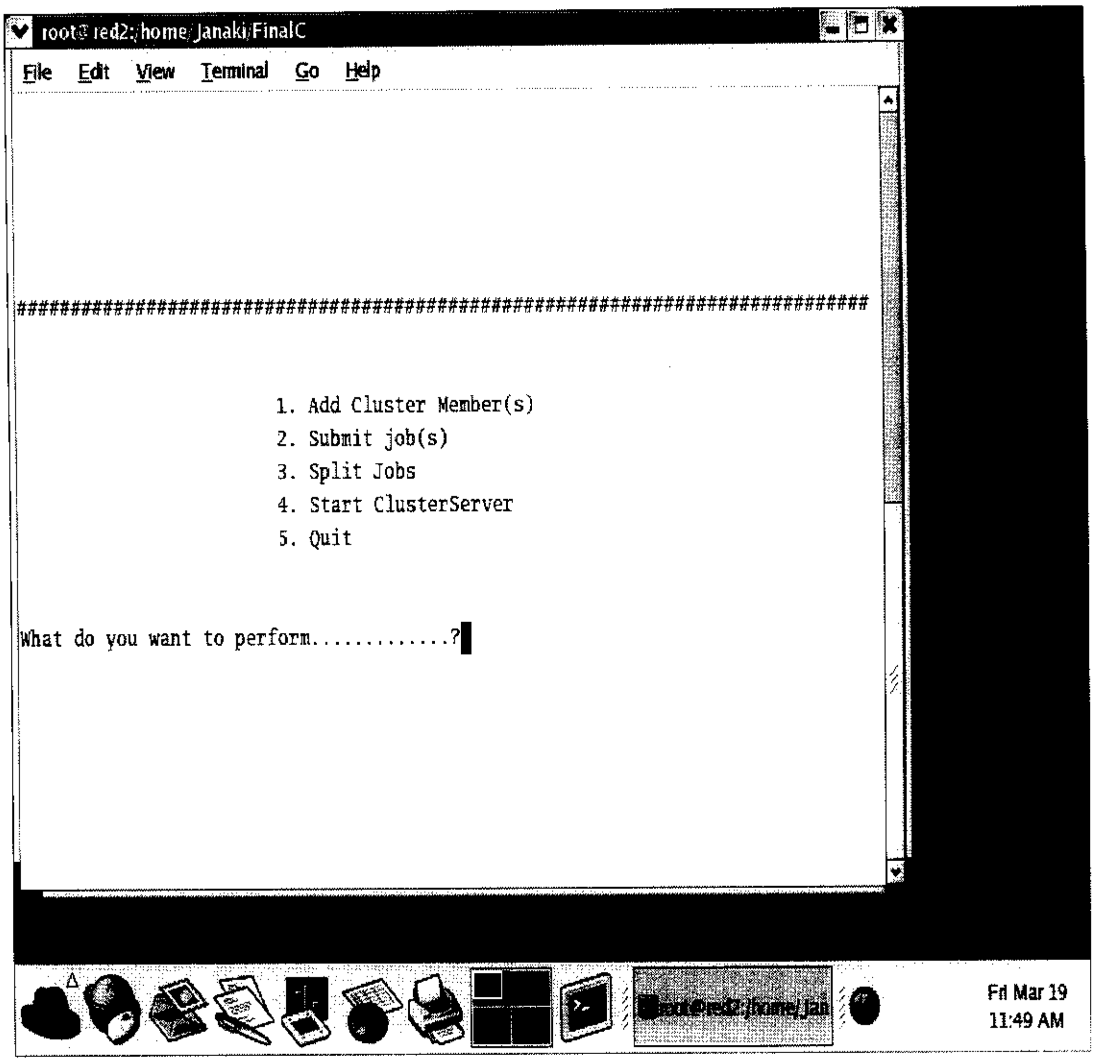
```
What is that U R planning...?


                        1. Add New Node...

                        2. Create Node List from Scratch..

                        3. View Node List..

                        4. Quit...



Enter Ur option: 1
Enter the ip-address of node: 192.168.200.10
Do U want to continue? say (y/n): y
```


root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help

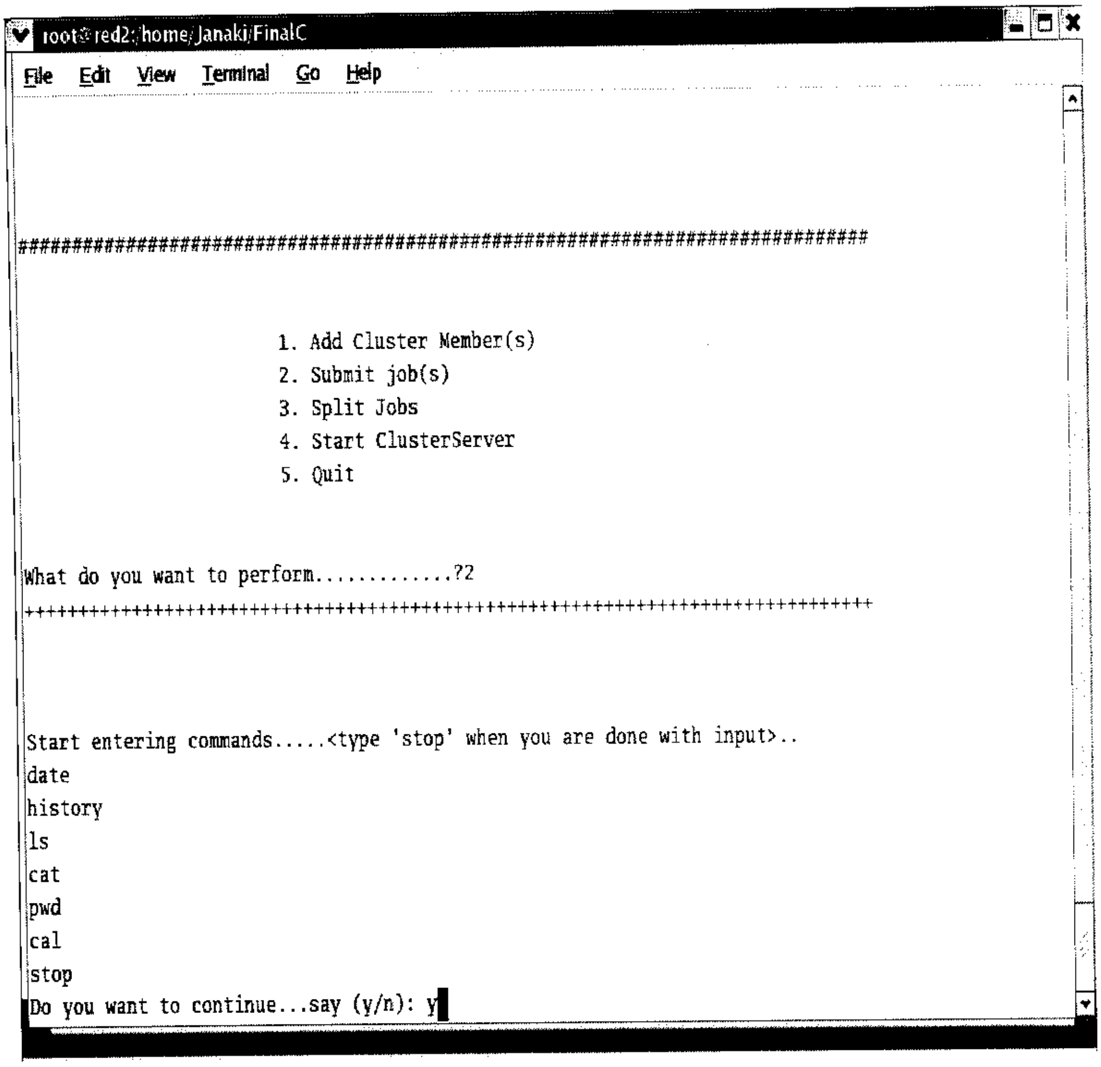
```
What is that U R planning...?


                         1. Add New Node...

                         2. Create Node List from Scratch..

                         3. View Node List..

                         4. Quit...



Enter Ur option: 2
Enter the ip-address of node: 192.168.200.15
Do U want to continue? say (y/n): y
```


root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help

```
What is that U R planning...?


                        1. Add New Node...

                        2. Create Node List from Scratch..

                        3. View Node List..

                        4. Quit...



Enter Ur option: 3
192.168.200.15
192.168.200.10
--------------Press any key to continue-----------------
```

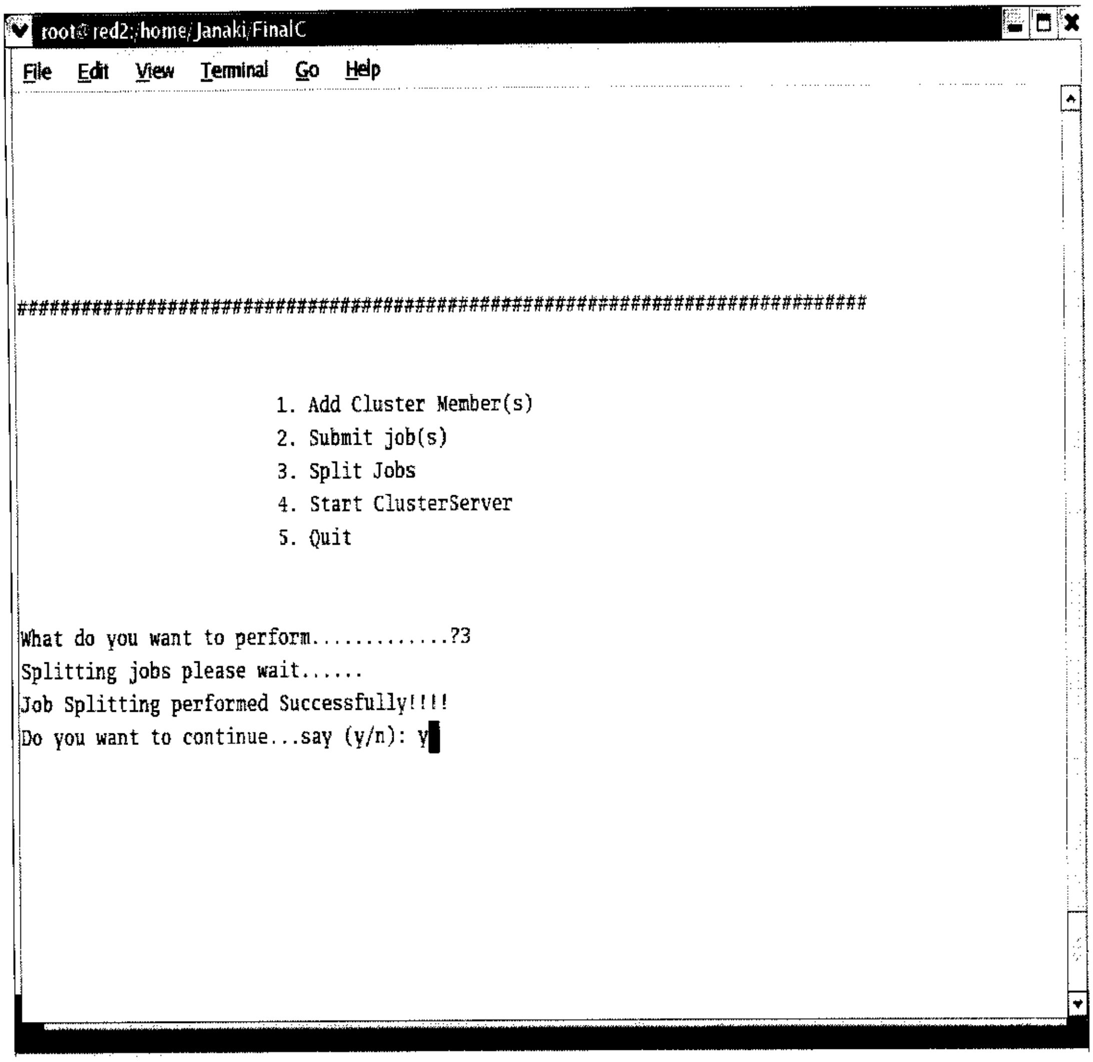

root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help
root@red2:/home/Jan
Fri Mar 19
11:49 AM

```
##################################################################################



                    1. Add Cluster Member(s)
                    2. Submit job(s)
                    3. Split Jobs
                    4. Start ClusterServer
                    5. Quit


What do you want to perform............?
```

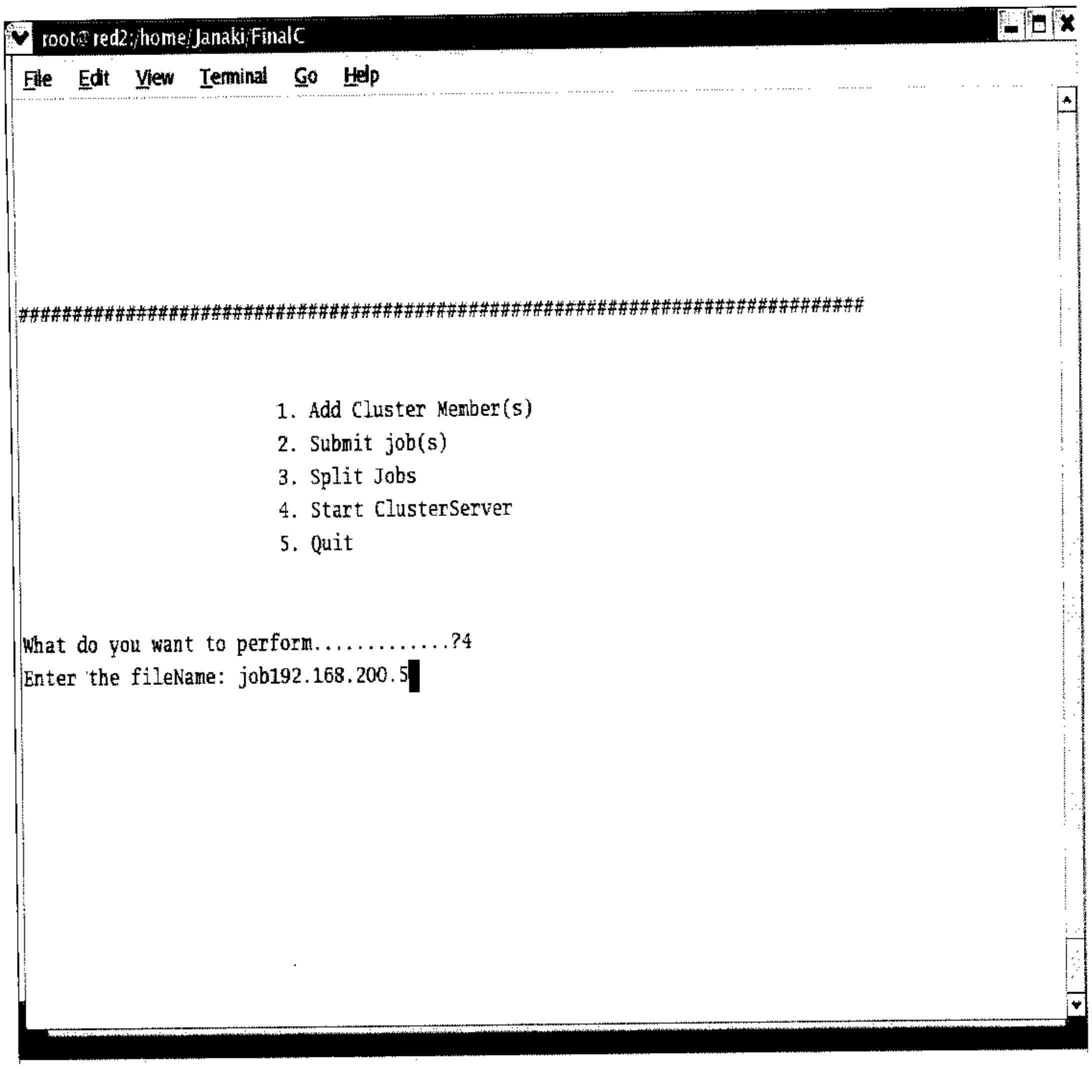


root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help

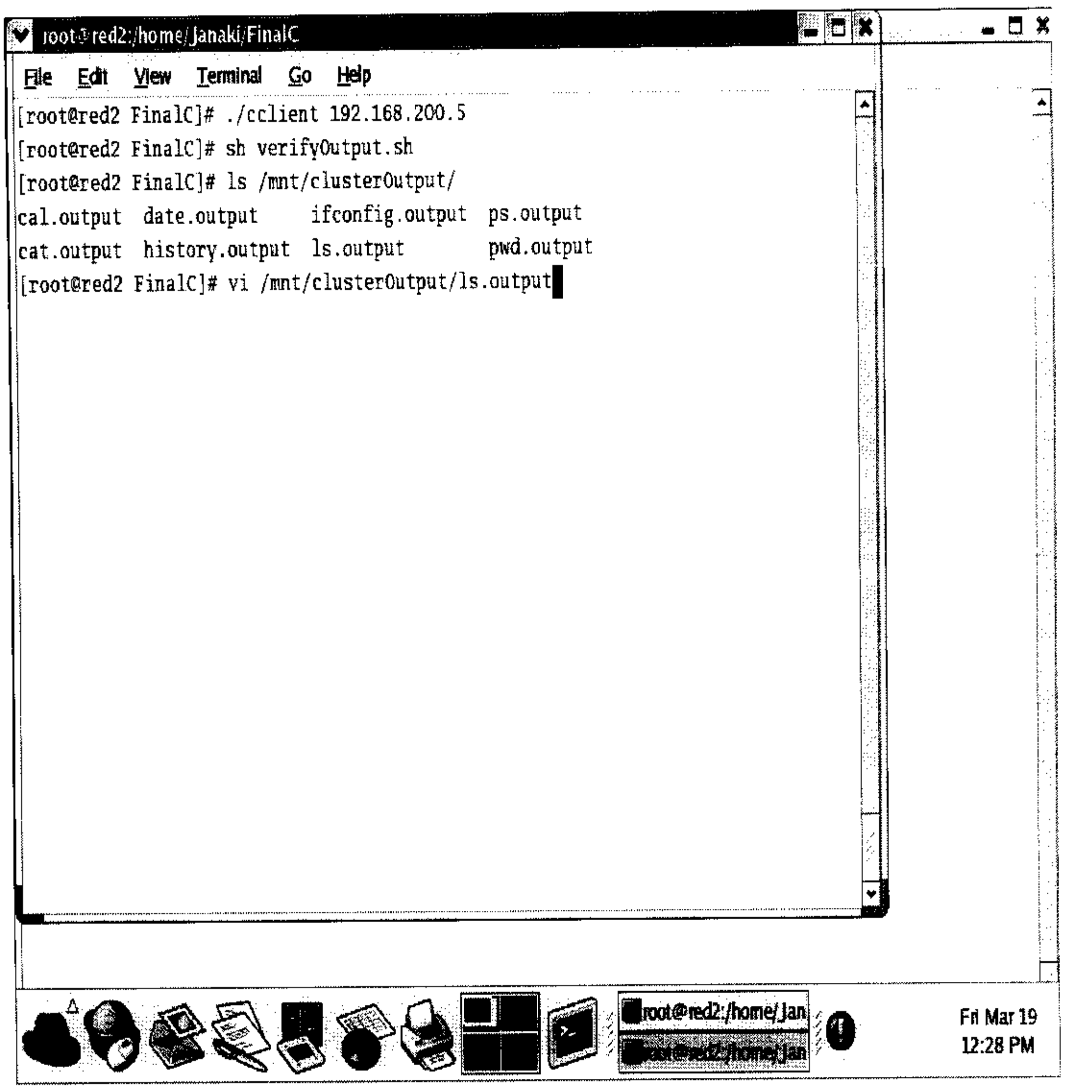
```
#################################################################################

                         1. Add Cluster Member(s)
                         2. Submit job(s)
                         3. Split Jobs
                         4. Start ClusterServer
                         5. Quit


What do you want to perform............?2
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++



Start entering commands.....<type 'stop' when you are done with input>..
date
history
ls
cat
pwd
cal
stop
Do you want to continue...say (y/n): y
```


root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help

```
################################################################################

                        1. Add Cluster Member(s)
                        2. Submit job(s)
                        3. Split Jobs
                        4. Start ClusterServer
                        5. Quit


What do you want to perform............?3
Splitting jobs please wait......
Job Splitting performed Successfully!!!!
Do you want to continue...say (y/n): y
```


root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help

```
################################################################################


                        1. Add Cluster Member(s)
                        2. Submit job(s)
                        3. Split Jobs
                        4. Start ClusterServer
                        5. Quit


What do you want to perform............?4
Enter the fileName: job192.168.200.5
```



root@red2:/home/Janaki/FinalC
File Edit View Terminal Go Help
root@red2:/home/Jan
Fri Mar 19
12:28 PM

```
[root@red2 FinalC]# ./cclient 192.168.200.5
[root@red2 FinalC]# sh verifyOutput.sh
[root@red2 FinalC]# ls /mnt/clusterOutput/
cal.output  date.output     ifconfig.output  ps.output
cat.output  history.output  ls.output        pwd.output
[root@red2 FinalC]# vi /mnt/clusterOutput/ls.output
```