A PROJECT REPORT

ON

IMPROVED CLUSTER MANAGER FOR LINUX BASED EXECUTABLE AND LINKABLE FORMAT FILES OVER BEOWULF CLUSTER

Submitted by

B80058505 Bhushan Jain
B80058517 Akshay Thorat
B80058565 Shrijeet Aherkar
B80058613 Mahesh Baheti

Guided by

Asst. Prof. Mrs. D. D. Londhe

DEPARTMENT OF INFORMATION TECHNOLOGY

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Pune

University of Pune

2013-2014

Department of Information Technology

Certificate

This is to certify that,

B80058505:-Bhushan Jain

B80058517:-Akshay Thorat

B80058565:-Shrijeet Aherkar

B80058613:-Mahesh Baheti

have successfully completed the Project entitled IMPROVED CLUSTER MANAGER FOR LINUX BASED EXECUTABLE AND LINKABLE FORMAT FILES OVER BEOWULF CLUSTER, under my guidance in partial fulfillment of the requirement for the award of the Bachelors of Engineering in Information Technology of Pune Institute of Computer Technology by University of Pune for the academic year 2013-2014.

Date:

Place:

Prof. D.D.Londhe
Project Guide

Dr. Emmanuel M.
HOD

Dr. P. T. Kulkarni
Principal

Acknowledgement

We take this opportunity to thank our project guide Prof. D.D.Londheand Head of the Department Dr. Emmanuel M. for their valuable guidance and for providing all the necessary facilities, which were indispensable in the completion of this project report. We are also thankful to all the staff members of the Department of Information Technology of Pune Institute of Computer Technology, Pune for their valuable time, support, comments, suggestions and persuasion. We would also like to thank the institute for providing the required facilities, Internet access and important books. We also thank Mr. Kiran Divekar of GEEP and other mentors from GEEP for their guidance which steered the development of this project in right direction.

Bhushan Jain
Akshay Thorat
Shrijeet Aherkar
Mahesh Baheti

Abstract

As user level applications are becoming more and more functional they require high-end system resources like GPU and physical memory. But some personal systems are still not sufficient to fulfil these requirements due to multitasking nature of operating system. Cluster computing can be considered as an effective solution to this .Cluster architecture can provide access to resources from other personal systems over networks. Beowulf cluster consists of personal computers connected by a small network and it is built using software utilities like OpenMPI, ssh, rexec on Linux operating system. But setting up a cluster is a tedious job for novice user and even after setting up a cluster load distribution is done without checking resources available at the slave PCs. This causes degradation of overall system performance. This project aims at providing easy setup of Beowulf cluster over small personal LANs for novice personal computer users. Also by checking resources available at slave PCs it aims at providing efficient cluster management at lower levels in Linux kernel (any device) and providing better execution environment using ORTE/OpenMPI.

KEYWORDS : resource sharing, cluster, Beowulf, Linux, ELF

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays more and more programming languages and APIs are being developed to design application level programs. These technologies are growing in number, hence, enabling developers develop applications with more and more functionalities. As more and more functionalities are integrated into applications the use of resources is increased. Lets take an example of graphic designing application, Maya 3D Animation Software. The minimum system requirements of Maya includes, 64-bit Intel or AMD multi-core processor 4 GB of RAM minimum (8 GB recommended)[1]

This implies that even a small business owner, such as, graphic designer needs high end computer system.

Not only computers used for professional purposes need high-end computer systems, but applications used for daily multimedia purposes require large amount of resources. Users are reporting[2] insufficient resources on systems consisting 64 bit architecture and 4 GB RAM. Due to multitasking nature of operating systems the effective resource usage increases as tiny accessories like Rainmeter also take considerable amount of physical memory and GPU.

Above issues imply requirement of high-end resources even on simple personal computer. These days it is not hard to find a end-user having multiple personal computers as technology change drives user to buy new systems time to time.

Once this end-user gets a new better equipped computer system, that individual hardly uses his old system. Although it is perfectly usable and it can compensate the extra resources necessary by the applications.

Above discussion presents a necessity for a system which enables an end-user to compensate extra resource necessity through multiple computer systems consisting same or different configurations.

This system can be configured and built using distributed computing.

Distributed computing consists of two types computer architectures, grid systems and cluster systems. Grid systems imply multiple computer systems connected through large networks and residing at different geographic systems.[3] By definition grid doesnt suffice the requirement mentioned earlier.

Cluster computer architecture consists of multiple hosts connected by small networks and residing in same geographic location. These hosts may be connected using small LANs either by wire or wireless. This definition exactly suffices the requirements mentioned earlier. Hence, we proceed with clusters for our proposed system.

After digging for cluster architectures we come to know a cluster architecture built using off- the-shelf hardware in early 2000s called Beowulf.[4] Beowulf cluster consists of personal computers connected by small networks and built using software utilities like OpenMPI, ssh, rexec on Linux operating systems. The basic hierarchy consists of one master and multiple slaves.

Although, Beowulf cluster architecture suffices the requirement of a system which enables an end-user to compensate extra resource necessity through multiple computer systems consisting same or different configurations. It is tedious to set up. Every package required should be installed and configured by end-user manually and in process of this setup the user may lose interest to build the system due to its confusing and tedious nature. Simplifying this process is necessary and this project enables user to complete this process with few clicks of buttons or with few commands at command-line.

After a cluster is setup a mechanism is necessary for communication between processes running on different nodes. This is enabled by using Message Passing Interface (MPI) which is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. Beowulf cluster uses the open-source counterpart of MPI i.e. OpenMPI. Although, the OpenMPI project completes all the tasks required for an application to run on a cluster, it is efficient for applications developed using MPI APIs like OpenMPI itself. This raises a requirement for intelligent and efficient execution mechanisms for non-parallel applications.

This proposed system also consists of an efficient and intelligent cluster management utility to complete the tasks of execution and management of load. This discussion presents an introductory overview of proposed system which can be summarised as an easy, efficient, automated and intelligent Beowulf cluster, hence, named as Hands-Down Beowulf (HD Beowulf).

Chapter 2

Literature Survey

The basic idea was to share resources between systems connected on a personal small network. This can be enabled by distributed computing, which enables users to create combined systems of multiple machines for parallel computing. This is briefly covered in Distributed Systems by Andrew Tanenbaum.[1]

Distributed systems are mainly classified in two categories:
1) Computing
2) Cluster Computing

The history of computer clusters is best captured by a footnote in Greg Pfister's In Search of Clusters: Virtually every press release from DEC mentioning clusters says DEC, who invented clusters.... IBM did not invent them either. Customers invented clusters, as soon as they could not fit all their work on one computer, or needed a backup. The date of the first is unknown, but it would be surprising if it was not in the 1960s, or even late 1950s.[8]

The formal engineering basis of cluster computing as a means of doing parallel work of any sort was arguably invented by Gene Amdahl of IBM, who in 1967 published what has come to be regarded as the seminal paper on parallel processing: Amdahl's Law. Amdahl's Law describes mathematically the speedup one can expect from parallelizing any given otherwise serially performed task on a parallel architecture. This article defined the engineering basis for both multiprocessor computing and cluster computing, where the primary differentiator is whether or not the interprocessor communications are supported "inside" the computer (on for example a customized internal communications bus or network) or "outside" the computer on a commodity network.

Consequently the history of early computer clusters is more or less directly tied into the history of early networks, as one of the primary motivations for the development of a network was to link computing resources, creating a de facto computer cluster. Packet switching networks were conceptually invented by the RAND corporation in 1962. Using the concept of a packet switched network, the ARPANET project succeeded in creating in 1969 what was arguably the world's first commodity-network based computer cluster by linking four different computer centers (each of which was something of a "cluster" in its own right, but probably not a commodity cluster). The ARPANET project grew into the Internetwhich can be thought of as "the mother of all computer clusters" (as the union of nearly all of the compute resources, including clusters, that happen to be connected). It also established the paradigm in use by all computer clusters in the world todaythe use of packet-switched networks to perform interprocessor communications between processor (sets) located in otherwise disconnected frames.

The development of customer-built and research clusters proceeded hand in hand with that of both networks and the Unix operating system from the early 1970s, as both TCP/IP and the Xerox PARC project created and formalized protocols for network-based communications. The Hydra operating system was built for a cluster of DEC PDP-11 minicomputers called C.mmp at Carnegie Mellon University in 1971. However, it was not until circa 1983 that the protocols and tools for easily doing remote job distribution and file sharing were defined (largely within the context of BSD Unix, as implemented by Sun Microsystems) and hence became generally available commercially, along with a shared filesystem.

The first commercial clustering product was ARCnet, developed by Datapoint in 1977. ARCnet was not a commercial success and clustering per se did not really take off until Digital Equipment Corporation released their VAXcluster product in 1984 for the VAX/VMS operating system. The ARCnet and VAXcluster products not only supported parallel computing, but also shared file systems and peripheral devices. The idea was to provide the advantages of parallel processing, while maintaining data reliability and uniqueness. VAXcluster, now VMScluster, is still available on OpenVMS systems from HP running on Alpha and Itanium systems.

Two other noteworthy early commercial clusters were the Tandem Himalaya (a circa 1994 high-availability product) and the IBM S/390 Parallel Sysplex (also circa 1994, primarily for business use).

No history of commodity computer clusters would be complete without noting the pivotal role played by the development of Parallel Virtual Machine (PVM) software in 1989. This open source software based on TCP/IP communications enabled the instant creation of a virtual supercomputera high performance compute clustermade out of any TCP/IP connected systems. Free form heterogeneous clusters built on top of this model rapidly achieved total throughput in FLOPS that greatly exceeded that available even with the most expensive "big iron" supercomputers. PVM and the advent of inexpensive networked PCs led, in 1993, to a NASA project to build supercomputers out of commodity clusters. In 1995 the Beowulf clustera cluster built on top of a commodity network for the specific purpose of "being a supercomputer" capable of performing tightly coupled parallel HPC computationswas invented,[9] which spurred the independent development of grid computing as a named entity, although Grid-style clustering had been around at least as long as the Unix operating system and the Arpanet, whether or not it, or the clusters that used it, were named.

From these two categories cluster computing enables parallel computing on small personal networks.On Linux systems, cluster was first created by Thomas Sterling et al. Using off-the-shelf hardware named Beowulf Cluster.[2]

The simple idea was to create massive parallel computing power using cheap hardware. The created cluster fulfilled the intended purpose. This cluster system was named Beowulf, which was a massive hit in developer community. Beowulf cluster was a system out of existing open source software tools. So it used well known utilities to suffice basic functionalities of cluster. It uses ssh for communication between nodes and utilities like rexec for remote execution.[2]

As the Beowulf cluster is Do-It-Yourself (DIY) system, the willing user has to install and configure all the utilities to 10lavou the basic functionalities. All though it seems straight forward task but the difficulty varies depending on installed 10lavour of Linux operating system.[3].To enable the communication between instances of a single process or two different processes executing on same or different nodes of cluster Message Passing Interface (MPI) is used. Beowulf uses OpenMPI [4], the open source counterpart of MPI. The official documentation of OpenMPI [8] maintains the details about architecture of OpenMPI project.

Beowulf cluster uses orterun to initialize all basic process structures on intended nodes. This orterun command-line utility is provided by the ORTE [5] layer of OpenMPI project. Orterun utility is efficient for applications developed using OpenMPI libraries.There has to be specific mechanism for applications to utilize the available resources on cluster efficiently. This specific mechanism is called load balancing. Beowulf uses the ORTE layer [5] of OpenMPI project for load balancing. But this load balancing technique is effective for applications developed using OpenMPI libraries.

Chapter 3

Concepts and Specifications

3.1 Project Statement

Improved cluster manager for Linux based executable and linkable format files over Beowulf cluster. As stated in the problem statement this project is going to develop an improvised cluster manager which is going to use the basic functionalities like load balancing, IPC(Interprocess communication) etc. Provided by Beowulf cluster. Hence it can be seen that it is only going to support Linux based executable and linkable formats files (ELF).

3.1.1 Beowulf Cluster

It is a computer cluster of what are normally identical, commodity-grade computers networked into a small local area network with libraries and programs installed which allow processing to be shared among them. The result is a high-performance parallel computing cluster from inexpensive personal computer hardware [3].

3.1.2 Load Balancing

It is a computer networking method for distributing workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any one of the resources [6].

3.1.3 IPC

Interprocess Communication, it is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated [8]. Consider an example if a person want to execute an application ex.3D MAYA-MAX and he has problem that he havent get proper working output. But, his friend has a PC of high end and he is thinking that can I use his processing power to run my application?

This can be solved using a cluster, but cluster is not easy to setup. People have to provide environment variables and hence they are going to be uncomfortable to use the cluster.
Proposed system works as shown in figure

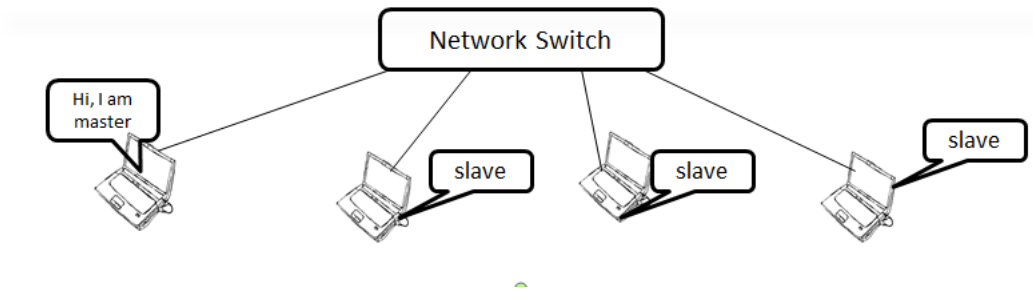

Figure 3.1: *Basic Cluster.*

Functionalities of the proposed system are:

- Personal network: System is going to use switch/router for building personal network.
- Communication: For communication purpose system is going to use TCP/IP protocol (current system uses ssh).
- Storage: The memory which is required by cluster is shared common secondary storage.

- Execution: For the execution of application the concept called Open-MPI (orterun) is used.
- Management: For management purpose OpenMPI (orte) is used.

Features of proposed system are:

- The system will have better execution environment for ELF executable with ORTE/OMPI
- System will provide improved and efficient cluster manager at kernel level

3.2 Technology Used

- Programming Language: C, Java
- OpenMPI C Library
- Beowulf Cluster

3.3 OS Requirements

Linux

Beowulf cluster project was built-up using Linux operating system. Linux is open source and so are projects required to run Beowulf cluster. The proposed system too is set to be open sourced and will be open for extension.

3.4 Hardware Requirements

Processor : Pentium IV or Advanced

Linux supports variety of architectures, but, stable OpenMPI version supports x86 architecture. Also all the kernel module code set to be written for proposed system will be written for x86 architecture because of availability of hardware and software tools for development.

RAM : 256 MB (min)

256 MB is the minimum physical memory requirement for Linux operating system.

HDD : 8 GB

8 GB is the minimum requirement for Linux operating system. HD Beowulf is set to use 256 MB to 512 MB of hard disk storage.

Chapter 4

Design and Specifications

4.1 Existing Architecture

Current Beowulf architecture can be depicted by following figure,

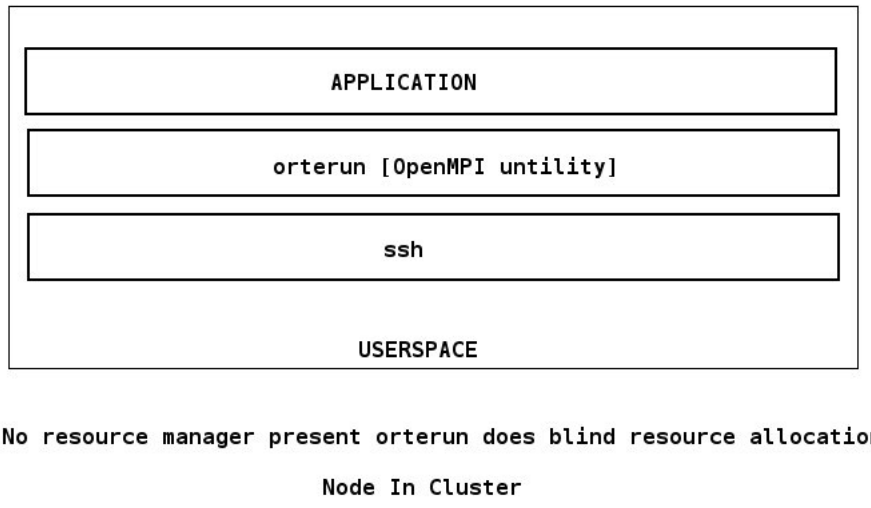

Figure 4.1: *Existing Beowulf Architecture.*

The existing architecture consists the following components:

1. Application

The user application which a particular user wants to execute on the cluster.

This application can be any application which runs on Linux platform.

2. orterun [OpenMPI Utitlity]

For interprocess communication Message Passing Interface can be used. In Beowulf OpenMPI, viz, an is a Message Passing Interface (MPI) library project combining technologies and resources from several other projects (FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI). It is used by many TOP500 supercomputers including Roadrunner, which was the worlds fastest supercomputer from June 2008 to November 2009, and K computer, the fastest supercomputer from June 2011 to June 2012.

The architecture of OpenMPI can be understood with following figure,

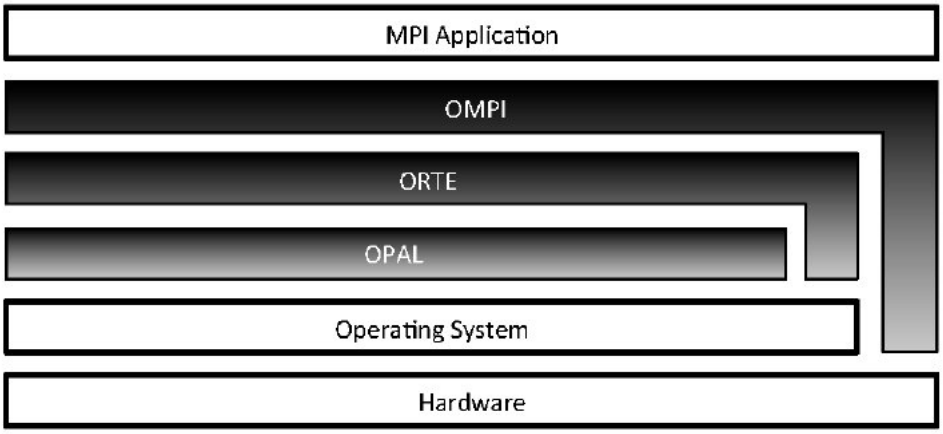

Figure 4.2: *OpenMPI Architecture.*

4.2 Proposed Architecture

For the Cluster Management Client server architecture is followed. In the Proposed Architecture, It can be seen that the management of the cluster is divided into two parts, one in the Application level another in the Kernel level.

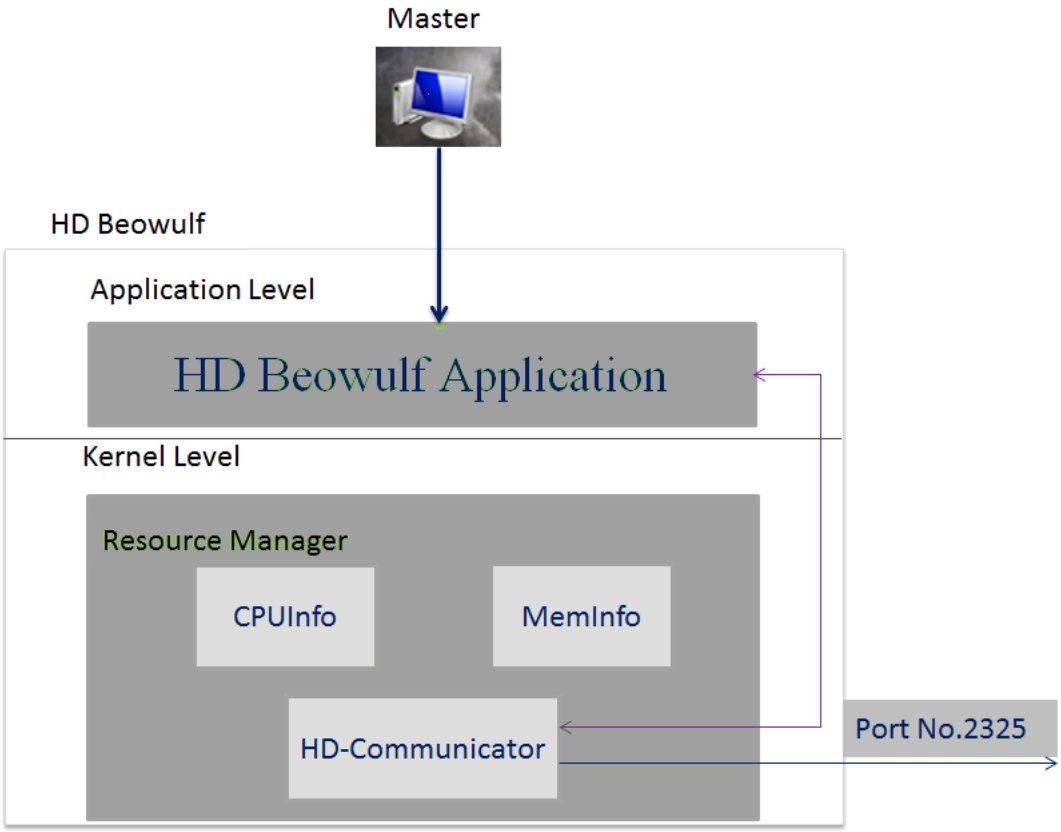

Figure 4.3: *Proposed Architecture.*

1. **Application Level**

The Interaction of user with the system is provided with HD Beowulf Application which has simple and user friendly GUI.

- HD Beowulf Application-The user interacts with the application and gives the input. Application transfers the information from user to the underlying kernel modules. The communication between the application and kernel is with the help of Sockets. It gets results back from resource Manager and shows to the user.

2. **Kernel Level**

Here there resides three modules which communicate with each other for efficient resource management.

- cpuinfo- It provides information regarding the CPU utilization of the system.
- meminfo- It provides information regarding RAM usage by the system.
- hdcommunicator-It provides communication between the Master and Slave systems. The HD-Beowulf Application interact with the HD-Communicator then it ping to the available nodes in the network chosen by user for their system status. Then CPUInfo and MemInfo are executed by individual node and results are forwarded to the Master. The communication is done with UDP sockets.

4.3 UML Diagrams

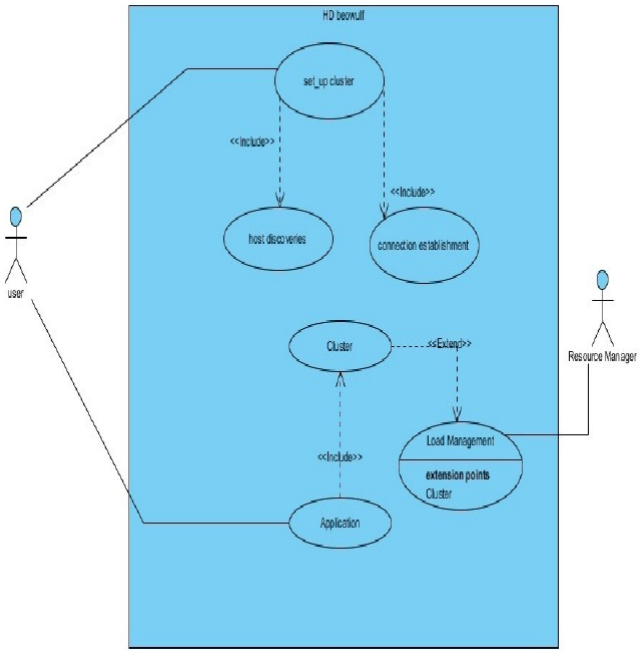

Figure 4.4: *Use Case.*

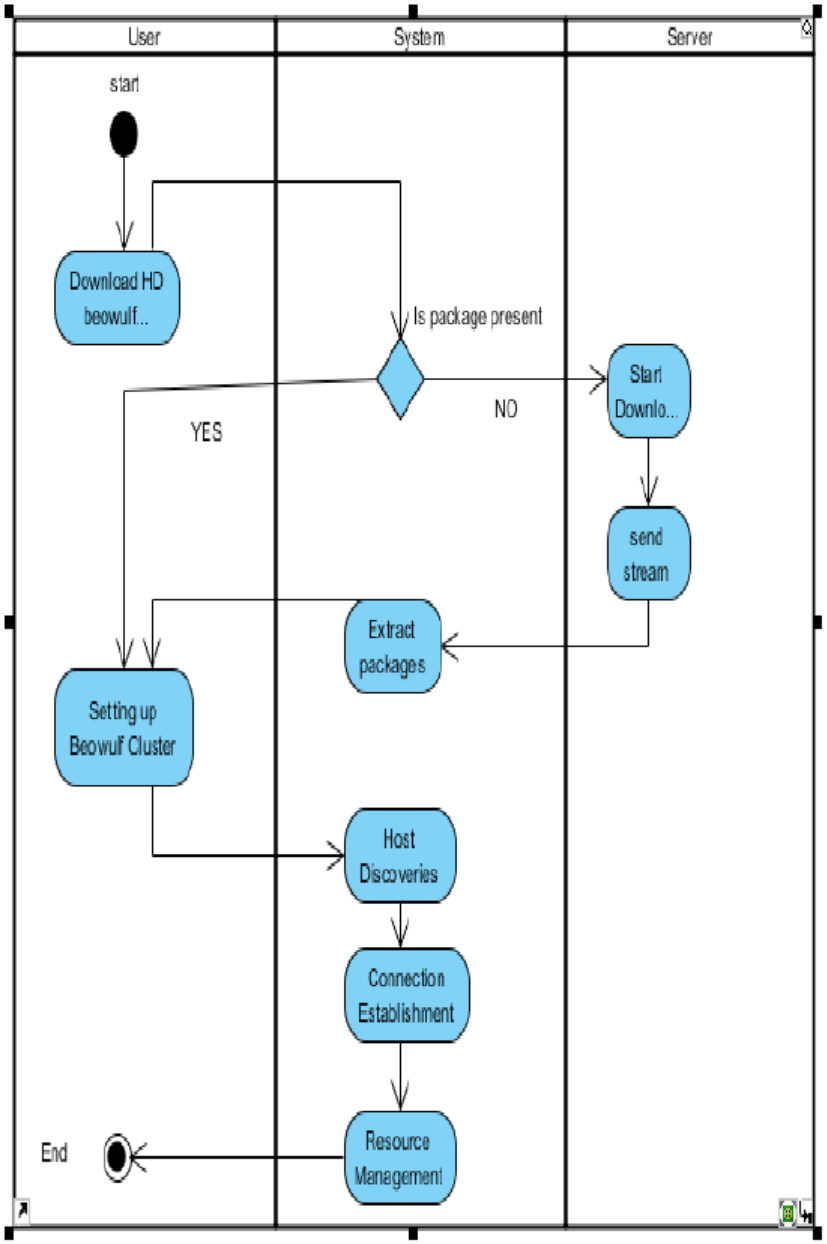

Figure 4.5: *Activity Diagram.*

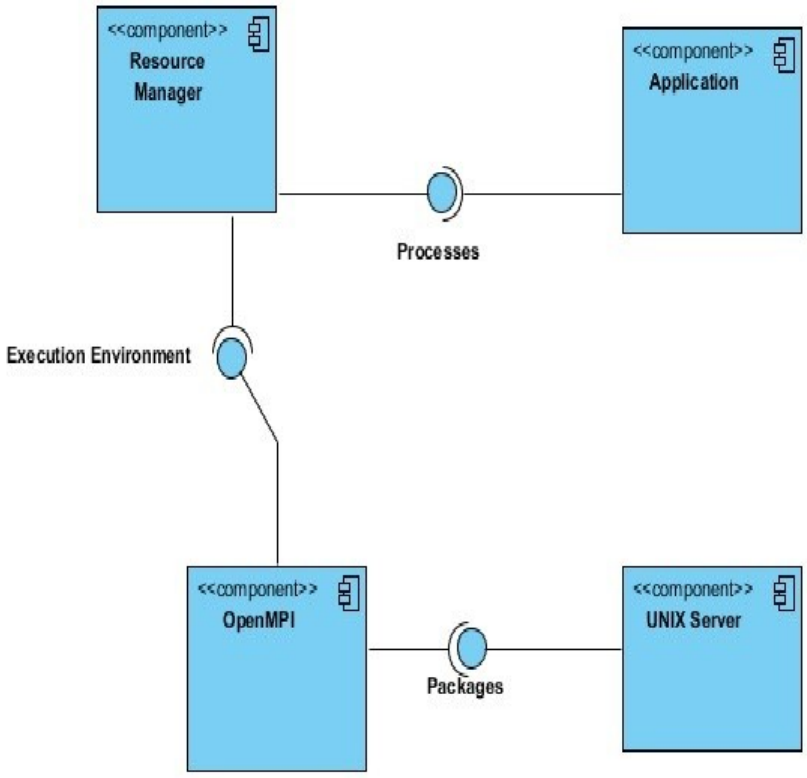

Figure 4.6: *Component Diagram.*

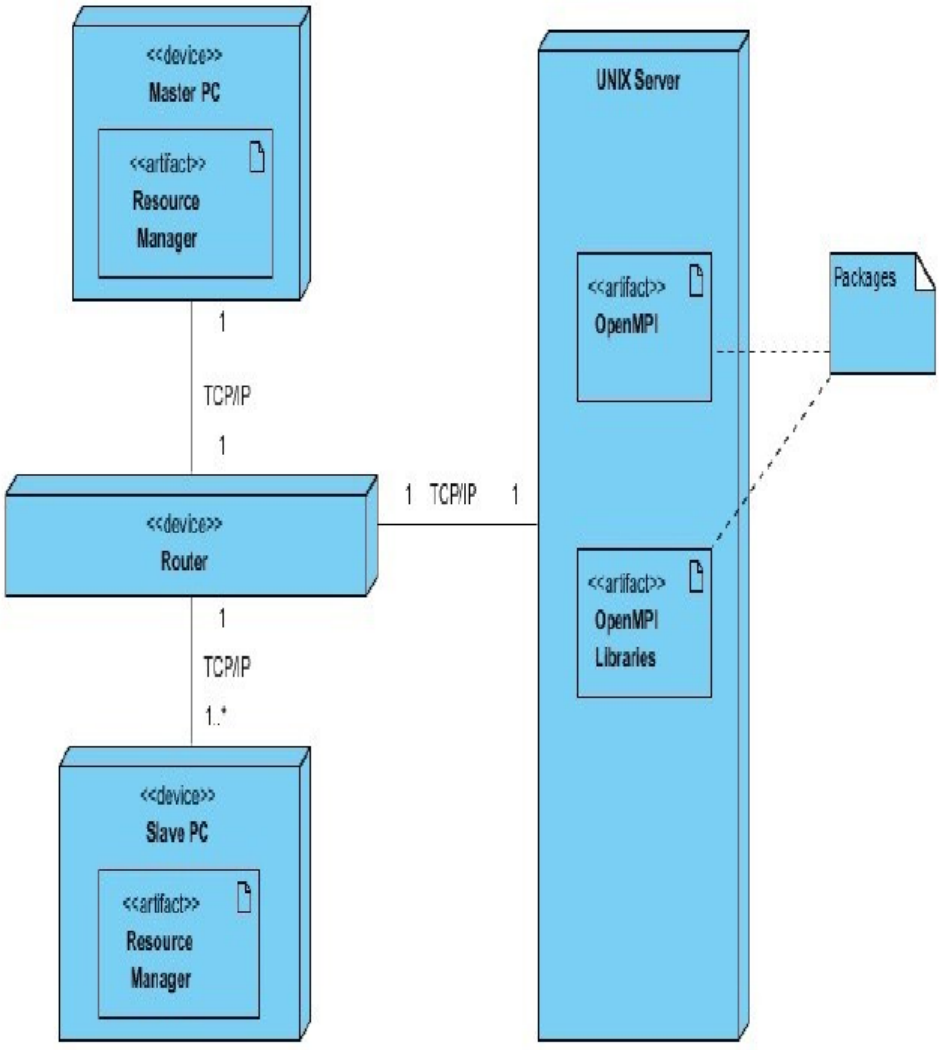

Figure 4.7: *Deployment Diagram.*

Chapter 5

Performance Testing And Evaluation

5.1 Code Snippets

5.1.1 meminfo.c

```
static void meminfo_read_file(char *filename, char buff[2][8])
{
        int i,flag=0;
        char temp,buf[8],buf1[8];
        int a=0;
        struct file * fd;
        mm_segment_t old_fs = get_fs();
        set_fs(KERNEL_DS);
        fd = file_open(filename, O_RDONLY, 0);

        if (fd >= 0)
        {
                while (file_read(fd,a, &temp, 1) == 1)
                {
                        a++;
                        if(temp >= 48 && temp <= 57)
                        {
                                buf1[0]=(int)temp;
                                for(i=1;i<7;i++)
                                {
                                        file_read(fd,a,&temp,1);
                                        a++;
```

```
                        if(temp >= 48 && temp <= 57)
                        {
                                buf1[i]=(int)temp;
                        }
                        else
                                break;
                }

                buf1[i]=  \0   ;
        }

        if(temp == 70)
        {
                while(file_read(fd,a,&temp,1))
                {
                        a++;
                        if(temp >= 48 && temp <= 57)
                        {
                                buf[0] = (int)temp;
                                for(i=1;i<7;i++)
                                {
                                        file_read(fd,a,&temp,1);
                                        a++;
                                        if(temp >= 48 &&
                                            temp <= 57)
                                        {
                                                buf[i]=(int)temp;
                                        }
                                        else
                                                break;
                                }
                                buf[i]=  \0   ;
                                flag=1;
                                file_close(fd);
                                break;
                        }
                }
        }
        if(flag == 1)
        break;
        }
}
```

```
                set_fs(old_fs);
                strcpy(buff[0],buf1);
        }
        strcpy(buff[1], buf);
}
```

5.1.2 cpuinfo.c

To calculate CPU usage on a host.

```
int cpuinfo_read_file(char *filename)
{
        int i,j,l,k,s,sum=0;
        char temp,temp1[10];
        int a[4],b[4],loadavg;
        struct file *fd;
        mm_segment_t old_fs = get_fs();
        set_fs(KERNEL_DS);
        fd = file_open(filename, O_RDONLY, 0);
        if (fd >= 0){
                i=k=l=s=j=sum=0;
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                while(i<4){
                        while(file_read(fd,s++,&temp,1){
                                if(temp!=     )
                                        temp1[j++]=temp;
                                else
                                        break;

                                temp1[j]= \0   ;
                                while(l!=j){
                                        sum=sum*10+(temp1[l]-48);
                                        l++;
                                }

                                a[k++]=sum;
                                sum=0;
                                l=j=0;
```

```
                        i++;
                }
        }

        file_close(fd);
        msleep(1000);
        fd = file_open(filename, O_RDONLY, 0);
        if(fd >= 0){
                i=k=s=sum=0;
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                file_read(fd,s++,&temp, 1);
                while(i<4){
                        while(file_read(fd,s++,&temp,1)){
                                if(temp!=     )
                                        temp1[j++]=temp;
                                else
                                        break;
                        }
                        temp1[j]= \0   ;
                        while(l!=j){
                                sum=sum*10+(temp1[l]-48);
                                l++;
                        }
                        b[k++]=sum;
                        sum=0;
                        l=j=0;
                        i++;
                }

        if(((b[0]+b[1]+b[2]+b[3]) - (a[0]+a[1]+a[2]+a[3]))!=0)
        loadavg = (((b[0]+b[1]+b[2]) - (a[0]+a[1]+a[2])) *100)/
            ((b[0]+b[1]+b[2]+b[3]) - (a[
        else
                loadavg=0;

        set_fs(old_fs);
        file_close(fd);
}
```

5.2 Testing

5.2.1 Unit Testing

Unit testing is testing the smallest testable part of a application. This is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use. Unit can be entire module or a function or procedure.

We tested the following modules:

Module 1:

Aim: To show IPs available in the network
Test Input: IP address of the master node
Expected Result: Shows all the available nodes in the network
Actual Result: Same as expected

Module 2:

Aim: To receive correct CPU and Memory information from slave nodes
Test Input: IP addresses of slave node
Expected Result: Correct CPU and Memory utilization information by slave nodes returned to master node
Actual Result: Same as expected

Module 3:

Aim: To distribute processes efficiently according the information received from slave nodes
Test Input: CPU and Memory information of slave nodes
Expected Result: Process should be dispatched efficiently on the basis of biggest hole first principle.
Actual Result: Same as expected

Module 4:

Aim: To show all the networks along with their IPs correctly on GUI
Test Input: IPs of the all available networks
Expected Output: IPs should be displayed correctly on the GUI
Actual Result: Same as expected

Module 5:

Aim: To show IP, Meminfo and CPUInfo of available PCs in selected net-

work on GUI
Test Input: IP, Meminfo and CPUInfo of all PCs in network
Expected Output: IP, Meminfo and CPUInfo of all PCs in network should be displayed correctly on GUI
Actual Output: Same as expected

5.2.2 Integration Testing

Integration testing corresponds to a phase in software testing in which individual software modules (which are unit tested) are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing can expose problems with the interfaces among program components before trouble occurs in real world program execution. Integration testing is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision. There are two major ways of carrying out an integration test, called the bottom up method and the top-down method. We have used bottom up test methodology.

In our system implementation we have applied bottom up integration strategy.

The order of integration steps for testing was as follows:
1. Building and testing of cpuinfo.c and meminfo.c
2. Integrating these modules in Communication module
3. Testing of communication module
4. Testing of resource manager

Test Results:
The system passed all the tests while integration.

5.2.3 Performance Evaluation

Following table compares performance of existing and built system

Table 5.1: Performace Analysis

Existing System	Built System
Cluster setup from terminal.	User friendly GUI.
Difficult to setup.	Easy to setup.
Selection of slave node without checking available resources at the slave.	Selection of slave node depending on the resources available at the slave.
IP is selected by system	IP is selected by User
Execution slower than built system	Execution is faster than existing system
No resource manager at all	Resource manager at kernel level for faster execution

Chapter 6

Results And Analysis

As stated earlier the aim of this project is providing simple configuration of Beowulf cluster. This is achieved through HDBeowulf Application Wizard. The user can setup a cluster using simple wizard.

Step 1. Selection of network

At first the available networks are displayed to which the host is connected. User is allowed to choose from the networks to create a new cluster.

Step 2. Selection of hosts

After selecting a certain network the user is guided to select the number of hosts he wants to select to create a new cluster.

The steps involved in the wizard are showed in following screenshots.

Figure 6.1: *Wizard: Network Selection.*

Figure 6.2: *Wizard: Host Selection.*

Chapter 7

Applications

7.1 At Industry Level

- Customized distributed system architecture (Supercomputer)
- Video encoding and decoding
- Simulation software execution/development
- Scientific and medical applications requiring high processing power(protein sequencing extraction, Flight path recognition)

7.2 User Level

- To use low end PCs for high processing applications
- For execution of programs with high resource requirements (like games, blue ray videos, 3D Maya)

Chapter 8

Conclusion And Future Scope

8.1 Future Scope

- Dynamically dispatching the process
- Replacement of socket by proc file system for user level to kernel level communication
- Execution of general programs supporting parallelism without customization for orterun

8.2 Conclusion

Thus, efficient resource management should be done while building a cluster. Hence we have built a resource manager which provides efficient resource management improving system performance and reducing execution time by placing resource manager in kernel.

Chapter 9

References

1 Distributed systems principles and paradigms, second edition by Andrew S. Tanenbaum and Maarten Van Steen.

2 How to Build a Hypercomputer, by Thomas Sterling, American Scientia, July 2001 issue.

3 BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION, Donald J. Becker Thomas Sterling et al., 1998, NASA

4 Open MPI: Goals, Concept and Design of a Next Generation MPI Implementation, Edgar Gabriel, IEEE transactions on Distributed Computing, 2004

5 The OpenRTE: A transperent Multi-Cluster Environment for High-Performance Computing, R.H. Castain et al, IEEE transactions on Distributed Computing , 2005.

6 Effective load balancing for Beowulf Clusters, Tetiana D., IEEE transactions on Distributed Computing , 2008

7 MPIRUN Manual, http://docs.oracle.com/mpirun

8 Pfister, Gregory (1998). In Search of Clusters (2nd ed.). Upper Saddle River, NJ: Prentice Hall PTR. p. 36. ISBN 0-13-899709-8.

9 http://www.beowulf.org/overview/history.html