

# Turning a group of independent GNU/Linux workstations into an (Open)Mosix cluster in an untrusted networking environment and surviving the experience. . .

Giacomo Mulas\*

28 November 2002

## Abstract

To have a more or less independent cluster of GNU/Linux workstations is an increasingly common situation in many research environments. They are normally subjected to widely different workloads depending on users and uses, and, in most cases, a large fraction of the potential computing power they offer goes unused. OpenMosix can provide a transparent way to make these resources available, but its implementation on an untrusted network poses serious security problems, which can be overcome by the use of an underlying authentication layer.

## 1 Introduction

It is increasingly common, in research environments, to have a more or less loosely interconnected cluster of “personal” GNU/Linux workstations which are used by different researchers for widely different purposes and with widely different demands. Such a configuration is much more cost effective than the, once more common, configuration based on few servers providing computing resources and a large number of more or less dumb X terminals just used to access them. The aggregated CPU power of a group of a few tens of commodity PCs can be quite respectable and, in many cases, goes largely unnoticed and, more importantly, unused. This is due to the widely different purposes they are used for, which depend on the research needs of single users. I will present here a test case of an heterogeneous cluster of 4 GNU/Linux computers, used by just as an heterogeneous group of PhD and undergraduate students of the AstroChemistry Group of the Astronomical Observatory of Cagliari.

---

\*INAF - Osservatorio Astronomico di Cagliari

<i>Job</i>	<i>Used by</i>	<i>CPU intensive</i>	<i>I/O intensive</i>	<i>Interactive</i>
ab initio molecular structure calculations	Giuliano	yes	medium	no
models of molecular levels population in astrophysical environments	Giuliano, Silvia	yes	yes	no
models of fluorescence by small particles	Giuliano	yes	no	no
chemical reaction networks	Silvia	yes	no	no
data analysis of astronomical observations	Silvia, Carla, Marianna	no	medium	yes
development of physical models	Giuliano, Silvia	medium	medium	yes

Table 1: Job categories and demands

## 2 Purposes and needs

Table 1 summarizes the main purposes the different students use their respective computers for, along with a rough characterisation of their demands. There is an obvious distribution of different demands, which is mostly apparent if one compares what is run by each student:

- Carla and Marianna use their computers mainly for data analysis, which is very interactive; they use sparse bursts of CPU time, and need their computers to be responsive to work effectively;
- Giuliano develops and runs all sorts of computationally expensive models; he only needs his computer to be responsive when developing/debugging, otherwise his jobs are non-interactive and last from tens of minutes to many days;
- Silvia is somewhat in-between the above two extremes, in that she both does interactive data analysis and some heavy duty number crunching.

What results is that Giuliano’s computer is usually fully loaded full time, Silvia’s computer just slightly less, Carla’s and Marianna’s computers are almost always idle, with occasional usage bursts.

This is clearly a very inefficient way to use the available resources. As a first, rough step, one may think to have heavy-duty users remotely log into other computers and running processes directly on them. However, this is clearly a

very inflexible solution: once a three day job is started on a given computer, it will run until its end, even if the computer it runs on becomes highly loaded (and others possibly idle) in the meantime. This is where (Open)Mosix comes to the rescue.

### 3 Pooling computing resources with (Open)Mosix

(Open)Mosix provides efficient, flexible and yet transparent process migration and load balancing within an x86 based GNU/Linux cluster. In essence, it pools together the computing power provided by a number of workstations and allows users to seamlessly draw from it: they just run their jobs and the system takes care to dynamically move them across nodes in order to have the whole cluster perform most efficiently. In this way, Giuliano's and Silvia's heavy number-crunching processes would distribute over our 4 nodes, making optimal use of them.

#### 3.1 Security implications

Therefore, since (Open)Mosix<sup>1</sup> clearly seemed to fit our bill, a couple of years ago I decided to move on and implement it on our cluster. However, the 4 workstations to be included are connected to the network of the Physics Department of the University of Cagliari which, as is often the case for academic networks, despite the efforts of the quite capable network administrators, for "historical" reasons includes a large sample of self-administered (some of them poorly) computers. In a word, the network the workstations are connected to is untrusted. Now, since (Open)Mosix does not yet support any sort of authentication apart from the IP numbers of the nodes which are supposed to be part of the cluster, this is quite a real issue. Any computer capable of, even temporarily, "stealing" the IP number of a node would be able to convince the others to do unpleasant things. As a particularly nasty example, if Mosix File System (MFS) support is enabled in the cluster (as it should, for efficiency reasons), a malicious hacker could simply issue a "rm -rf /mfs/1/\*" command as root from his computer, pretending to be node number 2, and your node number 1 would happily wipe away its entire filesystem. . . While I never heard of any actual case of such dreadful happenings with (Open)Mosix clusters, I decided I preferred to sleep at night, therefore would either find a way around this risk or not run (Open)Mosix at all.

Luckily, the communication between (Open)Mosix nodes takes place above the IP networking layer, therefore the solution is to create a Virtual Private Network (VPN) at or below the IP layer and then run (Open)Mosix on top of it.

---

<sup>1</sup>actually Mosix, OpenMosix did not exist yet at the time

## 3.2 IPsec and (Open)Mosix

A relatively simple way to secure all IP traffic between two computers is to implement an IPsec connection between them. Currently, there are at least two relatively mature options to implement IPsec support in 2.4.x Linux kernels, namely the FreeSWAN and the USAGI project. Since I was already familiar with FreeSWAN, I configured the 4 workstations to only talk via IPsec to one another. Since security was most important for me, I chose to use full encapsulation (ESP) instead of just authentication (AH). Finally, for more safety, I also set up a relatively tight firewalling script, to make really sure that no (Open)Mosix traffic could get through unauthenticated (and unencrypted).

To be completely honest, I have to say that I somewhat cheated here: I had *already* set up tight firewalling scripts and IPsec on the 4 workstations, in order to safely provide, among them, services as NFS, NIS and such. Therefore, (Open)Mosix could simply benefit of this existing structure with no additional work.

## 3.3 The price of security

Very few good things in life are free. While quite a bit of Open Source Software, including OpenMosix, is indeed free, this is not the case with the security provided by IPsec. The price to pay for it comes in term of CPU load, increased latency times and (possibly) reduced bandwidth. Encrypting and decrypting all network traffic on the fly *is* expensive. Before the introduction of the relatively new Rjindael algorithm in IPsec, 3DES encryption would saturate a 800 MHz Pentium 3 CPU at more or less 20 Mbits/s. Nowadays, present day commodity PCs, using Rjindael, can saturate a 100 Mbits/s Fast Ethernet Network Interface Card (NIC), still leaving some CPU power available<sup>2</sup>. Even with these improvements, network traffic is clearly much more expensive than it would be without IPsec, and furthermore the additional load takes place in kernel space, which makes it yet a bit less gentle with respect to other tasks. In principle, one can configure the current FreeSWAN implementation to just do authentication via Rjindael (or another algorithm) and then use the “null” cypher to encrypt/decrypt network traffic, but I did not test it, since there is some traffic (e. g. NIS password authentication exchanges) which I do not want to travel in cleartext.

For the above reasons, while in the more common setup of similar boxes directly talking via plain IP tuning may help *improve* performance but is not really needed, it becomes absolutely *mandatory* in a setup running (Open)Mosix over IPsec. Migrating a process with a large memory footprint is much more expensive in such an environment, as well as doing a large amount of I/O on the wrong node, and the system *must* be aware of this in order to make effective choices.

---

<sup>2</sup>There are NICs which include dedicated processors to offload encryption and decryption, but they are not much supported yet with FreeSWAN, to the best of my knowledge

## 4 Performance and caveats

After all appropriate tuning, the current situation on our small cluster has seen a large improvement in terms of efficiency. CPU-intensive jobs now consistently make use of a large fraction of the available computing power, dynamically migrating in order to optimize the load. When necessary, CPU hogs can be easily sent home, should they make the guest node unresponsive. Custom scripts handle locking and unlocking nodes for migration depending on the time of the day. All of this is completely transparent to users, who *need not* do anything particular to reap the benefits of this environment, but *can* optimize things, if they want, by running some processes with specifically tailored migrations options.

It looks like an ideal situation, so why doesn't everybody with more than one x86 GNU/Linux workstation build an OpenMosix cluster? There still are some issues left.

1. CPU hogs *must* be educated to be nice to other users: since the cluster behaves, by and large, like a single SMP computer, a single user can overload it. In this respect, users must run unlocked jobs with high nice values, in order to avoid bringing the whole cluster to a grinding halt.
2. I/O intensive processes make the node they run on unresponsive, regardless of the nice value they are run with.
3. processes which have a memory footprint comparable to, or larger than the RAM installed on the node they run on, make it unresponsive. There is currently no way to specify that a process may be allowed to migrate to a restricted subset of the cluster.
4. there is no simple way to restrict the use of (some part of) the (Open)Mosix cluster to only some users. You don't want the average first-year student who was granted an account to read email and browse the web to kill your cluster encoding mp3 files. Unfortunately, a process can always unlock itself if /proc is mounted.

While the first point attains to social engineering and the second is largely a Linux kernel issue, not directly related to (Open)Mosix, and will probably be improved quite a bit in the upcoming 2.6.x release of the kernel, the others are, to some extent, real drawbacks which may hinder a wider adoption of (Open)Mosix by institutes which have a large number of x86 based GNU/Linux workstations whose computing power goes largely wasted.

## 5 “Shaping” computing resources?

Richard Feynman used to repeat “the same equations have the same solutions”. In a sense, this may be true also for (Open)Mosix. It is tempting to compare a computer cluster to a network: independent workstations can be seen as a circuit

switched network, and an (Open)Mosix cluster as a packet switched network. In the former case, bandwidth (computing power) is rigidly divided in timeslices (separate workstations), while in the latter case it is pooled. Circuit switching is inefficient, as time slices can go wasted, but provides guaranteed bandwidth and latency times. On the other hand, packet switching maximises bandwidth usage, but a single user can cause congestion, killing the performance of the whole network. These problems are largely solved by implementing Quality of Service (QoS) and traffic shaping, which allow to flexibly guarantee low latency times and/or a given bandwidth to some uses, while still allowing it all to be used by others when unneeded. There is no such thing as “CPU time shaping”, but perhaps there should be. . . Maybe something for Linux kernel 2.8.x?