Study and Evaluation of Cluster Technologies (OpenMosix vs. MPI)

M Kashif Ghafoor[1] and Asim Riaz[2]
SZABIST
Karachi, Pakistan

ABSTRACT: *Most of the time computers in any organization are not consuming the resources and are idle. Unfortunately, when one requires those computing/CPU powers, the requirement is in bulk and at once. The idea behind clustering is to spread these loads/processes among all available idle computers, using the resources that are free on other machines. Two different clustering paradigms are available for implementation of cluster by any organization. Process migration and message passing are two different techniques used in making clusters.*

This IS covers the study and evaluation of cluster technologies for implementation in a government sector. Comparison of these technologies will provide the basis for choosing the most suitable technology for the organization.

Key Words: *Cluster computer, parallel processing, Process migration, Message passing.*

1. INTRODUCTION

In recent years, interest in high performance computing has increased [7] [10]. With the advent of new technology, the unused resources of computer can now be utilized simultaneously through cluster to achieve the results in quick and less time. Clustering is the technology by which number of computers in the network takes part to solve a big problem through distributing the jobs to other computers. Clusters are mainly used where tremendous amount of processing power is required to solve the heavy problems. Major tasks supported by the cluster tools consist of shorten installation process, provides system wide monitoring, support migration, etc [2].

Cluster computing has been used for many years as the primary computational platform for scientific applications [1]. Now scalable computing cluster, ranging from cluster of homogenous servers to a heterogeneous network of workstation, are rapidly becoming the standard platform for executing demanding applications, high performance and interactive computing. Two models that are used to achieve these goals are process migration and message passing [12] [13].

Process migration model is a technique in which a live process is transferred from one system to another. Mosix/OpenMosix is an example of this technique [14]. Message passing model is implemented in user space as a group of libraries. Application developers write their code according to set of standards. Message Passing Interface (MPI) and Parallel Virtual Machine (PVM) are two example of this technique [11].

Here in this paper, I will study and evaluate both high performance clustering techniques and will compare their performance with some experiments, focusing on the common goal of finding the best cluster paradigm for the organization.

2. ORGANIZATIONAL ENVIRONMENT

The setup of our organization is widely spread across the country. Five Unix servers are installed at different locations in Pakistan. These servers provide different application services ranging from inventory management to the simulation programs.

The servers at night need a lot of extra computational power to keep the system online besides taking backup of the system on tape libraries. Besides other routine activity, the system must be kept online for any transactions to meet the requirement of end user. Furthermore in some servers a lot of programs also run consuming bunch of computer resources resulting in time wastage. To provide access to every user at that part of time, whether for administering a database or updating the application or taking backup, the system available should be quite efficient and capable to deal with the entire task in lesser time utilizing all the idle resources.

3. OPENMOSIX

OpenMosix is a free cluster management system that provided single-system image (SSI) capabilities [6]. It is an extension to the kernel of the operating system, which turns the ordinary computers in the network into the cluster [5]. OpenMosix doesn't require additional programming to balance the load on computers, and computers can leave or join the cluster without any disruption. Every application automatically and transparently benefits from distributed computing concept. It allows the program processes to migrate to the less utilized nodes, thus automatically balancing the load on the computers.

3.1 OpenMosix Composition

OpenMosix has no central control or master/slave relationship between nodes. Each node operates as an autonomous system and makes all its decisions independently. This allows a dynamic configuration, where nodes may join or leave the network with minimal disruptions.

3.2 Advantages/Disadvantages of OpenMosix

Although MPI is the current technology, but the old technology openMosix is still popular among the cluster users. The openMosix provides following advantages [4]:

1. No requirement of any special library to distribute the processes in cluster.
2. Processes will be distributed over the cluster as soon as a node has less load comparing to other nodes and thus reduces the load on the server.
3. OpenMosix runs on any Linux flavor after applying kernel patch.
4. OpenMosix makes it possible to create a cluster out of old hardware.

Due to its old technology, openMosix has following disadvantages [4]:

1. The openMosix does not provide any security.
2. If a node is not reachable, the server will crash resulting completely restarting your cluster and loosing the work.
3. Jobs that have failed cannot be reassigned to the cluster and considered as lost.
4. The cluster's performance does not grow much anymore after connecting six nodes.

4. MESSAGE PASSING INTERFACE (MPI)

MPI is a standard for message-passing for distributed-memory applications used in parallel computing. It is a portable, efficient and flexible standard library for writing parallel programs [17].

4.1 MPI Composition

MPI clusters are scalable HPC clusters based on commodity hardware, on a private network, with open source software infrastructure [3]. MPI clusters usually operate on master/slave concept. The main program is started on the master who starts a bunch of worker-tasks on the remote nodes. All the management of jobs is centrally handled from the master node [18].

MPI works on MPSD (multiple program single data) execution model. The programs task running on each node synchronizing variables and exchanging data by e.g. broadcasting + gathering. The sub-results are combined afterwards when each program task computes its part of the data [15].

4.2 Advantages/Disadvantages of MPI

MPI came into the mainstream more recently than other message passing paradigms (e.g., PVM). It has become popular because of the following features [9]:

1. MPI provides a high degree of portability.
2. MPI library transparently performs the appropriate data conversion when data are transfer between different heterogeneous systems.
3. MPI standards are flexible.
4. MPI is widely supported by most of the vendors of parallel systems.

The disadvantages of MPI are:

1. Need to have expertise level for writing efficient parallel program.
2. You need to write complex parallel program by using MPI libraries.

5. EVALUATION

5.1 Computer Experiment One

The cluster is made of four computers, one P-III of 800 MHz and 03 P-II of 400 MHz. All the computers are having 256 MB of RAM and are connected with 100 Mb networks. A small database with two datafiles of different capacity is built on each computer (two of 2 GB, two of 1.5, two of 1 and two of 0.5) for the purpose of test by varying file sizes.

The copy process would be executed simultaneously with and without the simulation to check the results [8].

5.1.1 Tests With Simulation

5.1.1.1 without OpenMosix

P-III recorded the I/O of about 30m per GB, resulting in total time of about 5h (300m). And P-II recorded the I/O of about 20m, resulting in total time of 3h 20m (200m). The result obtained with this experiment shows that each process has copied only its data. Hence it's not the parallelism but simultaneity of events [8].

5.1.1.2 With OpenMosix

Firstly the copy program was executed without simulation only to check the adaptive resource algorithm of openMosix. It was noted that processes migration from P-II to P-III node was working fine.

Subsequently the experiment was repeated with simulation on P-III node. And effective load balancing was observed. But additional load on P-II node degraded its performance and the copy time recorded was 25 – 30 minutes per GB [8]. Comparison for two of the above experiments is depicted in table 1.

	Without openMosix	**With openMosix**
P-II total time / node	3h 20m	4h 30m
P-III total time	5h	4h
Balancing	Small	good

Table 1[8]

5.1.1.3 Using MPI

MPI requires extra work of program coding for parallelization. The tests were conducted using 1, 2 and 4 processes for each datafiles. Using MPI, P-II nodes recorded the copy times of 13 – 16 minutes and P-III recorded with 16 – 17 minutes.

	Without openMosix	With openMosix	Using MPI
P-II total time per node	3h 20m	4h 30m	2h 30m
P-III total time	5h	4h	2h 50m
Balancing	Small	good	sufficient

Table 2 [8]

5.1.2 Test Without Simulation

To get clearer picture between MPI and openMosix, the copy test was executed without any simulation. The test was executed thrice with a datafiles set of 8 GB on each node, 20 files of 0.4 GB, 8 of 1 GB, 6 of 1.4 GB and 4 of 2 GB respectively.

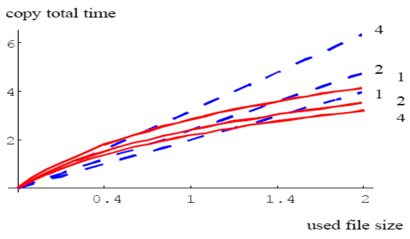

Fig 1[8]: Blue dashed line shows result of openMosix and red continuous line shows MPI.

Though the load balancing performed by openMosix is correct and good, but MPI technology takes a lead with its multiprocess programming [8].

5.2 Computer Experiment Two

The aim of this experiment was to check the efficiency of parallel runs and serial execution within their domain. The experiment was performed on two different clusters. P-III cluster with FE network consists of 16 CPUs (1.4 GHz) with total RAM of 16 GB. P-IV cluster with Myrinet also consists of 16 CPUs (2.2 GHz), but with total RAM of 08 GB [16].

5.2.1 Serial Execution on OpenMosix & Parallel on MPI

The task of 20 different simulations was executed on P-III cluster with FE network and P-IV based clusters with Myrinet network. Results collected from different experiments are enumerated in table 3.

P-III	**MPI (4 CPUs)**	**MPI (8 CPUs)**	**openMosix**
Total Time	1013	1138	894
Avg. Time	193.4	142.5	687.5
P-IV	**MPI (4 CPU UP)**	**MPI (4 CPU SMP)**	**openMosix**
Total Time	1432	2082	1066
Avg. Time	139.4	206.3	844

Total Time refers to Total execution time which is from the start of job execution to the job ending.
Avg. Time is the average simulation time of 20 simulations.

Table 3 [16]

It is clearly evident that openMosix technology is superior for this type of job. Although the total time for P-IV based MPI cluster is higher than P-III cluster which further go high as the number of processors increases. But the cluster also benefits from Myrinet network technology and average time for P-IV based MPI cluster decreases as compared to P-III cluster with Fast Ethernet [16].

5.2.2 Parallel Execution on Both OpenMosix & MPI

The goal of this experiment was to test the results obtained through running parallel program with and without openMosix technology.

Lattice Size	openMosix	MPI
4x4x4x4	5.3	5.6
8x8x8x8	17.2	17.6
12x12x12x12	101	126

Table 4 [16]: times are in seconds

Job with a smaller lattice size doesn't make much of a difference in the results. But as the lattice size grows the difference in time is remarkable, thus clearly indicating the openMosix technology to be superior.

6. CONCLUSION

These fundamental differences between the main structures of the two technologies provide advantage to one over the other. A great advantage of openMosix clusters comparing to MPI is that no special codes are required to be written to take the taste of implementing clusters.

MPI programming is quite complex. These can only be used in small dedicated communities i.e., research & development, space etc. We can conclude that the MPI cluster may be a little too complicated due to its special needs. And with openMosix the user does not need to worry about the program structure. Its simple ability of patching with kernel of operating system made it popular among the users. Moreover, it can be built from already available normal computer (PCs) within the organization.

7. REFERENCES

1. P Anedda, M Gaggero, G Busonera, O Schiaratura, G Zanetti. Flexible Clusters for High-Performance Computing, IEEE, 2010.

2. S Prueksaaroon, V Varavithya, S Vannarat. An Implementation of Virtualization Cluster: Extending Beowulf Cluster using Virtualization Cluster Management and Image Storage, IEEE, 2009.

3. Beowulf Project Overview, Beowulf Cluster Site, http://www.beowulf.org/overview/index.html, 2007.

4.Michels, W. Borremans, Clustering with openMosix, http://staff.science.uva.nl/~delaat/snb-2004-2005/p20/report.pdf, University of Amsterdam, 2005.

5. OpenMosix: An open source Linux Cluster Project, URL: http://www.openmosix.sourceforge.net, June 2004.

6. R Lottiaux, C Morin, G valle, P Gallard, D Margery, J Y Berthou, I Scherson, Kerrighed and data parallelism: Cluster computing on single system image operating systems. In Proceedings of Cluster 2004 – IEEE, Sep 2004.

7. A Boklund, C Jiresjo, and S Mankefors, The Story Behind Midnight; A Part Time High Performance Cluster, in proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, USA, Vol. 1, pp. 173-178, June 2003.

8. G Argentini, Workshop Linux cluster: the openMosix approach: Use of openMosix for parallel I/O balancing on storage in Linux cluster, Nov 2002.

9. Introduction and Advantage of MPI, http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/CommonDoc/MessPass/MPIIntro.html, Dec 2002.

10, Rajkumar Buyya, A Study on HPC Systems supporting Single System Image, Techniques and Applications, in Proceeding of the International Conference on Parallel and Distributed Processing Techniques and Applications, USA, 1997.

11. Blaise Barney, Message Passing Overview: Parallel Programming Workshop, URL: http://www.hku.hk/cc/sp2/workshop/html/message_passing/message_passing.html#message1, Jul 1996.

12. E T Roush and R H Campbell, Fast Dynamic Process Migration, in Proceedings of the 16th International Conference on Distributed Computing Systems, IEEE, pp. 637-645, 1996.

13 Y Paindaveine and D S Milojicic, Process vs. Task Migration, in Proceedings of 29th Hawaii International Conference on System Sciences, Software Technology and Architecture, Maui Hawaii, vol. 1, pp. 636-645, 1996.

14. F Douglis, Experience with Process Migration in Sprite, in proceedings of Workshop on Experience with Building Distributed and Multiprocessor Systems, Fort Lauederdale, FL, pp. 59-72, October 1989.

15. H J Sips, Programming Languages for High Performance Computers. URL: http://cdsweb.cern.ch/record/400331/files/p229.pdf.

16. Moshe Bar, S Cozzini, M Davini, A Marmodoro, OpenMosix vs. Beowulf: a case study, URL:http://www.democritos.it/activities/IT-MC/openMosix_vs_Beowulf.pdf.

17. Web site for MPI libraries, High Performance and widely portable MPI, http://www.mcs.anl.gov/research/projects/mpich2/

18. Web site for parallel programming for MPI, http://www.mhpcc.edu/trainingworkshop/mpi/MAIN.html#Message_Passing.