# Performance Analysis of Homogeneous Beowulf Cluster Setup on MPI Library

Tadrash Shah
Graduate Student (M. S.)
Computer Science
Stony Brook University, NY

Neel Patel
Graduate Student (M.S.)
Computer Science
Texas Tech University, Texas

Nishidh Chavda
Assistant Professor,
CSPIT - Changa
Charusat University

## ABSTRACT
Clusters have been an area of vast research in the domain of High Performance Computing and a variety of libraries are available for the cluster installation and administration. One such library named MPICH2 was selected and a selected variety of programs were run on the cluster setup and the performance of MPICH2 was tested against the prevalent cluster theories. The performance testing was done from simplest to quite heavy programs and they were tested for the execution time. The results affirmed the predicted pattern of execution time. Results also revealed the criteria for selection of number of compute nodes depending on the complexity of the problem to be solved. The results were encouraging for development of a large scale target application on the MPICH2 library, only held back by the fact that MPICH2 cannot support heterogeneous clusters.

## General Terms
Advanced Computing Technology, High Performance Computing, Cluster Computing, Parallel Systems.

## Keywords
Cluster; Beowulf; mpiexec; task distribution; mpiexec; calculation of Pi, MPICH2, OSCAR

## 1. INTRODUCTION
A cluster is setup for simultaneous execution of computational task on multiple processors for high throughput and higher performance gain. The Beowulf Cluster is normally a group of homogeneous commodity grade computers interconnected in a network running over a platform of libraries and programs which allow shared and parallel processing among the computers. Simple home computers connected with TCP/IP LAN with Linux installed as an operating system kernel and some cluster tool kits put together form the simplest Beowulf cluster. The biggest advantage is the cluster can be setup on any group of commodity grade computers with same system software.

Various cluster toolkits are available like OpenMPI, LAM-MPI, MPICH, OSCAR, etc.

The toolkit that has been used through development of Beowulf cluster is MPICH2, developed by Mathematics and Computer Science division - Argonne National Laboratory. MPI stands for Message Passing Interface which handles message passing mechanism among the cluster nodes. All the communication that needs to be done among the cluster nodes is done in the form of messages passed from one node to another. MPI provides these necessary process calls for this purpose and thus hides the underlying complexity from the developer. MPI, though not an IEEE or ISO standard, is essentially an "industry standard" [6]. MPI defines interface in C, C++ and FORTRAN languages. We have used the MPICH2 library of the version mpich2-1.4.1 and C implementation of the same has been chosen.

## 2. SETTING UP CLUSTER

### 2.1 Cluster Nodes
A cluster installation began with a preliminary setup with one master and two slaves which was then expanded. Through the paper we have shown the installation of 3 nodes which can be further replicated to many more nodes, for the sake of simplicity of understanding.

### 2.2 Hardware
Simple commodity grade computers with Intel Core2 Duo 2.93 GHz processor, 1 GB RAM and 250 GB hard disk with no graphics acceleration were used for this cluster setup. Although exactly these steps can be followed for virtual installation too.

### 2.3 Operating System
The system ran dedicatedly on Ubuntu 12.04 LTS 32 bit (i386 architecture) with configured bash shell and C/C++ compiler. The same system has been tested on RHEL 5.0 group of systems. We enabled root user which was used throughout the installation and for execution too. Also the virtual installation of all the nodes of cluster was does and the system tested thus.

### 2.4 Network
All nodes were connected with 24 port Ethernet switch 1-100 GBPS. Class B IP addresses in range of 172.16.0.1 as master and 172.16.0.XX as slaves were used. Host names were assigned from node00 onwards to node01, node 02 and so on with node00 assigned as the master node and all other being the slave nodes. This was done for easier access to each node rather than IP address. If this is to be virtually the network must be bridged.

### 2.5 Configuration
All then nodes in cluster systems must be aware of each other participating node, be it master or slave. This information was passed to each node through the host names. /etc/hosts [19] file was altered to confirm the host names of all the nodes on all the nodes. As a sample "hosts" file of node00 is shown below.

```
127.0.0.1 localhost
172.16.0.1 node00
172.16.0.2 node01
172.16.0.3 node02
```

Hence each node was intimated of host name of each other node. This facilitates the communication among the nodes with host names rather than IP addresses for abstraction and ease of access.

Password less login with use of authorized keys for SSH was used for communication among nodes to abate the requirement of administrator intervention each time when nodes must communicate. Hence master node which submits the job to slaves needed to be able to communicate with slaves through password less SSH. For this SSH was installed and enabled. SSH key for root user on all the nodes were generated using RSA key generation algorithm [6]. The keys were populated to all the nodes to allow automatic login on all the nodes. For the first time the master submitted the jobs, the SSH was authenticated and all later communication could follow without requiring authentication each time.

Files and programs of the MPICH2 toolkit were required on all the nodes for job execution. The modules of the toolkit were mounted on the master node to be accessed by slaves simultaneously through Network File System. The NFS-kernel-server was installed on master node and a directory in the root folder was setup to be shared among all the nodes. Hence this saved us from doing individual install of MPICH on all the nodes. The firewall in Ubuntu was disabled to unblock the slaves form accessing NFS directory.

## 2.6 Installing MPICH2

The MPICH2 package was downloaded from the site *http://mcs.anl.gov/mpi/mpich/download.html*. This library was installed on master node only in the directory which was shared among all the nodes through NFS. The download were extracted to the folder. Then the installation was done by firing the commands using terminal. Entering the MPI directory that was just created, the command was fired in bash shell as the root user [11]

```
./configure
```

The command sets up the check for the dependency that must be checked before installation of any package. After the configuration completes the command

```
make
```

was issued to setup the environment for the MPICH installation. All ran fine and finally MPICH2 was installed with a command

```
make install
```

Once this command was fired, it took few minutes to complete the process of installation. After all was done we were ready for executing the programs.

## 2.7 Running Programs

Once installed MPICH2 the sample program provided with the package was implemented directly. There was no need to compile the program as the object files were directly provided. The programs can be executed by specifying the number of processes that should run in parallel through an allowed option in the mpiexec command [10].

```
mpiexec -f machinefile -n <number> ./examples/cpi
```

The number of processes that can be accommodated by each compute node can be controlled with the help of machine file. The machine file specifies the number of processes that can be allowed on each node identified with its host name.

# 3. APPLICATIONS

Apart from the already provided examples in the MPICH2 package various other programs of MPI were tried. These were solely focused upon testing the process migration, effectiveness of machine file, load capacity of the cluster and the degree of parallelism that allows performance gain.

## 3.1 Task Division

A problem which is bent upon generating N numbers of tasks on master node and then distributing tasks to slaves depending upon their capacity is the task distribution problem. Thus it is a distribution of N processes on P no. of processors, which are essentially clustered. MPI environment is used for this migration on compute nodes. Hence, N may or may not be exact multiple of P.

The algorithm implemented is a "greedy" approach [15][21] and allowed approximate rounding of tasks, to avoid any left-over process. This also led to the result were one node may get one process less or more depending on the leftover. The process ids were indexed as provided by users and these indexes are used to send a process to processor [15]. The processes were distributed based on these indexes only. A sample output of this program is shown below –

```
Number of tasks T = 23

Number of processors P = 4

P_FIRST = 0
P_LAST  = 3
```

P_FIRST and P_LAST indicate the index of process ids so that process can be identified easily compared to the system assigned pid.

**Table 1 Task Division Problem**

| Processor | Number of Tasks | First Task | Last Task |
|---|---|---|---|
| 0 | 6 | 1 | 6 |
| 1 | 6 | 7 | 12 |
| 2 | 5 | 13 | 17 |
| 3 | 6 | 18 | 23 |

This concept established a clearer understanding of the process migration between the nodes. The program does nothing but just suspends, migrates, resumes and returns the process to and from the master node.

## 3.2 Calculating Value of Pi

A program to calculate the accurate value of mathematical constant PI (3.14) was evaluated for elapsed time and error in the calculated value. The benchmark value of PI was considered up to 25 decimal places whereas cluster computed the value up to 16 decimal places. Hence the error could be identified for 9 decimal places in accuracy as compared to the benchmark value using 25 decimal places. The error was observed to show very minor change which negligible and hence we focused mainly on the execution time of the program. Extensive use of machine file was made for submitting the processes.

The Task Division problem was a simplest program to know the process distribution. This example goes a step further. Here the processes are allowed to move back and forth the master node depending upon the free resources. The resource

allocation was made more dynamic than previous example. For example, say master node was allowed two processes. When the third process is to be scheduled, it will be scheduled on one of the slave node. In the meantime the fourth process is also queued and also one of the process on master node is terminated. Then there is no need to send the fourth process to the slave node, it will be executed on the master node itself.

Hence, processes were allocated dynamically depending on the free resources on a node, be it master or a compute node. Also note that the empty cells in table indicate that no output was obtained due to submission of processes beyond capacity of cluster.

**Table 2 Calculation of Value of Pi**

| Number of Processes | Node = 1 | Node = 3 | Node = 5 |
|---|---|---|---|
| 5 | 0.001481 | 0.001008 | 0.001211 |
| 10 | 0.001641 | 0.003463 | 0.00357 |
| 50 | 0.047269 | 0.108689 | 0.11757 |
| 100 | 0.115619 | 0.252974 | 0.2752 |
| 200 | - | 0.55463 | 0.584591 |
| 300 | - | 0.98335 | 0.993251 |
| 400 | - | 2.901754 | 2.925887 |
| 550 | - | - | 3.102552 |

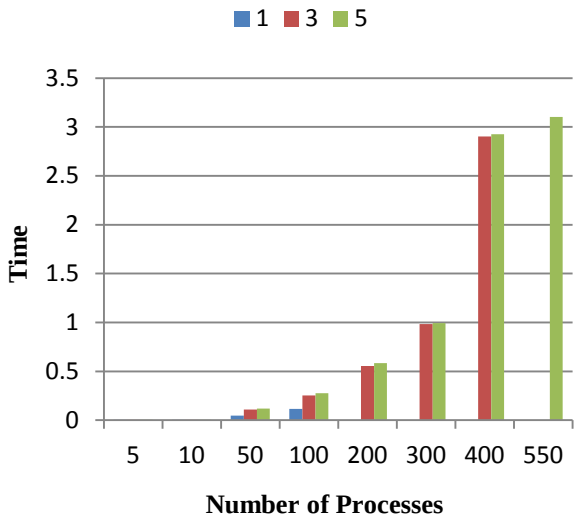


**Figure 1 Calculation of Value of PI - Analysis**

It can be seen from the graph that as the size of the problem increases i.e. the number of process increases the number of nodes needs to be increased else the problem cannot be solved to success. It was seen that a single node was not capable of running processes more than 100 whereas 5 nodes could run the problem as big as the 550 processes which is more than 5 times. In a sense the capacity of computation was increased by more than 5 times when running the same task on the cluster as compared to single node. Also it was seen that execution time of the same problem, on increasing the number of nodes, increased negligibly. This increase was because the time incurred on process migration which involved the process suspension on master node and resume on slave node. As the number of processes were increased to few hundred the time to break-up the problem and combine it for consolidated result was much higher than time incurred in migration. With increase in number of nodes the size of problem could also increase.

## 3.3 Sum of n Numbers

The problem considered so far were small problems which were not too heavy on execution. The size of the process was quite small. Hence, we considered a problem with larger size which could potentially require considerable amount of resources as well as CPU time. The problem was to add 1 to N numbers. Seemingly simple, the problem added considerable range of numbers like 1 to 1000 and other such [15]. As the processes were quite large in size the number of processes that the same number of nodes could run was less as compared to previous problem.

Again from the graph (Figure 2) shown it is clear that with increase in number of processes and corresponding increase in number of nodes the time required to complete the execution of the problem usually increases, though some variation are evident in 20 processes and an unrecognized peek in nodes 5 for 7 processes were seen. This peak was somewhat unpredictable but it was hypothesized that it may due to some effects of program structure. But rest of the observations that were see in calculation of value of Pi are the same, apart from the evident fact that the program execution was so heavy that number of processes cannot even reach a single hundred where in previous program it could reach to the order of few hundreds.

**Table 3 Sum of N Numbers**

| Number of Processes | Node = 1 | Node = 3 | Node = 5 |
|---|---|---|---|
| 5 | 0.002407 | 0.004257 | 0.005869 |
| 7 | 0.002375 | 0.010305 | 0.012143 |
| 10 | 0.003646 | 0.004451 | 0.005208 |
| 20 | 0.005789 | 0.004949 | 0.004637 |
| 30 | 0.008053 | 0.008549 | 0.009261 |
| 40 | 0.042496 | 0.043341 | 0.045106 |
| 50 | 0.04908 | 0.054628 | 0.058911 |
| 60 | 0.063604 | 0.064366 | 0.066098 |

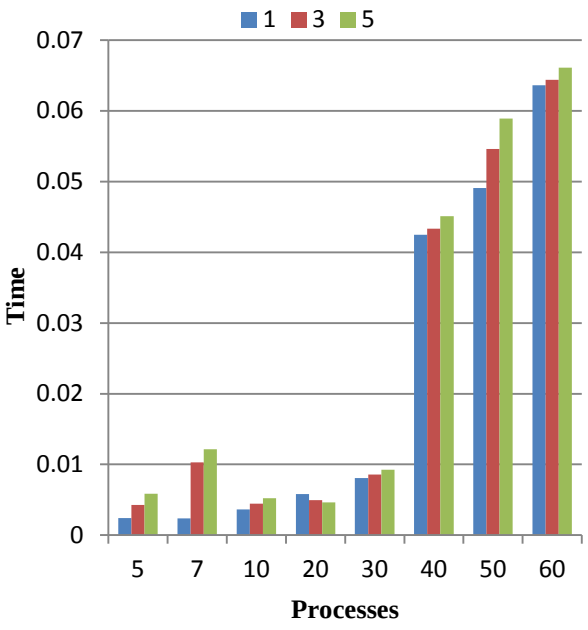


**Figure 2 Sum of N Numbers - Analysis**

It was also confirmed with the statement of G.Pfister that cluster mainly had three major principles: viz. Work Hard, Work Smart and Get Help. We did not work had as we did not increase the processing power on single node. We neither worked smart as we did not work much in optimization of the used algorithms. But we did Get Help by connecting commodity grade computers in network and exploiting their otherwise idle processing power for the benefit of a large problem execution on the master node.

## 4. DISCUSSION

It can be observed from the execution of the said MPI programs that number of nodes in a cluster must be in accordance with the target application. Also that a larger application needs more number of compute nodes else the problem cannot be solved due to shortage on resources. The time required for process migration and consolidation of the result on the master node increase with increase in number of nodes. Thus it can be noted that number of nodes must be increased with a care so that performance gain can be genuinely achieved.

Also through this way of clustering the master node becomes a bottleneck for a large number of processes. Furthermore, this cluster did not focus critically on the security issue. Neither has the security been compromised nor imposed stringently.

After extensive working with MPICH2 it was also observed that MPICH2 cannot support a cluster environment that has heterogeneous representation of data dependent on the nodes. There are number of things that need to be done before this cluster can be made into an application cluster.

## 5. CONCLUSION

There are some major conclusions. The clusters are indeed a powerful way, and even better when a cluster can be constructed from commodity grade computers to execute a larger process. Due to the limitation of the MPICH toolkit of working only on homogenous cluster system, the authors have thought of diverting to OSCAR [20], another toolkit that is found better for data-intensive application and allows GUI based wizard tool for installation and management of a Beowulf Cluster. The cluster setup described cannot be an extensive innovation in the dimension but surely is a precursor to the larger application that can be potentially built on OSCAR toolkit.

Authors also welcome any constructive criticism or suggestion in the discussed field.

## 6. ACKNOWLEDGMENTS

Authors would like to express their gratitude to Charotar Institute of Technology, Charotar University of Science and Technology, Changa for equipping them with all the necessary infrastructure and aesthetics. The institute and the heads took utmost care for the congenial environment to be provided to the authors for research. Authors are obliged and indebted.

## 7. REFERENCES

[1] Rajkumar Buyya, "High Performance Cluster Computing", Vol 1, Pearson Education, 1999.

[2] Amit Jain, "Beowulf Cluster Design and Setup", Boise State University, April 2006

[3] Discussion of topics related to Beowulf Clusters - http://www.beowulf.org

[4] News and software site for the Beowulf community - http://www.beowulf/underground.org

[5] Dr. Deven Shah, "Advanced Computing Technology",Dreamtech Press, pp.2, 2011.

[6] T. Sterling, J. Salmon, D. Becker and D. Savarese, "How To Build a Beowulf", MIT Press, 1999

[7] An overview of Beowulf clusters, from cluster design, to cluster use and maintenance by Mike Perry - http://fscked.org/writings/clusters/cluster-1.html

[8] Message Passing Interface Tutorials by Blaise Barney - https://computing.llnl.gov/tutorials/mpi/

[9] MPICH2 Official page - http://www.anl.gov/research/projects/mpich2/

[10] MPICH2 User's Guide

[11] Using MPICH to Build a Small Private Beowulf Cluster - http://linuxjournal.com/article/5690

[12] T. sterling, “Beowulf Cluster Computing with Linux”, MIT Press, October 2001

[13] R.J. Allan, S. J. Andrews and M.F. Guest, “High Performance Computing and Beowulf Clusters”,

[14] 6th European SGI/Cray MPP Workshop, Manchester, 7-8/9/2000.

[15] Examples of MPI Programs from Florida State University - http://people.sc.fsu.edu/~jburkardt/c_src/mpi/mpi.html

[16] Tadrash Shah, Neel Patel, Nishidh Chavda, “Formulation of Homogenous Cluster Environment using Commodity grade Computer and MPI Paradigm”, Vol.1 Issue 5, IJARCSEE, August 2012.

[17] Robert Brown, “Engineering a Beowulf-style Computer Cluster”, 2004, Duke University, Physics Department

[18] Kerry D. Wong, A Simple Beowulf Cluster

[19] Ubuntu Wiki, Setting Up MPPICH2 Cluster in Ubuntu

[20] OpenClusterGroup, OSCAR.

[21] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, Willian Saphir, Marc Snir, "MPI: The Complete Reference", Vol. 2, The MPI-2 Extensions, Second Edition, MIT Press, 1998.

[22] Tadrash Shah, Neel Patel, Nishidh Chavda, "Cluster Computing using MPI Paradigm : A Practical Approach", ISBN : 978-3659296185, December 2012.

[23] Preliminary Cluster with OSCAR registered at - http://svn.oscar.openclustergroup.org/php/clusters_register.php?cid=270