Clustering with openMosix

M. Michels & W. Borremans

February 2005

Master program System and Network Engineering
University of Amsterdam

Abstract

openMosix is a Linux kernel extension for single-system image clustering which turns a network of ordinary computers into a supercomputer. This project focuses on the performance, reliability and characteristics of openMosix. We also asked ourselves where it could be implemented. There are some limitations with a openMosix cluster. We have our remarks looking at the reliability of the cluster application. According to our findings threaded applications can be used, though the threads will not be distributed over the cluster. This means no increase of performance. Due to the large amount of programs that actually use these kinds of programming models, not many applications can be used.

The performance of the cluster dramatically drops when adding more then 12 nodes. The cluster's stability is depended on all nodes, when one node fails the complete cluster could crash.

Contents

1 Introduction

This document describes the findings in the openMosix[1] project carried out in order of the RP1 subject of the 'System and Network Engineering[23]' course at the University of Amsterdam[24]. The project is carried out by Maarten Michels[25] and Wouter Borremans[26] and is supervised by Harris Sunyoto[27].

This project focuses on the performance, reliability and characteristics of openMosix in which we will try to answer the following questions:

- Performance
  - Is the performance linear according to the number of nodes?
  - What kind of applications will benefit from openMosix?
- Reliability
  What happens if a specific part of a node fails?
- Network load
  How is the network load related to the number of nodes?

Depending on the answers of the above stated questions, we will conclude if openMosix is suitable in our field of study. In this report we will first explain what exactly a cluster is, after that we will continue to focus on openMosix.

1.1 Related work

As the popularity of openMosix is increasing, more and more people are starting to do experiments with it. We found several sites on which we could find reports of the experiments.

- Creating a low latency high performance gameservers[19] - Democritos[20]
  - Italian researchers tried to build a gameserver cluster using openMosix.
- openMosix Cluster - University of Kiev[21]
  -created for solving scientific and applied problems that require large computing power and operate with large volumes of information.

We were unable to find reports on performance numbers and reliability information on openMosix, we started some testing of our own.

1.2 What is a cluster?

Definition according to an English dictionary:

> ***cluster***; bunch, clump, cluster, clustering – (a grouping of a number of similar things; "a bunch of trees"; "a cluster of admirers")

A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers cooperatively working together as a single integrated computing resource.

These jobs vary from complex mathematical problems to generating high resolution images. Jobs like searching for extraterrestrial intelligence[2] are a well known projects in which multiple computers connected over the internet are working together on just one project.

When designing a cluster in general, there are a few goals to achieve;

- Complete transparency
  - Single entry point, think of FTP NFS etc.
- Scalable performance
  - Easy growth, the ability to expand your cluster with new machines continuously.
- Enhanced availability
  - Automatic recovery from failures, when a node fails the cluster automatically reruns the failed job.

The most used technique to build up a cluster is SSI (Single System Image). SSI is the illusion created by software and hardware that represents a collection of computing resources as one, more whole resource. SSI makes the cluster appear like a single machine to the user, to applications and to the network. Depending on the job you have to carry out, different kinds of clusters are applicable;

HTC High Throughput Cluster - Primary used for serial applications (i.e. openMosix)

HPC High Performance Cluster - Most of the times used in the computational science, think of clusters that carry out jobs of Shell or the Mathematics department of the University of Amsterdam.

SSI makes the use of system resources transparent and will offer improved system response time and performance. It simplifies the management because the system administrator does not have to know the underlying system architecture to use the machines effectively.

Computers in clusters are interconnected by a network, the network is -besides the nodes- the most important part of cluster. The network is the limit of the cluster, this means the higher the bandwidth the higher the performance of the cluster will be. The currently most used networks of 100Mbit and 1Gbit are in most cases not sufficient anymore. Other types of networks are available to face the need for higher bandwidth; SCI (Dolphin), Qsnet, Myrinet and Infiniband. These types of networks offer extreme high bandwidth with extreme low latency[1].

In general, the computer communication ratio in a parallel program remains fairly constant. If the computional power increases, the network speed must be increased also.

1. *Latency* is discussed in chapter 4.2

1.3 What is openMosix?

openMosix is a Linux kernel extension for single-system image clustering. This kernel extension turns a network of ordinary computers into a supercomputer for Linux applications. Once you installed openMosix, the nodes in the cluster will start talking to each other by exchanging messages. The cluster adapts itself to the workload.

openMosix adds cluster functionality to *any* Linux flavor. openMosix uses adaptive load balancing techniques, processes that run on a node can transparently be distributed to one another. Due to the complete transparency of openMosix, a process does not have to know where it is running. The process 'thinks' that it is running locally.

In this case, this transparency means that no additional programming is needed to take advantage of the openMosix load-balancing technology. This is a great and powerful feature of openMosix, it really lives up it's name. openMosix turns multiple Linux hosts into one large virtual SMP (Symmetric Multi Processor). Real SMP systems with two or more physical processors can exchange large amounts of data, in practice this means that real SMP systems are much faster. With openMosix, the speed at which the nodes can exchange data is limited to the speed of the LAN connection. Using a high bandwidth connection will increase the effectiveness of your openMosix cluster.

Another great advantage of openMosix is the ability to build a cluster out of inexpensive hardware giving you a traditional supercomputer.

openMosix can also be used with performance enhancing techniques like Hyper-Threading[4] available on intel Pentium 4 and Xeon processors. Using this technique enables you to enhance the performance of a node. The node can now handle multiple cooperating threads that cannot be separated and distributed among openMosix nodes. Use of the technologies mentioned may result in significant performance increase.

Besides the use of openMosix on Hyper-Threading systems, it has efficient parallelization algorithms of its own.

1.3.1 openMosix vs Beowulf

Together with openMosix, Beowulf[5] is one of the most well known cluster applications. The Beowulf project was started by Donald Becker in early 1994 at the NASA Goddard Space Flight Center. NASA was looking for a solution to build a cluster out of relatively inexpensive hardware. They ended up in using Beowulf to beat Grendel[6] .The project gained great popularity of other NASA research groups. The number of Beowulf clusters has grown relatively fast over the past few years.

There are some fundamental differences between the main structure of the two programs. A great disadvantage of Beowulf clusters comparing to openMosix is that only programs written using the PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) libraries are supported. The PVM and MPI libraries are the most used libraries by programmers which program for clusters. The PVM and MPI libraries allow computing problems to be solved at a rate that scales almost linearly in relation to the number of nodes in the cluster. Since PVM and MPI programming is quite complex, these kind of techniques will only be used in small dedicated communities. We can conclude that for the 'regular' cluster user Beowulf may be a little too complicated due to it's special needs, with openMosix the user does not need to worry about the program structure.

1.4 Why openMosix?

As mentioned above, openMosix does not need specially written applications to benefit from the cluster capabilities. This is biggest issue choosing an cluster application. openMosix makes it possible to create a cluster out of your old hardware. New nodes (with different hardware configurations) can be added dynamically, the cluster will automatically discover the new node and will send jobs to it. With openMosix you can use very different configurations for different use. The project has a large community which constantly adds new features and fixes bugs. More and more distributions come available which already have the openMosix module in it's kernel. Think of Cluster Knoppix[7] or a specially designed distribution PlumpOS[8].

OpenMosix offers a great opportunity to create a cluster for a low budget institution. Off course openMosix has also some disadvantages, these will be discussed in the conclusion of the report.

2 Methods

In this section we will describe how we are going to conduct the benchmarks on the cluster.

2.1 Hardware being used

In order to test openMosix in a proper way we received fourteen computers of our university. These computers have the following configuration:

- Intel Pentium III 1Ghz
- 256MB RAM
- 20GB Harddisk
- 3COM 3C905 10/100 network card

The computers are coupled to each other using an SuperStack 3 3300TM switch offering an 100Mbit Ethernet switched network connection to each node.

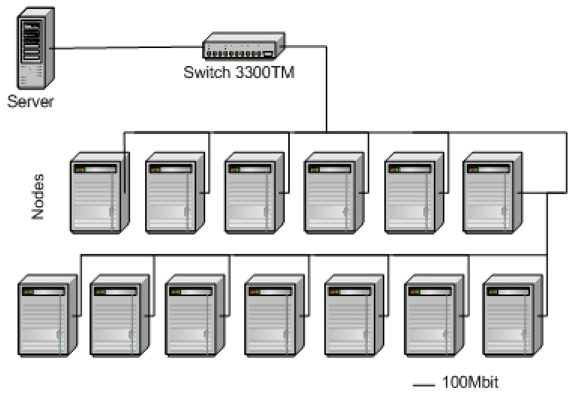

Figure 1: Test setup openMosix cluster

We created a netboot environment using Cluster Knoppix. Cluster Knoppix is a Linux (Debian[9] based) operating system which runs completely from CDROM. When booting up, Knoppix automatically detects all the hardware and starts up a graphical user interface called KDE[10]. We made a hard disk install on the server of the cluster. Cluster Knoppix already has an openMosix modified kernel built in. The package comes with several openMosix utilities to smoothly built up your cluster. The server computer acted as the main node for the cluster, DHCP and NFS server. The clients booted from the server using a TFTP netboot. The netboot environment did not completely face our needs so in some area's we had to change the configuration.

2.2 Applications being used

Before the project started, we looked for some programs which would run smoothly on the cluster and would give a good idea of the cluster's performance. Initially we did not know for which kind of applications to look for. After some testing and reading it became clear that there was no possibility to run threaded applications on the cluster. This made the selection of the programs relatively easy. More on this issue can be read in the section about Threads and Processes in chapter 4.1.

2.2.1 Povray

Povray is a persistence of Vision Ray-Tracer(tm) and was developed from DKB-Trace 2.12 (written by David K. Buck and Aaron A. Collins) by a bunch of people (called the POV-Team.) in their spare time. The POV-Ray package includes detailed instructions on using the ray-tracer and creating scenes. Ray-tracing is a rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world. Ray-tracing programs like POV-Ray start with their simulated camera and trace rays backwards out into the scene. The user specifies the location of the camera, light sources, and objects as well as the surface texture properties of objects, their interiors (if transparent) and any atmospheric media such as fog, haze, or fire.

For every pixel in the final image one or more viewing rays are shot from the camera, into the scene to see if it intersects with any of the objects in the scene. These ”viewing rays” originate from the viewer, represented by the camera, and pass through the viewing window (representing the final image).

The standard povray package cannot be used on a openMosix cluster. Normally there is only one job which can only be run on a single machine. To be able to distribute the povray job on the cluster, we used PovMosix[11]. PovMosix splits the rendering scenes into sub-jobs, these jobs can be distributed over the cluster which results in a interesting performance increase.

After the the rendering (sub)jobs are done, the tool migrates the (sub)jobs together again.

2.2.2 Encryption

We used a C++ program called 'Distributed Key Generator' written by Ying-Hung Chento[12] to test how the cluster performs generating RSA public/private key pairs. The program will generate 4000 public/private keypairs with 1024 bits. It forks 14 processes so it can be distributed over the openMosix cluster. Each of the processes running on the nodes will generate a part of the 4000 keys.

2.2.3 Compiling

The compiling benchmark was performed with the regular 'make' procedure. After downloading a complete kernel from Kernel.org we started the compiling progress with the option ˇj28` (two processes per CPU) to be able to distribute the processes over the cluster.

2.2.4 Synthetic benchmark

For this test we were going to use LAPACK[22] (Linear Algebra PACKage). LAPACK provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complexmatrices, in both single and double precision. It also has optimized libraries to run efficiently on shared-memory vector and parallel processors.

2.3 Collecting the test data

To be able to collect test data, we wrote some shell scripts. We used several -on the Cluster Knoppix installation available- tools such as:

MySQL[16] database server. We used this application to inject the test data originating from the console into a database. See the shellscript in appendix A.

IPTraf[17] Unix utility to monitor and summarize the network traffic of a specific interface

Time[18] Unix utility to monitor the System, User and process time of a program started.

You can find the scripts in appendix A. All the tests were performed three times in order to create reliable test-data. Our shellscripts worked as follows collecting the test-data:

- Start the traffic logger (IPTraf) at a specific time.

  IPTraf starts collecting traffic information at the server node on a specfic interface.
- Actual test begins (with IPTraf running on the background)
- Test stops, console output is stored in the database using an SQL statement
- The PID (Process ID) of IPTraf is resolved and will stopped sending a specific kill signal.
- Bash script stops running.

2.4 Reliability testing

We tested the cluster reliability by shutting down nodes and removing network connections during tests.

3 Results

In this section we will analyze the test data from the experiments.

3.1 Experiments

To test the cluster's performance we looked for some programs which we could use to do some tests. We choose for three kinds of which we thought were useful. In the following paragraphs each program will be discussed.

3.1.1 Povray

Test results
In this test we generated a picture with PovMosix[11] from a file '*Benchmark.pov*'. This file contains the instructions to generate a 1024x768 high resolution picture. The results of this test can be found in figure 2 diplayed below. Analyzing

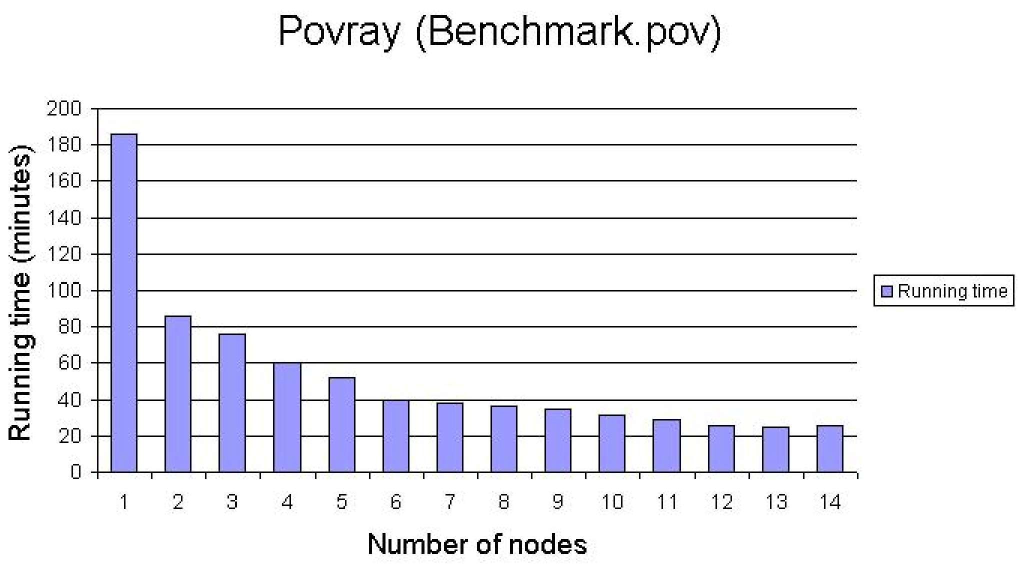

Figure 2: Povray: # nodes vs running time

figure 2 tells us that;

- A significant performance increase between 1 node and two nodes. (nearly 50%!) can be mentioned
- After six nodes, the performance does not grow much
- By adding more then 12 nodes, the performance does no longer increase . This is probably because of the (network) overhead.
- The performance is not linear according to the number of nodes.

In this test we were unable to collect network traffic data. IPTraf does not support multiple instances which were needed with this experiment. (See chapter 2.2.1)

3.1.2 Encryption

Testresults
Below in figure 3 you can find an overview of the findings on this benchmark.

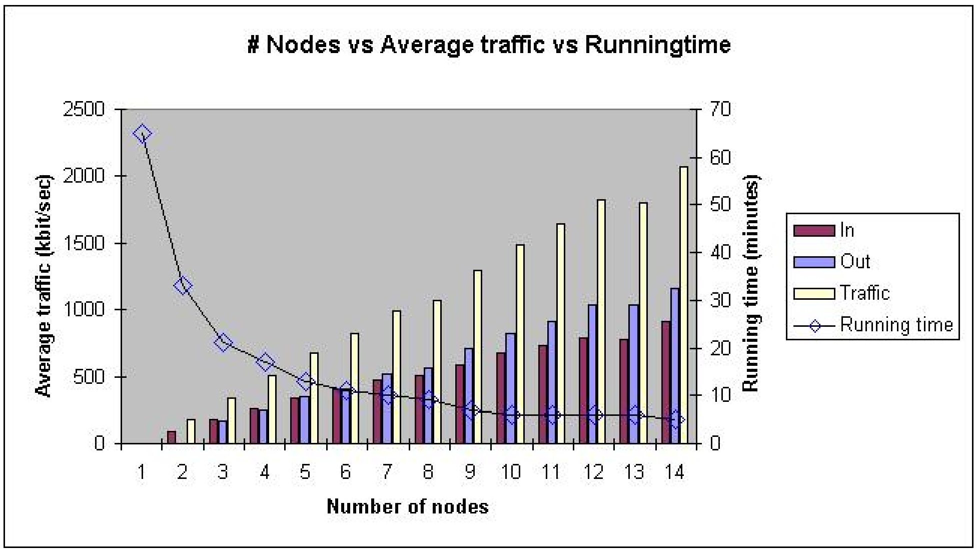

Figure 3: Encryption: Nodes in relation to traffic and running time

Looking at the benchmark results of this test, we can conclude that;

- We see the same trend as with the povray benchmark looking at the running time.
- Adding a node will increase the traffic by approximately 20%
- The increase of the network traffic influences the server node in such a way that the performance of the cluster itself will dramatically drop. For more information about this matter see chapter 4.2 about latency.

3.1.3 Compiling

Due to the way ‘make‘ works, we ran into much problems compiling a kernel. The cluster caused so much interrupt events that there was not enough space in the network interrupt queue. This resulted the compiling process to be stopped. More can be read about this matter in chapter 4.2.

Another problem which had to do with the kernel building process is best explained by the following example;

> *As a kernel compile process is running, openMosix distributes the separate process to other node. The compiler polls if the process that just has been created still exists, it notices that the process isn't there anymore because it is still running at another node. This causes the dependencies to fail, and the compile process to be stopped immediately.*

We could not solve this problem in a proper way because it occurred in a random matter. Therefore we do not have any compiler benchmark results.

3.1.4 Synthetic benchmark

Unfortunately we could not perform this test due to the lack of time. Many problems occurred during the other tests.

3.1.5 Other benchmarks

We also tried to perform a MP3 encoding test. This resulted in a server node fully loaded with the encoding of an MP3. openMosix did not distribute it over the network. The program was using pthreads, a kind of thread that cannot be distributed using openMosix. Another program called ‘SMP mgzip‘[13] is a distributed version of ‘gzip‘, this application also used pthreads.

The above programs could not be used for testing. A big disappointment.

3.2 Reliability

The reliability of the cluster is a big issue according to our findings. We tested the cluster's behavior during a benchmark by removing nodes from the network. As we did, the cluster server node sometimes crashed and also corrupted our X-server. This meant that the X-server could no longer function. If a (sub) process is still running on the node, the server crashes because it cannot access the distributed process on a specific node anymore. The same story for suddenly disconnecting the power of a node.

The server had to be rebooted to be able to run the cluster again. A great disadvantage, this makes openMosix less interesting using it across the internet. Another reason why openMosix cannot be used across the internet is the lack of security precautions.

4 Discussion

Looking into the results we noticed several things. One thing we noticed is that openMosix does not spread threads to the other nodes. Another thing we noticed was that the performance of our cluster did not grow much after six nodes. Further investigation of this identified the problem to be in the latency of the communication channel. These two subjects are going to be explained in the next two subsections.

4.1 Threads and Processes

OpenMosix was unable to take a thread and place it on another node. Further investigation revealed that this was because of some key differences between threads and processes. When a thread is created it it does not have its own address space. Instead it uses the address space of the process that created the thread. This makes it possible for threads to read and write directly to any data that is accessible by the parent process. (see figure 4)

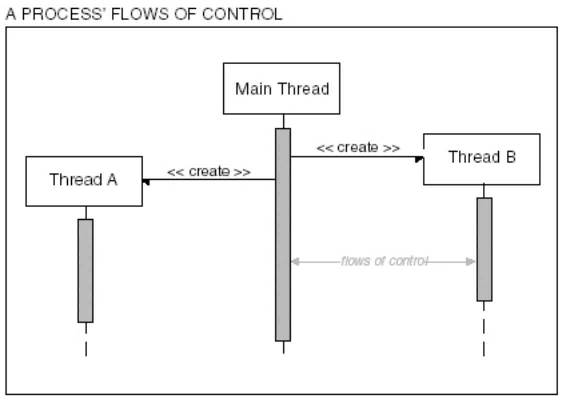

Figure 4: Process flow control

When a child Processes is created by a parent process it has its own address space and a copy of the data segment. If the child process changes data then it does not affect the data of the parent process. To send data between the parent and child process it needs to use interprocess communication mechanisms, such as fifos and pipes. What openMosix does is sending the entire address space to another node. This is of course nice if you run multiple programs or a program that spawns a lot of processes but it is not working with threads. This is something we saw with two multithreading file compression programs. A closer look at the source code of those programs it seemed that it takes a part of a file (input) and run a compression algorithm over it. The output was compressed data. If openMosix would have analyzed the program first it could have noticed this and would have been able to transfer the thread to another node. (a part of the address space that was necessary to allow to let the thread do its function). It would of course be a different situation if there was some data that constantly changed and all the threads needed to know this kind of information.

4.2 Latency

While conducting tests on the cluster we monitored the network closely. During one of our pre-tests to see if the cluster was working properly we executed the following command:

```
awk 'BEGIN for(i=0;i<10000;i++)for(j=0;j<10000;j++)print j;'
```

OpenMosix transfers this process to a node and returns the output j to the screen of the server. With this test we noticed a constant incoming network traffic of 55 Mbit. This was the only test that had such a high network load. All the other test stayed well below 10 Mbit. When we look at the povray results we noticed that performance did not longer grow very steadily by adding more then six nodes. Something was slowing us down. While running kernel compile benchmarks the network card gave us an encrypted answer:

```
Eth0: Too Much work in interrupt, status e401.
```

This error message meant that the maximum number of events to handle at each interrupt had been exceeded. The default value is 32 events per interrupt. This means that once an interrupt has been received the CPU had to handle a lot of events before it could continue to it original job.

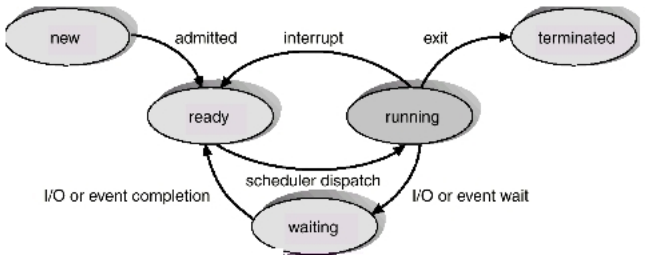

Figure 5: Process states

Every time information was being shared between the nodes a interrupt was being called. This means that the CPU has to stop what is is doing and place the current job back into the ready queue. See figure 5 for a graphical representation. When that is done it handles the interrupt request. The more data in an interrupt the more the CPU has to finish before it can continue with it's original job. In this interrupt request it also tells other nodes how busy it is, how much memory is being used etc. The more nodes there are on the network the more interrupts are being called. This means that the nodes are one point start to use more and more CPU cycles to communicate with each other and have less time to do the original work.

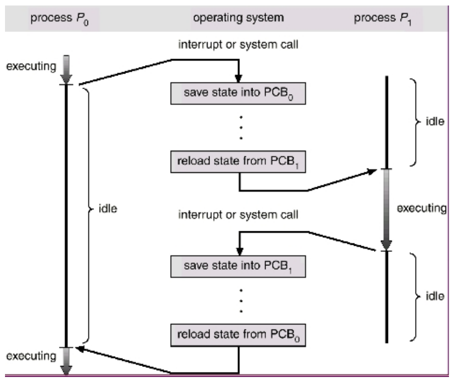

Figure 6: Process states of a interrupt request

Also if a process needs to access data thats on the server node it has to wait for an answer and can't do anything at that moment unless it has something else to work on. With some pre-testing we also noticed that if the amount of processes was large (90+) the time for the same test to complete was longer than that we used 48 processes. This probably could be explained because of CPU needs to do a lot of context switching and that the amount for communication between the nodes is larger because they are constantly trying to find a node that has less to do. an example of what happens when a CPU needs switch from one process to another is displayed in figure 6.

We can conclude that openMosix is a nice open source program for low budget institutions which want to start working with a cluster as long as applications are used which have the following criteria's:

- applications which create processes
- applications which will be started in larger numbers
- applications which store there results during it is running (before it stopped) are preferred.

Advantages

- Without special needed library's openMosix can distribute processes over the cluster
- Processes on the server will be distributed over the cluster as soon as a node has less load comparing to other nodes. This reduces the load of the server.
- openMosix runs on any Linux flavor after a kernel patch is applied
- As long as you use processors out of the same architecture, any configuration of your node is possible.

Disadvantages

- The package of openMosix does not come with any security facilities
- If a node cannot be reached either by a network failure or node failure, a great change will exist that the server will node will crash resulting completely restarting your cluster and loosing the work.
- The distribution of processes lasts relatively long, when starting a new job the server node will face extreme high load distributing the processes over the cluster. This can result in a overloaded server.
- Jobs that have failed cannot be reassigned to the cluster. The process can be considered as lost.
- After connecting six nodes, the cluster's performance does not grow much anymore.
- If two nodes are busy with a process, the process could be assigned to a node which is not busy. Since this node has the same configuration, there is no performance gain distributing the process to this node.

Using openMosix in public environments is asking for troubles, everyone can inject a job in the cluster or can jam a non-encrypted line.

Another issue we discovered during our tests was that when the server was booting and all the client nodes were already booted up, the server did not recognize the cluster nodes at all. The only solution for this was to remove the cache information from the server and let it rediscover the cluster nodes.

In general we can say that if openMosix is not being used in a mission-critical environment, it is a nice instrument to perform calculations on. As mentioned above, there are some great limitations which makes the program not very attractive to work with and rely on.

4.3 Summary

In short we can conclude based on experiment result that:

- Performance
  - Performance is not linear according to the number of nodes.
  - Applications that spawn more processes or more then one single applications will benefit from the use of openMosix
- Reliability
  - There is a chance that the complete cluster fails and needs to be restarted when a node failed.
- Network load

  How is the network load related to the number of nodes?

  - Network traffic increases by about 20% for every node added to the cluster.

4.4 Future work

If we have some more time in the future we would like to investigate ;

- Peformance of an Apache webserver on a openMosix cluster.
- DIEP - A chess simulation program which relies on integer calculations (Vincent Diepenveen)
- Rewriting some parts of openMosix to allow threaded applications to be distributed over the cluster.
- How to find the optimal amount of processes per node.

References

[1] openMosix - http://www.openmosix.org
[2] Seti at Home http://www.setiathome.com
[3] Introduction to openMosix, Daniel Robbins, Intel corporation http://www.intel.com/cd/ids/developer/asmo-na/eng/20449.htm
[4] Hyper-Threading Technology, Intel Corporation http://www.intel.com/technology/hyperthread/
[5] Beowulf cluster, Beowulf.org http://www.beowulf.org/
[6] Grendel cluster, http://www.nsc.liu.se/systems/cluster/grendel/
[7] Cluster Knoppix, http://bofh.be/clusterknoppix/
[8] PlumpOS, http://plumpos.sourceforge.net/
[9] Debian GNU Linux, http://www.debian.org
[10] KDE, Desktop environment, http://www.kde.org
[11] PovMosix, http://povmosix.sourceforge.net/
[12] Ying-Hung Chen, http://ying.yingternet.com/mosix/
[13] SMP mgzip, http://lemley.net/mgzip.html
[14] Operating System Concepts, Silberschatz, Galvin, Gagne ISBN: 0471262722 http://wiley.com/college/silbershatz
[15] Parallel and Distributed Programming using C++, Cameron Hughes, Tracey Hugnes
[16] MySQL Database server, http://www.mysql.com
[17] IPTraf, http://iptraf.seul.org/
[18] Unix Time, http://linux.about.com/library/cmd/blcmdl1_time.htm
[19] Building low-latency, high performance gameservers. http://www.democritos.it/events/openMosix/papers/conecta.pdf
[20] Democritos, www.democritos.it
[21] University of Kiev, http://www.cluster.kiev.ua/eng/
[22] LAPACK, NetLib http://www.netlib.org/lapack
Project members:

[23] Master education System and Network Engineering http://www.os3.nl
[24] University of Amsterdam http://www.uva.nl
[25] Maarten Michels mailto:mmichels@os3.nl
[26] Wouter Borremans mailto:wborremans@os3.nl
[27] Harris Sunyoto http://staff.science.uva.nl/˜sunyoto/e-mail.html

A Source shellscipt Encryption benchmark

```
#!/bin/bash

# **********************************************************
#  RSA key benchmark testscript v1.0
#   by Maarten Michels & Wouter Borremans
# **********************************************************

# unlock myself
echo 0 > /proc/self/lock

# Define the number of nodes which will participate in the cluster
nodes=14

#for ((run=4;run <=6;run++))
 #       do
echo ’Traffic logger started’

timestart=$(date +%Y-%m-%d-%H:%M:%S)

# Start iptraf on eth0 with a time-stamped log
iptraf -d eth0 -B -L
"/root/iptraf/iptraffic-encryption-$timestart-$nodes.log"

# Get the current process id of IPTraf
processid=‘ps ax | grep iptraf | grep -v grep | awk ’{print $1}’‘
DBS=‘./timertest 2>&1‘
timeend=$(date +%Y-%m-%d-%H:%M:%S)
kill -s USR2 $processid
echo ’Traffic logger stopped’
‘mysql -uroot  -D povray --exec="insert into data_encryption
(data,run,type,start,stop) values
(’$DBS’,’$run’,’$nodes’,’$timestart’,’$timeend’)"‘
```

B Source shellscipt starting RSA bechmark

```
# ************************************************************
#  Distkeygen startscript
# (requires RSA benchmark testscript)
#   by Maarten Michels & Wouter Borremans
# ************************************************************

#!/bin/sh
time ./distkeygen2 >/dev/null 2>/dev/null
```