EASILY DEPLOYABLE CLUSTERING SYSTEMS

Adrian Neațu, Dan Tuşaliu, Marius Dumitraşcu

University of Craiova, Faculty of Automation, Computers and Electronics,
Computer and Communications Engineering Department,

Abstract: 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 machine, sharing threads, processes and jobs. Computers (nodes) in clusters are interconnected by a network, the network is – besides the nodes – the most important part of cluster; thus the network is the limit of the cluster, the higher the bandwidth the higher the performance of the cluster will be.

Keywords: cluster, Linux, inexpensive components, openMosix.

1. INTRODUCTION

This paper is the result of the research work (both theoretical and experimental) in cluster computing domain carried out by the authors. This project focuses on the performance, reliability and characteristics of clusters.

1.1 About clusters

According to the definition given by an online dictionary:
cluster; bunch, clump, cluster, clustering – (a grouping of a number of similar things).
When designing a cluster as a collection of computers in a network area for parallel or distributed systems, in general there are a few goals to achieve:
- 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;
- Complete transparency – single entry point.
Basically there are 3 types of clusters:
- *Fail-Over Clusters*, ***FO***, consist of 2 ore more network connected computers with a separate heartbeat connection between the 2 hosts. The heartbeat connection between the 2 machines is being used to monitor whether all the services are still in use: as soon as a service on one machine breaks down the other machines try to take over.
- *Load-Balancing Clusters*, ***LB***, where the concept is that when a request for say a web-server comes in, the cluster checks which machine is the least busy (most available) and then sends the request to that machine. Actually most of the times a Load-balancing cluster is also a Fail-over cluster but with the extra load balancing functionality and often with more nodes.
- *High Performance Computing Clusters*, ***HPC***, in which case the machines are being configured specially to give data centres that require extreme performance what they need. Beowufl systems have been developed especially to give research facilities the computing speed they need. These kind of clusters also have some load-balancing features; they try to spread different processes to more machines in order to gain performance. But what it mainly comes down is being parallelized and that routines can be ran separately will spread on different machines instead of having to wait till they get done one after another.

The most deployed types of clusters are probably the Fail-Over Clusters and the Load-Balancing Clusters, but for a high performance, High Performance Computing Clusters are the most recommended.
Most common known examples of LB and FO clusters are web farms, databases and firewalls. People want to have 99, 99% uptime for their services, the internet is always up and running.
The most used technique to build up a cluster is SSI (Single System Image). SSI is the illusion created by the software and hardware that represents a collection of computing resources as one, more while resources. 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:
- *High Throughput Clusters*, ***HT*** – primary used for serial applications;
- *High Performance Computing Clusters*, ***HPC*** – used in the computational science
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 in order to use the machines effectively.
There are different ways of doing parallel processing:
- *(Non) Uniform Memory Access*, ***(N) UMA, machines*** for example have shared access to the memory where they can execute their code. In the Linux kernel there is a NUMA implementation that varies the memory access times for different regions of memory. It then is the kernel's task to use the memory that is the closest to the CPU it is used.
- *Distributed Shared Memory*, ***DSM***, has been implemented in both software and hardware, the concept is to provide an abstraction layer for physically distributed memory.
- ***PVM*** and ***MPI,*** tools that are most commonly being used when people talk about GNU/Linux based Beowulfs. *Parallel Virtual Machine* is quite often being used as a tool to create Beowulf. PVM lives in user space so no special kernel modifications are required, each user with enough rights can run PVM. *Message Passing Interface* is the open standard specification for message passing libraries. ***MPICH*** is one of the most used implementation of MPI based on the free reference implementation of the libraries.

1.2 About openMosix

openMosix is a Linux kernel extension for single-system image clustering (SSI). This kernel extension turns a network of ordinary computers into a supercomputer for Linux applications. Once one 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 distribution. **openMosix** uses adaptive load balancing techniques, processes that run on a node can transparently be distributed to other nodes. Due to the complete transparency of **openMosix**, a process 'thinks' that it is running locally. **openMosix** turns multiple Linux hosts into one large *Symmetric Multi Processor* (***SMP***). Real SMP systems with two more physical processors can exchange large amounts of data, in practice this means that SMP system are much faster. With **openMosix** the speed at which the nodes can exchange data is limited to the speed of the Local Area Network (LAN) connection. Using a high bandwidth connection will increase the effectiveness of **openMosix** cluster.
Another 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 available on Intel and AMD last generation processors. Using this technique enables one to increase the performance of a node. The node can now handle multiple cooperating threads that cannot be separated and distributed among **openMosix** nodes.

2. HARDWARE BEING USED

All experimental work has been carried out in one of the laboratories within the Faculty of Automation, Computers and Electronics.
Unfortunately, because of some circumstances, there was not possible to use a modern laboratory, with 20 or 30 nodes and high-performance network equipment; all available equipment is quite old, but this could actually prove as an advantage, all this paper intends is to prove that such experiments can be conducted, and later it will be continued with programme development.
For the conducted experiments, the computers only needed an optical drive and enough RAM memory.
In the tests carried out, the following low performance equipments were used:
- One Intel Pentium III, 500MHz, 512MB RAM, named mlc11;
- One dual-processor Intel Pentium II, 350MHz, 256MB RAM, named mlc12;
- One dual-processor Intel Pentium II, 350MHz, 512MB RAM, named mlc13;
- 10/100 network cards for each station

The computers were coupled each other using one 8 Port Fast Ethernet Switch offering 10/100Mbit Ethernet switched network to each unit.
ClusterKnoppix is a Linux operating system which runs completely from CDROM, using computer's RAM instead of HDD.

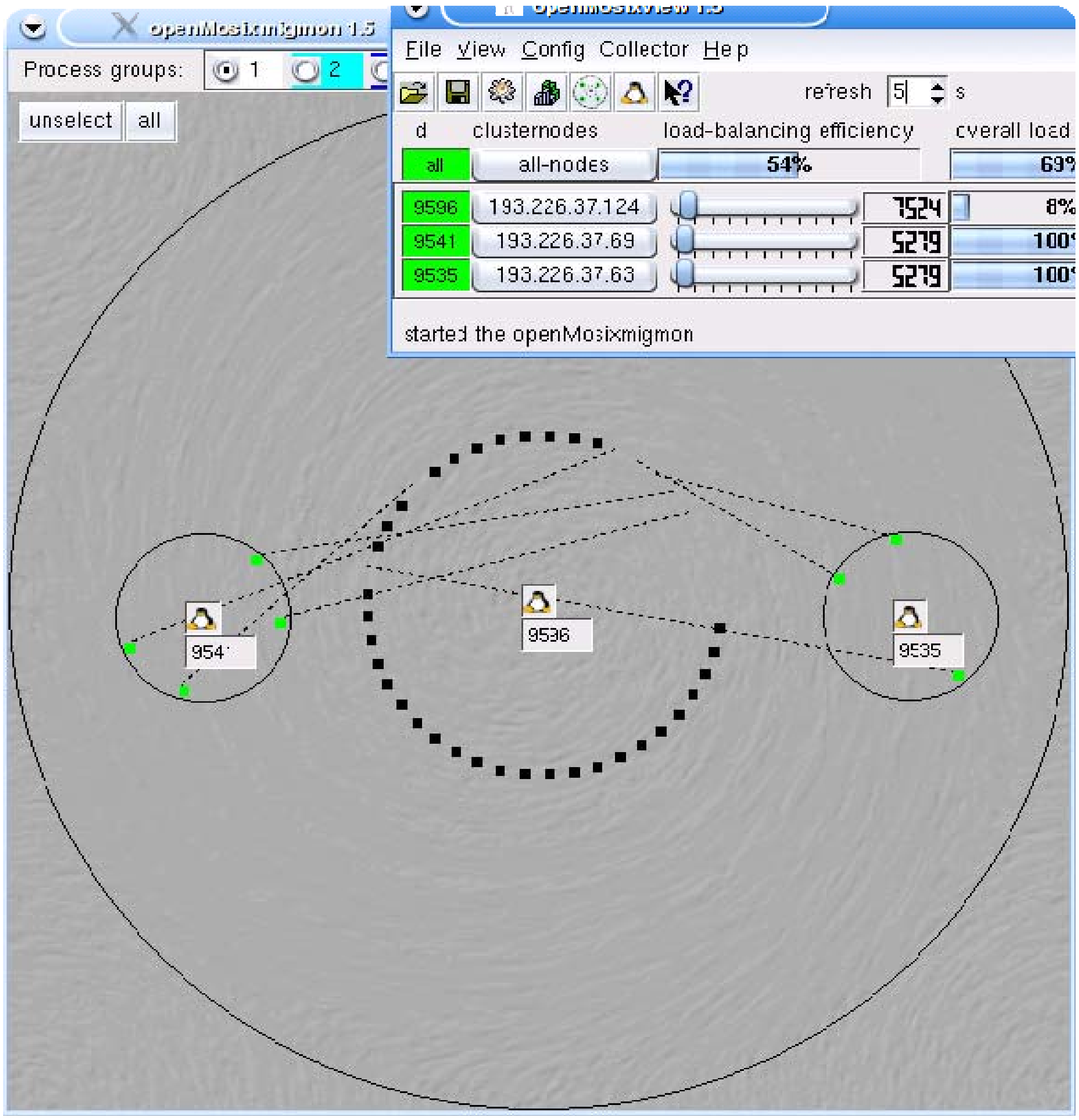


Fig. 1. Cluster structure.

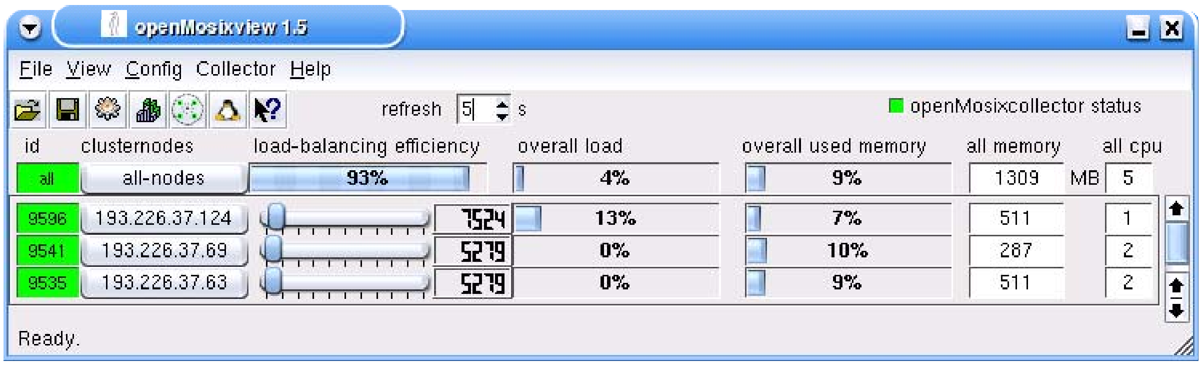
When booting up, Knoppix automatically detects all hardware and starts up a graphical user interface (KDE). ClusterKnoppix already has an **openMosix** modified kernel built in. The package comes with several **openMosix** utilities to smoothly build up a cluster.

3. EXPERIMENTS

3.1. Methodology

The method of building the cluster and experimenting with it is straightforward. There is no need of a central point – a server – for the cluster. In this case, there exists a DHCP server which runs on one of the departments web servers. The first node is powered on. By default, it searches for an available IP address. The program is run and the execution time is recorded. Next, the second node is powered on and receives an IP address. In the same time, it searches for any other available node running the same type of software (SO + **openMosix**). The two nodes then "become" aware of one another, starting communication and thus forming the cluster. The experiment is then rerun, recording the new result as well. The same procedure applies foe all newly added nodes.

Before trying to experiment anything, there was the need of knowing which kind of applications would run on the cluster and would give a good idea of the cluster's performance. As all information provided by the **openMosix** project community implies, there is no possibility to run threaded applications on the cluster given the 2.4 kernel, thus simplifying the search for programs to be run on the desired cluster.



refresh 5 s

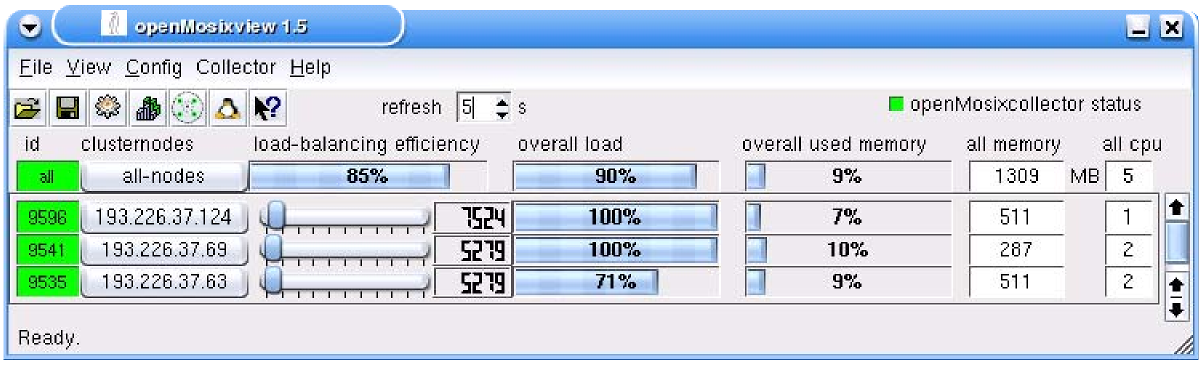
id	clusternodes	load-balancing efficiency		overall load	overall used memory	all memory		all cpu
all	all-nodes	93%		4%	9%	1309	MB	5
9596	193.226.37.124		7524	13%	7%	511		1
9541	193.226.37.69		5279	0%	10%	287		2
9535	193.226.37.63		5279	0%	9%	511		2

Ready.

Fig. 2. Initial state of cluster (initial LOAD) – when comprising 3 nodes

3.2. Generation of security keys

First step in our analysis was to run a program that would generate 8000 RSA Key Pairs with 1024 bits key length each. The program creates 4 child processes by means of the fork function; the entire program can be distributed to other stations, if any exist.



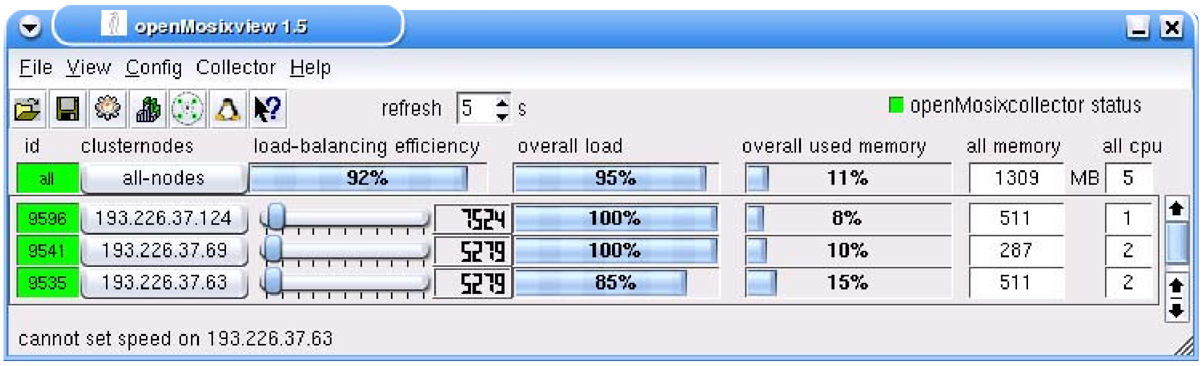
refresh 5 s

id	clusternodes	load-balancing efficiency		overall load	overall used memory	all memory		all cpu
all	all-nodes	85%		90%	9%	1309	MB	5
9596	193.226.37.124		7524	100%	7%	511		1
9541	193.226.37.69		5279	100%	10%	287		2
9535	193.226.37.63		5279	71%	9%	511		2

Ready.

Fig. 3. First test (cluster containing 3 nodes)

Each of the processes will generate 2000 keys (first process will generate keys from index 0000-1999, the second one will generate from 2000-3999 and so on).



refresh 5 s

id	clusternodes	load-balancing efficiency		overall load	overall used memory	all memory		all cpu
all	all-nodes	92%		95%	11%	1309	MB	5
9596	193.226.37.124		7524	100%	8%	511		1
9541	193.226.37.69		5279	100%	10%	287		2
9535	193.226.37.63		5279	85%	15%	511		2

cannot set speed on 193.226.37.63

Fig. 4. Second test

3.3. Compiling experiment

The second and more important experiment that was conducted was compiling of a program. The idea was to prove that such a cluster would be suited for fast compiling of software in general.

The software program chosen for this experiment is a video player, MPlayer.
For the compiling process to be split into more processes, the “make” was run with the “–j” option, followed by the number of processes one needs to create.

4. TEST RESULTS

4.1. Generation of security keys

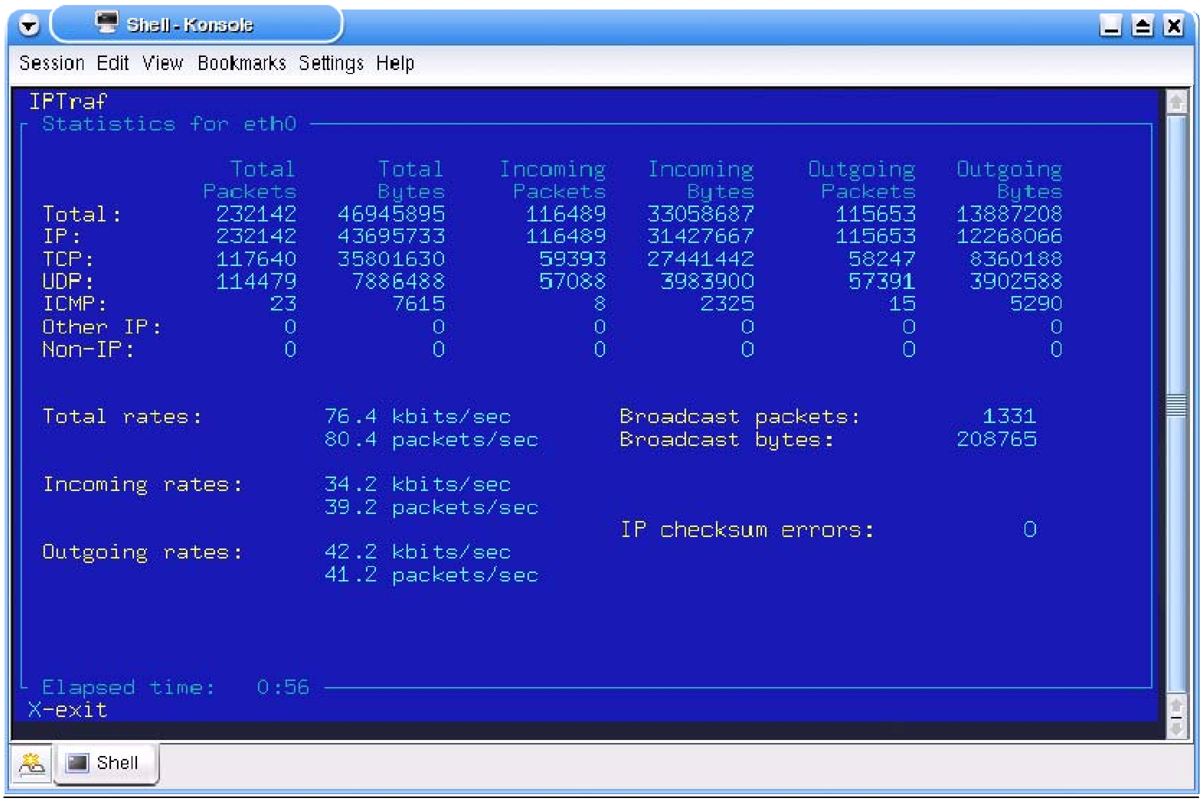
The results are as expected.
First the program was compiled and run on the mlc11 machine (which has only one processor). The program took 38 minutes to complete generation of the 8000 keys.
Secondly, the mlc12 machine was powered on and added into the cluster. It is a dual processor computer; the result was immediate. The time needed for generating the keys decreased almost by a factor of 2, to only 20 minutes.
The next step consisted in adding the third machine, mlc13, which is also a dual processor one. At this time, the cluster consisted of 5 processors and 1309 megabytes of main memory.

Total time of execution was of about 15 minutes.

This is the result as shown in the linux console:

```
root@ttyp0[rsa]# date; ./DistKeyGen; date;
Wed Jun  8 13:42:57 EDT 2005
CHILD: I'm finished here! 2350
CHILD: I'm finished here! 2347
CHILD: I'm finished here! 2952
CHILD: I'm finished here! 3633
PARENT: About to quit!
Wed Jun  8 13:57:44 EDT 2005
root@ttyp0[rsa]#
```

Given below is a screenshot presenting the traffic generated by the communication among the nodes of the cluster.



```
IPTraf
 Statistics for eth0

                Total      Total   Incoming   Incoming   Outgoing   Outgoing
              Packets      Bytes    Packets      Bytes    Packets      Bytes
 Total:        232142   46945895     116489   33058687     115653   13887208
 IP:           232142   43695733     116489   31427667     115653   12268066
 TCP:          117640   35801630      59393   27441442      58247    8360188
 UDP:          114479    7886488      57088    3983900      57391    3902588
 ICMP:             23       7615          8       2325         15       5290
 Other IP:          0          0          0          0          0          0
 Non-IP:            0          0          0          0          0          0

 Total rates:          76.4 kbits/sec         Broadcast packets:      1331
                       80.4 packets/sec       Broadcast bytes:      208765

 Incoming rates:       34.2 kbits/sec
                       39.2 packets/sec
                                              IP checksum errors:        0
 Outgoing rates:       42.2 kbits/sec
                       41.2 packets/sec

 Elapsed time:   0:56
X-exit
```

Fig. 5. Network traffic.

4.2. Compiling experiment

Firstly, the software was compiled on the mlc11 machine (the one with only one processor). The “make” step took around 35 minutes to complete.
Then, the mlc12 (a dual processor one) machine was added to the cluster. The compiling time decreased to 17 minutes. The observation is that the two computers are not equivalent as performance, so the result will be relative.
Finally, the third machine was added to the cluster, mlc13, which is also a dual processor one. At this time, the cluster consisted of 5 processors and 1309 megabytes of main memory.

Total time of execution was of about 10 minutes.
The results were expected. The more computing power, the smaller the time it takes to compile.
During the compiling of the program, for two nodes, the value of the network traffic monitored for the network interface of one of the two nodes was of about 50kbits/sec, with peaks reaching even 1000kbits/sec. The values increased for three nodes, to an average of 300kbits/sec

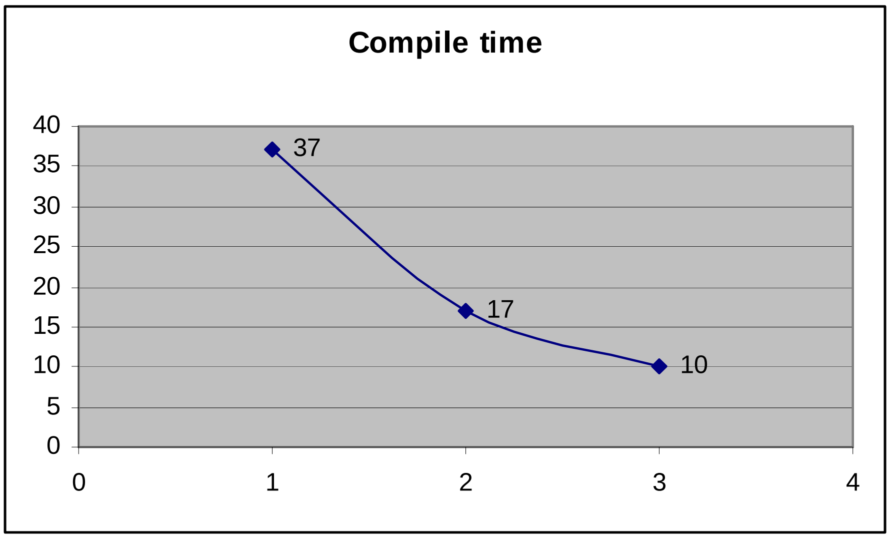


Fig. 6. Compile time graphic

Obviously, the higher the number of nodes, the higher the traffic is. At one point, the necessary bound width would no longer be satisfied, so there is a limit in what concerns the number of nodes. In other words, if the cluster is formed of too many nodes, there would not be an increase in performance, if there wasn't available a high performance network to sustain the inner node communication.
In this case, given a network of 100 megabits/sec, the optimal number of computers is smaller than 10 (around 6 or 7). If there is the need for a larger cluster, other types of network equipment should be considered – the first step being the implementation of a gigabit LAN.

5. CONCLUSIONS

Given a cluster, in general, its advantages are obvious. There are a large number of possible applications that could benefit from the existence of an implemented cluster system.
The solution chosen to rapidly deploy a cluster has its advantages, and there can be mentioned the simplicity and the type of hardware needed (no special type of processor is needed). A major disadvantage, for the moment, is that the applications making use of threads instead of processes cannot be distributed over the cluster.
The work carried out so far - the experiments conducted – is intended to be continued by the authors in the near future. The domain of distributed systems and moreover the development of distributed applications to benefit from their existence will become more and more interesting. Solutions are available for easy implementations – if there is considered a laboratory comprising a computer network that is not used during the night time, this could be used until the morning for such computations –

REFERENCES

[1] openMosix, http://www.openmosix.org
[2] Cluster Knoppix, http://bofh.be/clusterknoppix/
[3] Ying-Hung Chen,
http://ying.yingternet.com/mosix/
[4] MPlayer, http://www.mplayerhq.hu