

Performance Analysis of MPI (mpi4py) on Diskless Cluster Environment in Ubuntu

Sayma Sultana
Chowdhury

Department of Computer
Science and Engineering
Shahjalal University of Science
and Technology
Sylhet, Bangladesh.

Marium-E-Jannat

Department of Computer
Science and Engineering
Shahjalal University of Science
and Technology
Sylhet, Bangladesh.

Abu Awal Md. Shoeb

Department of Computer
Science and Engineering
Shahjalal University of Science
and Technology
Sylhet, Bangladesh.

ABSTRACT

Now-a-days Cluster computing has become a crying need for the processing of large scale data. For computing large amount of data, which need huge execution time, the run time can be reduced using multiple processors and task distribution through cluster computing. It is the technique of sharing two or more computers' resources through a network (usually through a local area network) in order to take advantage of the parallel processing power of those computers. Clusters of computers are usually deployed to improve processing speed and/or reliability and scalability over that provided by a single computer. In this paper we proposed a High Performance computing approach on Linux platform (Ubuntu) using Parallel Programming environment with the collaboration of multiple nodes for large scale computational work.

General Terms

Performance analysis, Comparative Study.

Keywords

MPI, Cluster, Diskless Cluster, Ubuntu, mpi4py performance analysis etc.

1 INTRODUCTION

Cluster computing is very useful not only in the large scientific and engineering projects but also in various business and commercial use. A large number of research organization and communities using cluster computing environment to carry out their sheared data resource out come. Clustered computer is a set of computers dedicated to a network designed to capture their cumulative processing power for running parallel-processing applications.

In cluster system there is a server node and one or more client nodes. Clustered computers are specifically designed to take large programs and sets of data and subdivide them into component parts, thereby allowing the individual nodes of the cluster to process their own individual chunks of the program and finally collecting the result as a whole. However the use of the processing power of clustered computers remains to be as a challenge for the users due to the complexity involved in the creation of the application and deployment of distributed computational resources.

Here we present a computational environment, where we can use any number of processing resource extensively and thus can improve the performance. Since we are talking about the speed and scalability, we can use a diskless clustered environment with the collaboration of message passing interface (MPI) through which we can add as many diskless

node (a computer with no hard disk) as we can and maintain them centrally through a root node for getting more processing speed. Hence we can improve processing speed as well as scalability and have the parallel programming scope with the use of MPI.

This paper is organized as follows: section 2 discusses the related works and section 3 shortly discusses Diskless Cluster architecture, section 4 describes the problem scope of programs that require huge computational power, section 5 discusses the proposed solution, section 6 illustrates the experimental evaluation and performance analysis with some examples. We conclude in section 7.

2 RELATED WORKS

In our short survey we found several projects that are related to the work described in this paper. The Kerrighed^[14] Cluster, Mosix^[13] cluster, Beowulf^[14] cluster, Clustermatic^[15] efforts, Warewulf^[20] project and so on.

In the online documentation of Kerrighed cluster they showed the way to build a diskless environment in Linux. But it does not include any way to parallel processing.

On the other hand in the online documentation of Warewulf project they showed how to best build a disk-less cluster. But it also does not include any documentation or experiment done with parallel programming.

The Clustermatic efforts, at Los Alamos National Labs (LANL) showed a way of minimizing the system environment that runs on the compute nodes. It requires kernel modifications and uses a static ram-disk for shared libraries.

Beowulf is a programming model for parallel computation. It needs distributed application programming environments such as PVM (Parallel Virtual Machine) or MPI (Message Passing Interface).

MOSIX is a software package that was specifically designed to enhance the Linux kernel with cluster computing capabilities. It is a kernel implementation of process migration. It runs best when running plenty of separate CPU intensive tasks. But its big drawback is shared memory, like Beowulf. For applications running they use shared memory. So there will not be any benefit from Mosix and Beowulf because all processes accessing said shared memory must resided on the same node.

Clusters are used where tremendous raw processing power is. Clusters use parallel computing to reduce the time required for a CPU intensive job. The workload is distributed among the nodes that make up the clusters and the instructions are executed in parallel. More nodes mean faster execution and less time taken. But no one projects that we found during our survey shows an experimental results of MPI implementation on a diskless cluster. They didn't show how to combine a parallel processing implementation to a diskless cluster server to gain processing in parallel and what is the result or performance of those. In our work we tried to give an overview of how a high performance diskless cluster system works with an MPI implementation on Linux (Ubuntu) by showing the comparisons of execution times with different numbers of processes on different number of nodes.

3 DISKLESS CLUSTER OVERVIEW

The architecture of the cluster computing system is shown in Figure 1. A cluster consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource. A computer node can be a single or multiprocessor system PCs, workstations, I/O facilities, and an operating system. A cluster generally refers to two or more computers nodes connected together. The nodes can exist in a single cabinet or be physically separated and connected via a LAN. An interconnected LAN-based cluster of computers can appear as a single system to users and applications.

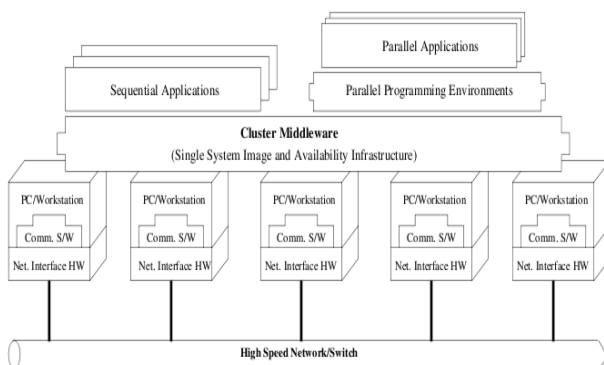


Fig 1: Cluster computing architecture

Such a system can provide a cost effective way to gain features and benefits fast and reliable services that have historically been found only on more expensive proprietary shared memory systems.

In Diskless cluster system client systems do not need a local file system or any storage device to boot and run. This system consists of a cluster server and one or more cluster clients all attached to a network. Each computer in the cluster is referred to as a cluster node or cluster member. The client nodes boot and load their operating system from the cluster server and obtain their root file systems from their server. Figure 2 shows a diskless cluster system scenario.

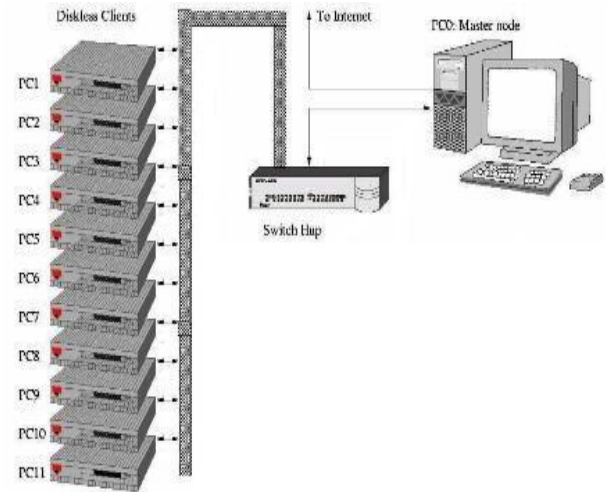


Fig 2: Diskless cluster system

The simplest way to achieve this is to use a NFS server with the collaboration of PXE, DHCP, TFTP and SSH server configured to host the generic boot image for the single system image cluster nodes.

NFS (Network File Sharing) is a distributed file system protocol allowing a user on a client computer to access files over a network in a manner similar to how local storage is accessed. Since it is completely diskless boot, the client computer must have a network-bootable (PXE) network card. Wireless will not work, with exception of a wireless-to-Ethernet external bridge. Many motherboards have a PXE-compatible Ethernet card built in, but you will need to enable support in the BIOS.

PXE (Preboot Execution Environment) allows computers to boot up remotely through a network interface. PXE enables a client machine to boot from a server independent of the hard disks and installed operating system.

TFTP (Trivial File Transfer Protocol) is intended to be used when bootstrapping diskless systems. The virtual machine running Linux listens on TFTP and NFS servers to provide a bootable kernel image and a mounted root file system respectively for the diskless client. The development time is reduced as we need not reinstall the kernel and/or root file system every time on the target. We develop device drivers and kernel code on the host and simply reboot the target to load and test. To allow multiple clients to bootstrap at the same time, a TFTP server needs to provide some form of concurrency. Because UDP does not provide a unique connection between a client and server (as does TCP), the TFTP server provides concurrency by creating a new UDP port for each client. This allows different client input datagram to be demultiplexed by the server's UDP module, based on destination port numbers, instead of doing this in the server itself.

DHCP (Dynamic Host Configuration Protocol) is a network configuration protocol for hosts on Internet Protocol (IP) networks. Computers that are connected to IP networks must be configured before they can communicate with other hosts. The most essential information needed is an IP address, and a default route and routing prefix. DHCP eliminates the manual task by a network administrator. It also provides a central

database of devices that are connected to the network and eliminates duplicate resource assignments.

SSH (Secure Shell) is a protocol for conducting a secure session over a non-secure network. It is a network protocol for secure data communication, remote shell services or command execution and other secure network services between two networked computers that it connects via a secure channel over an insecure network: a server and a client (running SSH server and SSH client programs, respectively).

Diskless client systems can still have disks and other peripherals directly attached to them.

4 PROBLEM SCOPE OF PROGRAMS THAT REQUIRE HUGE COMPUTATIONAL POWER

Now a day, for research purpose and providing upgraded technology we need high computing performance. For various scientific calculations and bioinformatics research, we need high performance computing system.

Bioinformatics is a great upcoming research area which needs a huge processing of data. In this field of computation researchers take a very large sequence of data like gene sequence or other features of any species or organ as input. Then process them and have the desired output, which takes a long time to execute and give the total result.

On the other hand in case of security issue prime numbers are used in a large scale. But producing a random prime number of a large length is very time consuming through a single computer.

For searching a data in a huge database, it takes huge time for executing a single query while traversing through the whole large database.

For various scientific research issues, huge calculation is needed. These types of calculation needs large scale data processing which leads to the need for large amount of processing power.

The above computation programs don't need huge storage capacity. It only needs a large and fast processing of data. A Diskless cluster computing environment can give one the facility of having the above types of computation. The advantages of this type of system are: ease of administration, efficient sharing of resources, power saving and so on. Physically-unsecured systems contain no data after they are powered off. Diskless operation ensures this element of security. Diskless computer's booting from the same file system makes homogenous computing environment which is suitable for cooperative computation. It has centralized storage and control. It is easy to install, update and reconfigure. Diskless nodes are typically ordinary personal computers or workstations with no hard drives supplied, which means the usual large variety of peripherals can be added. For distributed load and peripheral support this system is very comfortable.

5 PROPOSED SOLUTION

Among many parallel computational models, message-passing has proven to be an effective one. This approach is used in today's most demanding scientific and engineering application related to modeling, simulation, design, and signal

processing, bioinformatics and many more scientific research works. Message Passing Interface is a standardized and portable message-passing system designed to function on a wide variety of parallel computers. The standard defines the syntax and semantics of library routines and allows users to write portable programs in various programming languages like C, C++, Perl, Python etc.

MPI provides a simple-to-use portable interface for the user. It is a communication protocol used to program parallel computations on different computers. It is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation. MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today.

So we can use MPI implementation for distributed parallel programming on two or more diskless nodes. We can use it for dividing the computational task or data or program into different chunks called process and collecting the results from those processes and finally having the total desired result. MPI does the inter communication among various processes and combine the outcome from different processes running on different nodes.

So the combination of Diskless cluster computer and MPI gives us the opportunity to have the high performance computing environment. Diskless cluster system provides two or more processors with centralized file system booted from the root or cluster server to have large processing power and MPI gives the facility to have the parallel programming capacity to use the processing power of those attached nodes by dividing the task into different processors. Diskless cluster server allows us to add as many nodes as we wish since the boot file system is centralized on the server and nodes are being booted from that file system. Hence we can have the high performance computing system with speed-up and scalability option.

6 EXPERIMENTAL RESULT AND PERFORMANCE ANALYSIS

An experiment environment has been setup with 16 nodes. The configuration of the server computer is given in Figure 3 and the configuration of connected nodes is given in Figure 4.

Three different application programs written in python have been used to evaluate the performance comparison. In our implementation, all the applications take an integer as input and give a number of integers as output. At first, we developed each program with normal syntax of python. Then we apply MPI with every program. We use **mpi4py-1.3** with the collaboration of **openmpi-1.4.3**(a MPI2 implementation) to make a normal **python program MPI enabled**.

In all applications, same input is used to compare between them. We will show the performance comparisons in the context of a multiple process and also in the context of multiple nodes. These cases will give us an overview of how diskless cluster with parallel processing approach effects in case of a process and in case of nodes. Here we have shown different performance comparisons where our proposed approach gives fruitful performance in almost every case.



Processor: Intel(R) Core(TM) i3-2100 CPU 3.10GHz
Ram: 4 GB DDR2
OS: Ubuntu10.10

Fig 3: Server computer



Processor: Intel(R) Core(TM)2 Duo CPU 1.18GHz
Ram: 2 GB DDR2
OS: none

Figure 4: Connected node

6.1 In the context of Multiple Process

This section gives an overview how the MPI improves performance by decomposing a task into several small segments. Here we give the comparisons between the performance of a normal application program with no MPI enabled and the modified MPI enabled implementation with various numbers of processes.

Message Passing Interface (MPI) divides the computational task into several chunks called process so that the total computation can be divided into several segments and each process can be computed independently and at last MPI collects the results from different processes and give the outcome as a whole.

6.1.1 Test case 1

In this test module we used our first program. We showed how much time our first normal regular syntax program with no MPI enabled takes to finish in the Server computer and

how much time that program with MPI enabled takes to finish in the same Server computer. This section also shows the comparisons between the times taken by the MPI implementation with multiple processes.

The normal syntax MPI disable program takes **20.382060051 seconds** to accomplish the task. The following table shows comparison of execution times for different number of processes.

Table 1. Data Sheet for Test Case 1

No. of Process	Execution time(seconds)
1	21.4745261669
2	17.7691469193
3	11.1424319744
4	8.95005784035
5	8.60596203804
6	8.92707180977
7	8.86440896988
8	9.00355291367
9	9.00034999847
10	9.06372880936
11	8.88456916809
12	8.92707180977
13	9.39892196655
14	9.02412104607
15	9.17201113701
16	8.91704487801
17	9.20024704933
18	9.02293586731
19	9.21808195114
20	8.86900401115
21	9.12131094933
22	9.16933393478
23	9.05712008476
24	9.28680014613
25	9.07506990433
26	9.23475599289
27	9.12313318253
28	9.19380187988
29	9.36317300797
30	9.24970197678

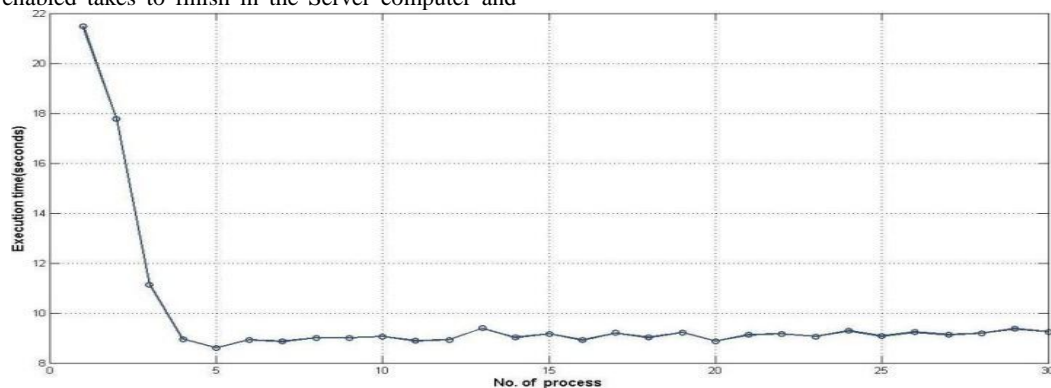


Fig 5: No. of process vs. Execution time (1st program)

6.1.2 Test case 2

In this test module we used our second program and do the same as Test case 1.

The normal syntax MPI disable 2nd program takes **20.3069760799 seconds** to accomplish the task. The following table shows comparison of execution times for different number of processes for 2nd program.

Table 2. Data Sheet for Test Case 2

No. of process	Execution time(seconds)
1	21.4272689819
2	17.8335881233
3	12.0140181065
4	9.85466599464
5	9.24217700958
6	9.07933688164
7	9.02370095253
8	8.87904500961
9	9.21515703201
10	9.01245093346
11	9.46864199638
12	8.88917493802
13	9.63691496849
14	9.07993102074
15	9.06729006767
16	8.93704390526
17	9.34647893906
18	9.06111311913
19	9.00326609612
20	9.31683707237
21	9.16021901925
22	9.06684589386
23	9.14765787125
24	8.86730003357
25	9.02105410099
26	9.10720307827
27	9.07207107544
28	9.22094583511
29	9.15287804604
30	9.17227101326

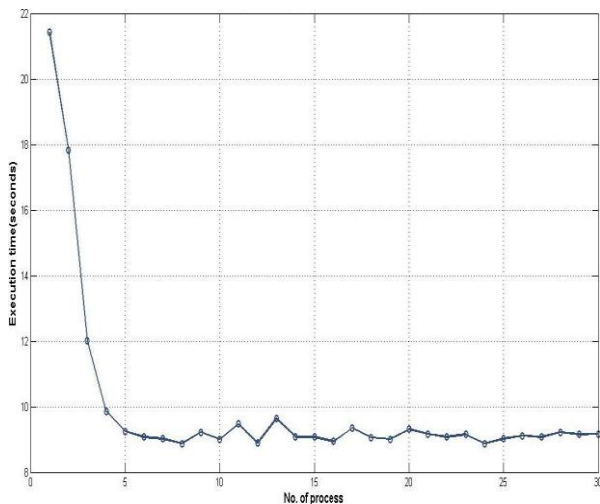


Fig 6: No. of process vs. Execution time (2nd program)

6.1.3 Test case 3

In this test module we used our third program and do the same as above Test case 1.

The normal syntax MPI disable 3rd program takes **141.114938974 seconds** to accomplish the task. The following table shows comparison of execution times for different number of processes for 3rd program.

Table 3. Data Sheet for Test Case 3

No. of process	Execution time(seconds)
1	141.382604122
2	105.773243904
3	92.3824980260
4	80.3803570271
5	78.2872718334
6	75.7585599422
7	72.7426919937
8	72.3323883605
9	72.1017578125
10	71.8621147156
11	72.1569843292
12	71.6311168671
13	73.2939510345
14	74.3767967224
15	72.0735118866
16	72.7691612244
17	74.0880203247
18	73.3069419861
19	71.9958248138
20	72.3030853271
21	71.6330032349
22	74.1029167175
23	75.0380022049
24	73.9989967346
25	71.8399429321
26	71.5238189697
27	72.7839641571
28	72.6795129776
29	72.1450138092
30	72.1558971405

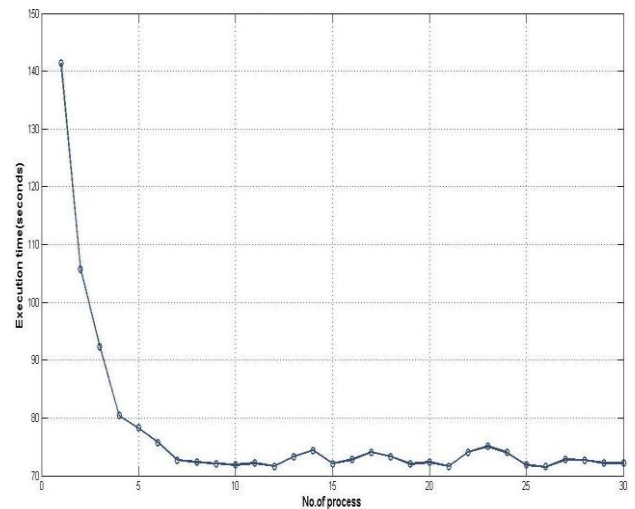


Fig 7: No. of process vs. Execution time (3rd program)

6.2 In the context of Multiple Nodes

This section gives an overview how multiple nodes improve performance with the collaboration of MPI. Here we give the comparisons among the performance of a modified MPI enabled program implementation with various numbers of processes on various numbers of nodes. Here we use number of node=number of process and “Head node” to indicate the server node.

Message Passing Interface (MPI) divides processes into different nodes so that the total computation can be divided into several processors. Here our nodes are different CPU with no storage system attached.

6.2.1 Test case 1

In this test module we used our first program. The following table shows comparison of execution times for different numbers of processes being processed on different processors.

Table 2. Data Sheet for Test Case 1

Machine	No. of process	Execution time (seconds)
Only 1 node	1	24.5755219001
Head node + 1node	1	21.7996483687
Head node + 2nodes	2	12.1023389700
Head node + 3nodes	3	8.95005784035
Head node + 4nodes	4	6.60596203804
Head node + 5nodes	5	5.92770770718
Head node + 6nodes	6	4.86486984089
Head node + 7nodes	7	3.84765208439
Head node + 8nodes	8	3.00403499987
Head node + 9nodes	9	2.86348747653
Head node + 10nodes	10	2.08050900801
Head node + 11nodes	11	1.99892969695
Head node + 12nodes	12	1.99758973807
Head node + 13nodes	13	1.90200321720
Head node + 14nodes	14	1.71855797732
Head node + 15nodes	15	1.00023793780

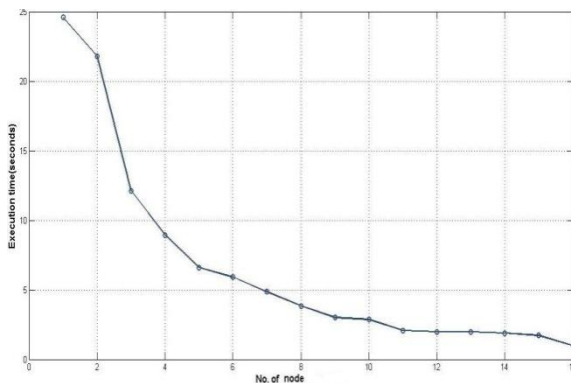


Fig 8:No. of node vs. Execution time (program 1)

6.2.2 Test case 2

In this test module we used our second program and do the same as Test case 1.

Table 2. Data Sheet for Test Case 2

Machine	No. of process	Execution time (seconds)
Only 1 node	1	24.8610630035
Head node + 1node	1	22.5589540005
Head node + 2nodes	2	11.8765909672
Head node + 3nodes	3	8.9438710213
Head node + 4nodes	4	6.2204985619
Head node + 5nodes	5	5.8132009506
Head node + 6nodes	6	4.4078810215
Head node + 7nodes	7	3.7399802208
Head node + 8nodes	8	3.4490003586
Head node + 9nodes	9	2.9340898514
Head node + 10nodes	10	2.8918004036
Head node + 11nodes	11	2.8610630035
Head node + 12nodes	12	1.9589540005
Head node + 13nodes	13	1.8765909672
Head node + 14nodes	14	1.7438710213
Head node + 15nodes	15	1.3204985619

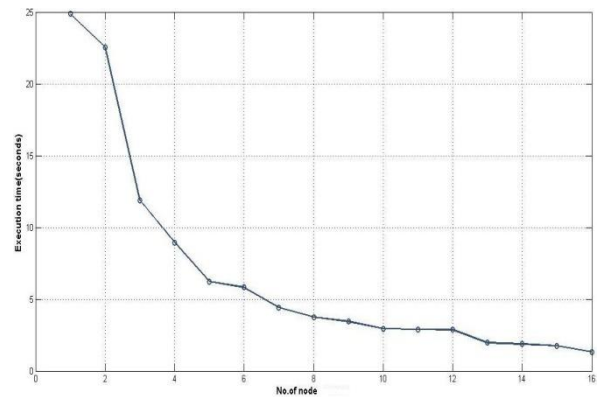


Fig 9:No. of node vs. Execution time (program 2)

6.2.3 Test case 3

This test module uses third program.

Table 2. Data Sheet for Test Case 2

Machine	No. of process	Execution time (seconds)
Only 1 node	1	143.745227499
Head node + 1nodes	1	140.779079794
Head node + 2nodes	2	104.824980260
Head node + 3nodes	3	91.343035496
Head node + 4nodes	4	82.833872714
Head node + 5nodes	5	73.742285599
Head node + 6nodes	6	62.229369197
Head node + 7nodes	7	48.883682305
Head node + 8nodes	8	32.581217575
Head node + 9nodes	9	21.568821471
Head node + 10nodes	10	11.832991692
Head node + 11nodes	11	7.671611161
Head node + 12nodes	12	5.345023951
Head node + 13nodes	13	4.972436796
Head node + 14nodes	14	3.986693511
Head node + 15nodes	15	3.007497292

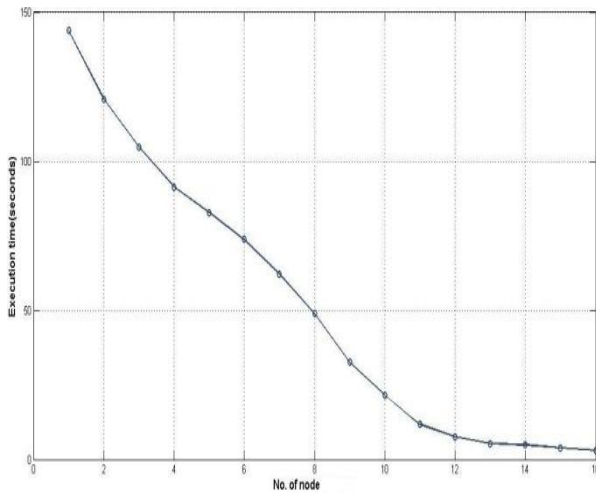


Fig 10:No. of node vs. Execution time (program 3)

7 CONCLUSION

This paper focused on the high performance computing architecture of the Diskless Cluster system. It also focused on the parallel processing on multiple processors/nodes via the master node and ability to have higher performance through MPI. This paper shows the comparison of execution times taken to execute a program between a single computer and multiple nodes attached, through parallel processing. It focused to the easy way to do the parallel computing of large scale data with diskless cluster computing environment.

As we get better performance by using diskless approach with the collaboration of MPI, so we can propose that this mechanism can be used to compute large scale data which needs huge processing power. Thus it can push the Cluster computing one step further.

Our future plan is to modify this high performance system to a high availability cluster system, so that power failure to any of the node or any kind of hardware failure can be recovered and the other nodes can complete the incomplete task of the dead node.

8 REFERENCES

- [1] A Proposal for Creating a Computing Research Repository (CoRR,<http://www.arXiv.org/>) on Cluster Computing
[\[http://www.buyya.com/papers/ClusterRepository.pdf\]](http://www.buyya.com/papers/ClusterRepository.pdf)
Rajkumar Buyya, Monash University, Melbourne, Australia.
- [2] Linux HPC Cluster Installation by International Technical Support Organization
[\[http://mmc.geofisica.unam.mx/femp/HPC/redbook.pdf\]](http://mmc.geofisica.unam.mx/femp/HPC/redbook.pdf)
Counting Polyamines: A Parallel implementation for Cluster Computing Iwan Jensen, Department of Mathematics and Statistics, The University of Melbourne, Victoria3010, Australia
[\[http://www.ms.unimelb.edu.au/~iwan/Publications/2003/LNCS_2659_203.pdf\]](http://www.ms.unimelb.edu.au/~iwan/Publications/2003/LNCS_2659_203.pdf)
- [3] Cluster Algebras, Invariants, Jacobians, and Tropical Curves
Gregg Musiker's Research Statement
[\[http://www.math.umn.edu/~musiker/ResState.pdf\]](http://www.math.umn.edu/~musiker/ResState.pdf)
- [4] Cluster Computing at a Glance
Mark Bakery and Rajkumar Buyyaz
- [5] Division of Computer Science, University of Ports mouth South sea, Hants, UK
School of Computer Science and Software Engineering, Monash University, Melbourne, Australia
- [6] Microwulf Software and Network Configuration Notes, Tim Brom.
[\[http://www.calvin.edu/~adams/research/microwulf/sys/microwulf_notes.pdf\]](http://www.calvin.edu/~adams/research/microwulf/sys/microwulf_notes.pdf)
- [7] CentOS Cluster Server , Ryan Matteson
[\[http://prefetch.net/presentations/CentosClusterServer.pdf\]](http://prefetch.net/presentations/CentosClusterServer.pdf)
- [8] Cluster Computing R&D in Australia, by the Asian Technology Information Program (ATIP)
- [9] [\[http://www.buyya.com/papers/ClusterComputingAU.pdf\]](http://www.buyya.com/papers/ClusterComputingAU.pdf)
- [10] Cluster Computing White Paper, Mark Baker, University of Portsmouth, UK
[\[http://arxiv.org/ftp/cs/papers/0004/0004014.pdf\]](http://arxiv.org/ftp/cs/papers/0004/0004014.pdf)
- [11] MPI/Python | James Noble
- [12] A STUDY IN CLUSTERING, Prof. Nathan Linial
[\[http://www.cs.huji.ac.il/~nati/PAPERS/THESIS/ori.pdf\]](http://www.cs.huji.ac.il/~nati/PAPERS/THESIS/ori.pdf)
- [13] http://www.kerrighed.org/wiki/index.php/Main_Page
- [14] <http://www.mosix.cs.huji.ac.il/>
- [15] <http://www.beowulf.org/>
- [16] www.linux-mag.com/id/1916/
- [17] <http://www.ks.uiuc.edu/Training/Workshop/Cluster/files/clustermatic.html>
- [18] Red Hat Enterprise Linux 5 Cluster Administration, Configuring and Managing a Red Hat Cluster , Redhat.
- [19] Cluster Computing, Kick-start seminar,16 December,2009, High Performance Cluster Computing Centre(HPCCC) Faculty of Science, Hong Kong Baptist University
- [20] Evaluating the Shared Root FileSystem Approach for Diskless High-Performance Computing Systems
Christian Engelmann, Hong Ong, and Stephen L. Scott
Computer Science and Mathematics Division,
Oak Ridge National Laboratory, OakRidge, TN37831, USA; {fengelmann,hongong,scottsl}@ornl.gov
- [21] A Performance Comparison of Linux and a Light weight Kernel
Ron Brightwell, Rolf Riesen, Keith Underwood
Sandia National Laboratories PO Box 5800
Albuquerque, NM87185-1110
{rbbright,rolf,kdunder}@sandia.gov Trammell B. Hudson
Operating Systems Research, Inc. Albuquerque, NM
hudson@osresearch.net Patrick Bridges, Arthur B. Maccabe
Computer Science Department, University of New Mexico Albuquerque, NM87131
{bridges,maccabe}@cs.unm.edu
- [22] Implementing Scalable Disk-less Clusters using the Network File System (NFS) James H. LarosIII, Lee H. Ward
Sandia National Labs, 30th October, 2003
- [23] NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters
Philip M. Papadopoulos, Mason J. Katz, Greg Bruno
The San Diego Supercomputer Center, University of California San Diego, La Jolla, CA 92093-0505
- [24] A Minimal Linux Environment for HighPerformance Computing Systems
James H. Laros III ;Sandia National Laboratories Albuquerque, NM 87123 Cynthia Segura, Nathan Dauchy
High Performance Technologies, Inc. Reston, Virginia 20190
- [25] Warewulf - <http://www.warewulf-cluster.org/cgi-bin/trac.cgi>