Distributed software platforms for rehabilitating obsolete hardware

Ruggero Russo
Department of Computer Science
La Sapienza University of Rome
Email: ruggero.russo@email.it

Davide Lamanna
Ingegneria Senza Frontiere
Rome, Italy
Email: davide.lamanna@isf-roma.org

Roberto Baldoni
Department of Computer Science
La Sapienza University of Rome
Email: baldoni@dis.uniroma1.it

Abstract: Diffusion of ICTs created the issue of an enormus quantity of old computers to be discarded (e-Waste). Sustainable dismantle is becoming a global enviromental emergency. In this scenario, Trashware movement is spreading worldwide to give to computers the correct time of obsolescence and to face the ecological problem related to e-waste. Workgroups aim to profitably reuse discarded computers as an alternative to dismantling them. The spread of this phenomenon is deeply related to the Open Source and Free Software movements. The purpose of this piece of research is to combine Trashware to Clustering, in order to verify if further optimisations are possible. Experimentation was conducted on clusters of old machines and results are hereby presented.

Keywords: Trashware, Clustering, Recycling, Environmental issues, Digital Divide.

I. Introduction

The tremedous diffusion of Information and Communication Technologies (ICTs) has had as a consequence that nowadays a huge quantity of computers are widespread around the world. This is particularly evident in that part of the world, referred to as Global North, where ICTs are massively developed and adopted. Computers are often replaced with a high frequency: 12-18 months for business users, 4 years for home users [6]. As a result, every year 150 millions of computers are discarded in the world. This situation has led to the urgent problem of managing the waste of obsolete ICTs (e-Waste).

As a consequence of Moore's law[1], power of microprocessors is expected to duplicate every one and a half years. Such an extimation comes, of course, from an empirical observation and does not answer the basic question: why are computers replaced? It is a fact that software systems become more and more complex with time, often providing new features and functionalities which do not address real user needs. This is particularly true for "standard users", i.e. users who only need simple office applications and navigation tools and who are by far the most common. Workers of public administrations are a good example. Moreover, the common practice of not distributing software source-code prevents users from almost any possibility of optimising their systems.

It can be, hence, said that the main reason why hardware needs to have higher and higher performance is that software is, often uselessly, more and more resource consuming and not accessible. Software development relies on hardware development and viceversa. Such a vicious circle causes a damage for users, who are obliged to buy new hardware even if they do not really need new software, thus generating what can be defined "ICTs consumism".

Unfortunately, discarding computers does not come for free. Often computers end up in dumps which are not suitable for disposing of them. They release polluting substances, such as mercury, lead, hexavalent chromium [4], resulting in a real environmental emergency. Chemical draining and recycling are not efficient processes: only the 10% of materials can be recovered, by means of not favourable expenses [6], [13]. Laws exist which regulate disposition of computer waste[2], though they are often disregarded, so that 90% of obsolete computers end up in dumps.

Despite of this, the demand of computers in the world remains constantly high and in most cases increases [15]. For example, less developed countries are trying to fill the big gap existing between Global North and Global South in terms of ICTs diffusion, the so called Digital Divide [12]. If, from one side, ICTs are essential for human develpment, from the other side, the present computer production and software development represent a totally unsustainable process.

For these reasons, the authours believe that rehabilitating computers, which are erroneously considered obsolete, for redistributing them is a topic of high relevance. The objective of the piece of research hereby presented is an evaluation of the technical possibility of optimising peformances of discarded computers through clustering (Section III) and open source software. In particular, focus is given to the possibility of making old boxes usable by "standard users". This is relevant for the social purposes of Ingegneria Senza Frontiere (Engineering Without Frontiers), a non-profit organisation that is involved in international cooperation, appropriate technologies and education for development.

[1]Number of transistors per square inch on integrated circuits doubles every 1.5 years.

[2]Directive 2003/108/EC of the european parliament and of the council of 8 December 2003.

II. TRASHWARE: DEFINITION AND RELATED WORK

The term Trashware was first used in the year 2000 by Golem, a Linux User Group of Empoli, Italy. It merges the words Trashing and Hardware and refers to the profitable reuse of discarded computers as an alternative to dismantling them. A common inducement for who works on Trashware today, in Italy and in the world, is the ecological question. As a matter of fact, doing Trashware means working on a sustainable adoption of hardware resources which are still effectively usable and are instead destined to dumps. The social purpose is another shared motivation. Boxes are collected from firms, public bodies or private citizens, in order to be refurbished and donated to non-profit organisations, international cooperation and solidarity projects.

Trashware is indissolubly related to Open Source, as this represents an indispensable tool for it. Open Source enables full control on hardware/software optimisation. Besides, software development is supported by a community that can provide online help, frequent updates and bug fixes. Costs are pull down, as there is no need to pay licenses and hardware is basically for free. This is particularly relevant if social purpose and developing countries are concerned. Not to mention that Free Software garanties independence from foreign actors and big firms, which actually favour technical obsolescence by proposing and adopting proprietary software.

Among Trashware initiatives, the work of Golem, a worldwide pioneer in the field, is certainly remarkable. They provided the idea, technical bases and guidelines for Trashware projects [7]. Since year 2000, they have carried on projects supporting voluntary organisations, both locally and in developing countries.

Another interesting workgroup is Lazarus Project, an international initiative promoted by EdOsNet.org pursuing reuse of discarded hardware in primary and secondary schools. They are involved in supporting Free Software, open formats and the culture of reutilization.

Also Prodigi, an association based in Rome, works in the field. Prodigi implements strategies to fight digital exclusion. In 2003, they carried on a project in Tunisia [8], where they set up two computer labs based on Free Software and Trashware and held computer courses.

The year before, Ingegneria Senza Frontiere (ISF) implemented a similar project in Kosovo [9], [10]. ISF believes that Trashware can play an important role in international cooperation to fight Digital Divide. For this reason, a specific workgroup was created in 2003 in collaboration with the Department of Computer Science of La Sapienza University of Rome (DIS). Aspects of Trashware were studied in an original way, i.e. by making use of clustering, in order to optimise resources to the extreme.

III. CLUSTERING

Clustering systems are used to integrate resourses belonging to two or more computers (that could othewise work separately), working together to increase both computational power and reliability of the entire system. The basic idea of clustering is distributing load among the computers in the cluster (cluster nodes), so that free resources of a node become available to the whole system. Cluster dimensions can grow by adding machines. Speed and computation capacity of a cluster depend on the ones of its nodes and on the speed of the network.

A clustering system should use the available hardware at best, even in dynamically variable conditions. This represents a real challenge when dealing with an etherogeneus cluster (built with unhomogeneous hardware) or when setup changes in an unpredictable way, because of the addition or removal of nodes. There are three kinds of clusters: Fail-over, Load-balancing and High Performance Computing. A Fail-over Cluster consists of two or more connected computers. Their work is continuosly monitored so that when one host breaks, another replaces it. A load-balancing Cluster uses a balancing algorithm to migrate each job to the less loaded node. A HPC Cluster is a system in which computers are set up to provide a data processing center with high performances, which are obtained by distributing processes between different machines and parallelizing their execution. To implement the clusters used in the present piece of work, two very different systems were chosen:**OpenMosix** and **LTSP**.

A. OpenMosix

OpenMosix [2], [3] is an Open Source software used to turn a GNU/Linux computer network in a cluster. It authomatically balances the load between different nodes that can join or leave the system, without service disruption. Load is distributed among nodes according to their CPU speed. OpenMosix is a Kernel Linux patch, and hence applications, files and resources of a network of computers keep working without changes. User do not notice differences between a single Linux system and an OpenMosix one; from his perspective, the entire cluster works as a normal, very fast GNU/Linux system. OpenMosix has got a decentralized control, i.e. each node acts as it was an indipendent system and it takes a decision based on a partial knowledge of other nodes. This provides the possibility to add and remove nodes trasparently, thus getting high scalability.

The load-balancing and mamory ushering algorithm are used to manage available resources. The former lets cluster efficiently split load among nodes, by forcing processes to migrate from heavy loaded nodes to computers with free resources. The latter balances memory load of cluster nodes. It prevents performance decrease which is caused by a lack of free memory. When a node starts to use swap, the algorithm tries to migrate its processes towards another node with free memory.

OpenMosix has got a hard limitation: it does not allow migration of shared memory processes. Consequently, applications using shared memory (basically the most common end-user programs) do not gain benefits from its use. This fact constituted a remarkable obstacle for the aims of a research very much based on end-user applications like the present one. In fact, OpenMosix was designed for HPC, where shared memory process are not necessarily relevant.

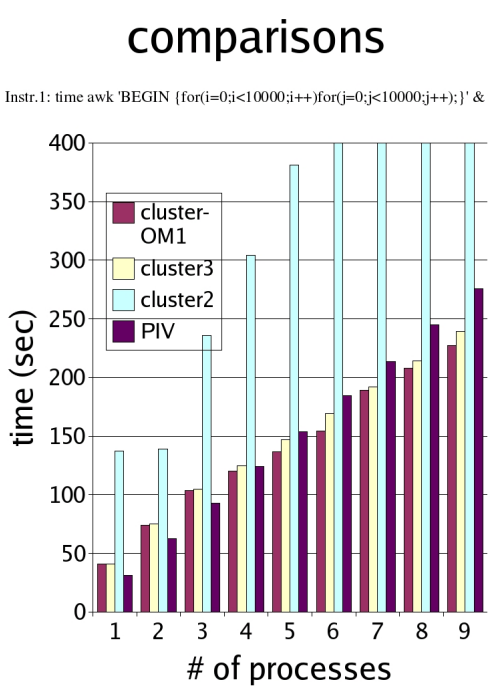

Fig. 1. Comparison between the ”poor” cluster and a modern architecture (PIV), varying the number of nodes and of processes.

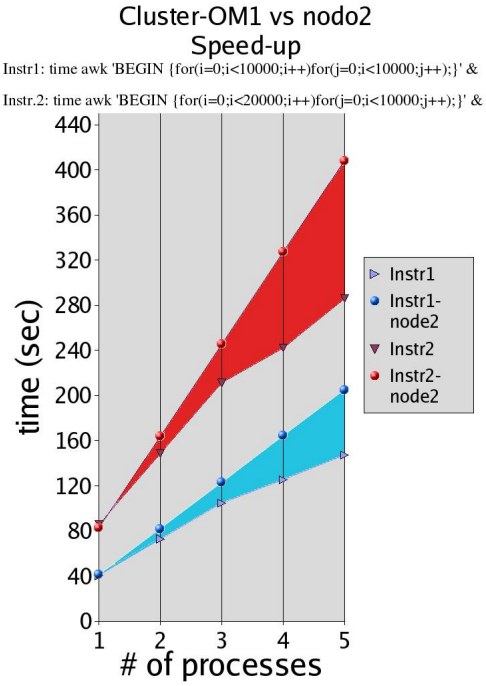

Fig. 2. Speed up reached by the four-node cluster in comparison to node-2.

B. MigSHM

OpenMosix community is currently working on an additional experimental kernel patch to address the issue of migrating shared memory applications: Migshm [1], [14]. Migshm consists of 4 modules:

- Migration of shared memory processes;
- Communication Module;
- Consistency Module;
- Access log and migration decision

The first module allows OpenMosix to consider shared memory processes as transferable to other nodes (thus revolutionizing this clustering system behaviour). The second module allows the communication between processes migrated to different nodes that use the same shared memory page. The third module regulates access to a shared memory page, so that consistency for local copies of the same shared memory page is preserved. In order for the system to work correctly, it must be ensured that the last copy of each memory page is read and that, when a process modifies a page, changes are communinicated to all the other nodes. The last module keeps track of the accesses to each shared memory page; on the basis of information collected, load balancing algorithm is improved, allowing OpenMosix to takes better decisions about the most suitable node to receive a migrated process.

C. LTSP

Besides OpenMosix, another clustering system has been studied and tested: LTSP [11]. This system is designed according to a totally different philosophy with respect to OpenMosix. It is a Master/Slave Thin Client system, which consists of a certain number of diskless workstation connected to a server. All the applications, processes and data management run on it. Each workstation only provides keyboard, screen and mouse, for the user to access applications and data. LTSP gives a simple way to reuse low cost computers as graphical terminals, connecting them to a GNU/Linux server. This project was thought and built to be used in environments in which a great number of computers is needed and strict economical constraints prevail. As a matter of fact, LTSP gives the possibility to use obsolete PCs, remove hard disk, floppy and cdrom, add a network interface card, and generate a low cost cluster very rapidly and cost effectively. This kind of system has got great advantages from an economic point of view, because clients are very cheap, more robust (they consist of a lower number of components), have a longer time of obsolescence, less energy consumption and are more resistant in obstile environments. Each application runs on the server, so updating is cheaper and faster. Using this kind of system, one important disadvantage occurs, which is related to server performance and robustness. When the number of clients increases, the RAM of the server and its CPU speed must grow too. This could represent a problem where economical constraints are concerned. Furthermore, the entire cluster is deeply dependent from the server, which becomes then a single point of failure.

IV. Experiments

Experimentation was developed in three steps in which different clustering systems (OpenMosix and LTSP) and different

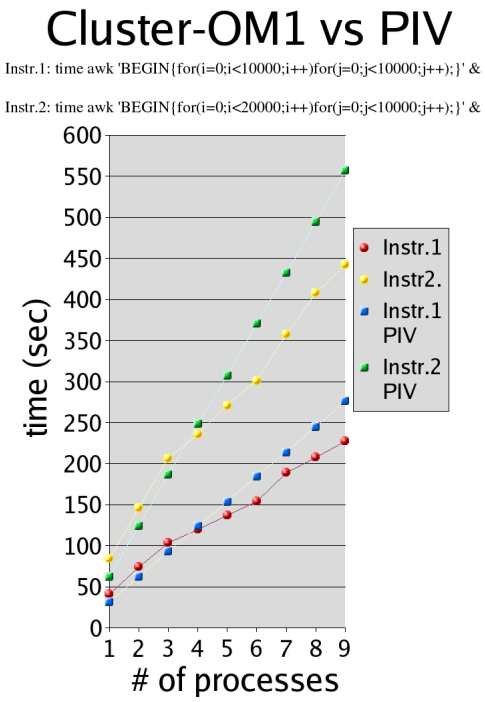

Fig. 3. ”Poor” Cluster compared to Pentium 4.

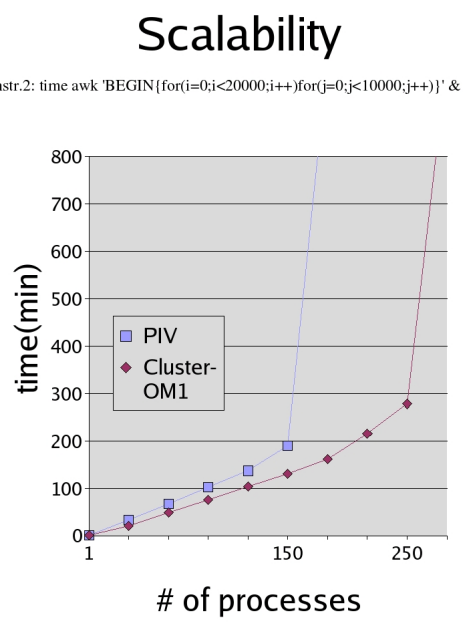

Fig. 4. Scalability comparison.

versions of the same system (OpenMosix 2.4.22 and 2.4.21) were installed and tested. The three steps are:

- test on OpenMosix
- test on OpenMosix+Migshm
- test on LTSP

A. Testbed

An etherogeneus cluster consisting of four nodes was used for testing:

- node 1: Pentium MMX 166MHz 32Mb Ram and NIC intell eepro100
- node 2: Pentium Celeron 733 MHz 64Mb Ram and NIC 3com 3c905
- node 3: Pentium MMX 200MHz 64Mb Ram and NIC realtek RTL-8139C
- node 4: Pentium 133MHz 48Mb Ram and NIC ISA 3com 3c509

A 100Mbit/s switch connected the nodes, a GNU/Linux Debian-3.0 distribution was installed on each computer and X server was installed on node 2 and 4. On each machine, kernels 2.4.21 and 2.4.22 were present, both compiled with the OpenMosix patch. Cluster performance was compared to an AMD Athlon XP 1700+ with 1Gb Ram and a GNU/Linux Gentoo-2004 distribution running on it. For measuring LTSP performance, a Pentium IV 1700 was used, with 384Mb RAM and RedHat 9.0 distribution, working as a server and a Pentium 133Mhz with 48Mb of RAM and GNU/Linux Debian-3.0 as a client.

B. Software test

Various kinds of tests were performed on the different clustering systems. The behaviour of these systems was evaluated with the aim of comparing ”poor” architectures to a modern one. The first kind of tests was made on the OpenMosix cluster (labelled Cluster-OM1) and is based on a purely theorical approach. Dummy cycles with a large number of loops were executed on it. The instructions used are as the following:

$time\ awk\ 'BEGIN\{for(i=0;i<20000;i++)for(j=0;j<10000;j++);\}'\ \&$. Each instruction is a single process, so more than one of them needs to be launched in order to see a migration and to observe an advantage from the use of OpenMosix. These cycles have a huge processing cost and stress the CPU consistently. The command *time* returns the time spent to execute the instruction; the & operator provides a new prompt for launching a new instruction straight away.

The second test is similar to the first one, but in this case an existing application was launched: **KFract**. Here is the instruction:

time kfract & xkill & ;

time returns the time spent to execute the instruction kfract. Once again, & sends the process to background, thus allowing to launch immediately another instruction before the previous application ends. The command *xkill* is used to terminate KFract with a click on its window (thus obtaining a sufficiently precise measure of time).

Each instruction kfract is a process and it is processed entirely on a node. Hence, advantages for the use of OpenMosix take place only if different instances of the same intruction are launched, so that they migrate on different nodes.

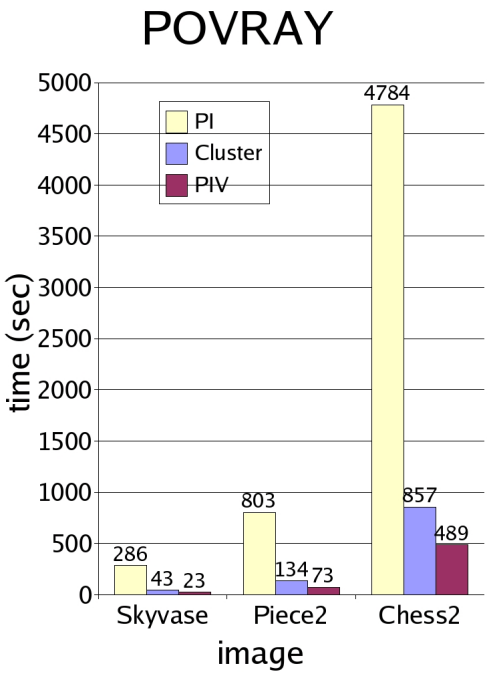

Fig. 5. Time needed by the Cluster to complete an image rendering of growing complexity, compared to the performance of a Pentium IV and of a Pentium I.

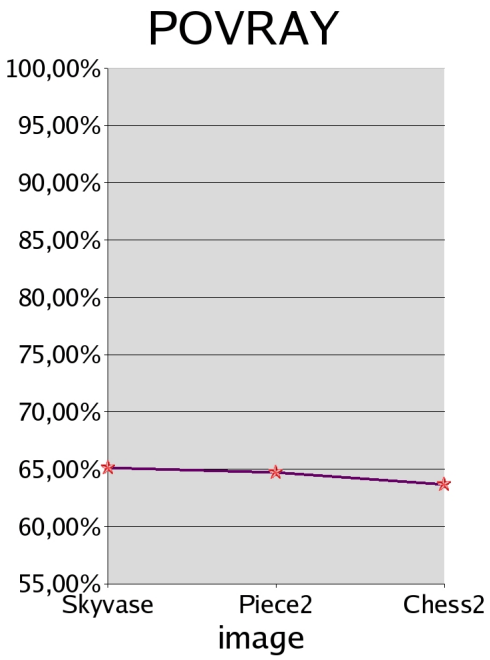

Fig. 6. Trend of the percentual difference of time.

In the third test, we measured the time needed by the cluster to execute a graphic rendering of a number of images with a growing complexity. For this purpose, a popular Open Source application called **Pov-ray** was used. Time to complete the rendering of different images is given by the application itself, at the end of the execution. In the test, cluster performance was compared to the PIV.

The last kind of tests, refers to applications that are frequently used by common users (i.e. Kword, Konqueror and Koffice). They all use shared memory, so OpenMosix patched with Migshm was put to the test in this case. To test the migration of shared memory processes, each of the following instructions were launched 5 time:

time kword openmosix-howto.txt & xkill &

time konqueror openmosix-howto.txt & xkill &

Time used to open the fifth instance of each instruction was considered to evaluate whether or not there is a gain by using Migshm.

The gain reached by a Pentium I when it is put inside an LTSP cluster as a diskless workstation was tested in the same way.

V. Results

In this Section, the results of the experimentation are presented and commented.

A. Critical number of nodes

Figure-1 shows time spent by different configurations of the ”poor” cluster to execute dummy cycles, varying the number of nodes and of processess. Cluster performance are compared to the PIV one.

The two-node cluster never reaches the performance of PIV. The difference between the ”poor” cluster with three nodes and the PIV, instead, decreases when the number of processess grows. In particular, the cluster reaches and surpasses PIV performance by the fifth process. The same result is obtained for the four-node cluster. In this last case, the overtaking of performance occures by the fourth process. Since the difference between the three-node and the four-node cluster is so small, it is possible to assume that four nodes are enough to compete with the PIV and we use this number of nodes for the other tests.

B. Cluster speed-up

Figure-2 shows the time spent by the four-node cluster to execute the two kinds of dummy cycles varying the number of processes, compared to the time spent by the most powerful node in the cluster. It shows time linearly growing for node-2 when the number of processes grows. The cluster reaches better performance growing both the number of processes and the complexity of the instruction.

C. ”Poor” Cluster vs. Pentium 4

In Figure-3, the execution of two kinds of dummy cycles is shown. The cluster obtains better performance in comparison to the Pentium 4 when the number of processes grows; in particular, the picture shows the crosspoint, i.e. the point in which the ”poor” cluster overtakes the PIV.

Figure-4 points out the comparison between the two kinds of architecture (”poor” and ”rich”) in terms of scalability, i.e. in

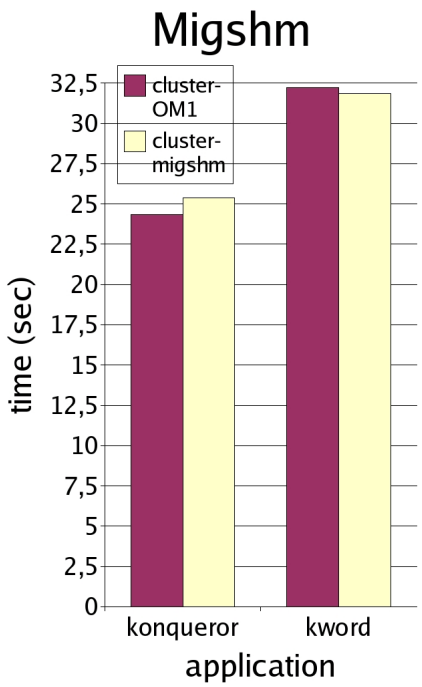

Fig. 7. Comparison of the times spent to open different shared memory applications, between the Cluster with and without migshm.

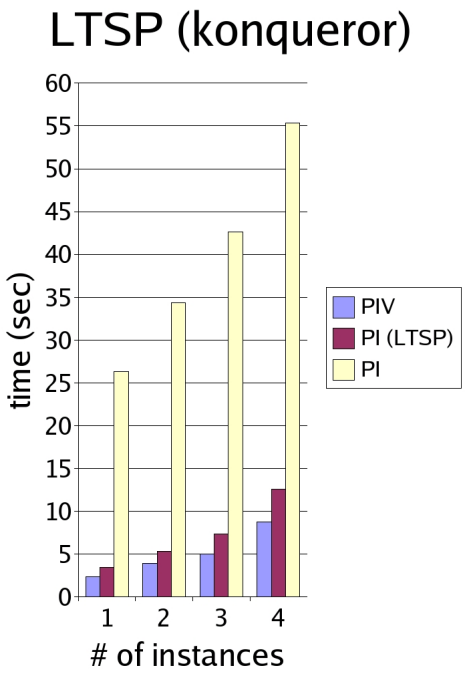

Fig. 8. Comparison between the LTSP cluster, PIV and PI, in terms of time spent to open different common use applications.

terms of number of jobs we can launch before the two systems saturate (i.e. when performance collapses). PIV reaches this condition with about 150 processes, while the cluster gets to saturation with more than 250 jobs launched.

D. Graphic rendering

Figure-5 depicts the results of rendering tests. Pentium I gets a huge benefit being a node of the cluster, obtaining a great reduction of the rendering time. Even if the cluster never reaches PIV in performance, it is remarkable that, as the complexity of the images grows, the difference between the two architectures decreases percentually, as Figure-6 shows.

E. Migshm and common user programs

Figure-7 presents a comparison between the two kinds of OpenMosix cluster analyzed in this study, i.e. with and without migshm patch. Tests verifyed that migshm effectively allows the migration of shared-memory processes, a fact that is a little revolution for OpenMosix. Improvements of the overall performance were not significant, though. Nontheless, the possibility of migrating applications is itself an advantage for users and in the case of a high number of applications open (which is not a so unfrequent situation) such an advantage can play an important role.

F. LTSP and common user programs

Figure-8 shows the time needed by a client of the LTSP cluster to open different instances of Konqueror, compared to the time the client alone (i.e. without LTSP) needs. The picture also shows how close the performance of the LTSP-client and of the PIV are.

VI. Conclusion

This reserch presented Trashware in a peculiar way, i.e. merging it with the pure accademic research on clustering systems. The aim was to optimize available resources through load distribution among computers in a cluster. Two clustering systems were tested: OpenMosix and LTSP, which differ for their architecture (centralized vs. decentralized) as well as for the purpose of their design (HPC vs. low-cost cooperating terminals).

Tests on OpenMosix showed that the more CPU-bounded applications are concerned, the better ”poor” clusters compete with modern architectures, or even make better. Speed-up increases with complexity and weight of applications.

In order to allow migration of shared-memory processes, migshm patch was experimented. Overall performance were not significantly increased, though the possibility of migrating applications is itself an advantage for users, especially when several applications need to be run at the same time. This is relavant for end-uder applications, which are the main objective of the investingation.

In this respect, much better results were obtained with LTSP. This system is suitable for contexts with economical constraints, thus resulting ideal for development and cooperation projects.

VII. Future work

Reaserch will be carried on along three main directions:

- Experience with Migshm, following the development of new versions for newer kernels;
- Combine thoroughly OpenMosix and LTSP approaches. This practice is already the subject of other workgroups [5], but to the best of our knowledge their research is only limited to provide a simple and fast way to add new diskless nodes to clusters rather than focusing on the distribution of all functionalities of the LTSP server through OpenMosix.
- Combine OpenMosix and UML (User Mode Linux) to have a virtual OpenMosix cluster. This would allow to test new kernels and software safely. Moreover it could be possible to use the computational distribution of OpenMosix to try to migrate entire virtual Linux system (generated by UML).

ACKNOWLEDGMENT

The authors wish to thank the Trashware group of ISF-Roma for its constant support, the colleagues at the Bugs Lab of Strike S.P.A. who provided the main lab for the experiments and Francesco Ruffino of IASI, CNR (Italian National Research Council) who gave us access to a more advanced testbed for benchmarking.

REFERENCES

[1] Moshe Bar and MAASK team. Mihshm. In *Linux Congress*, 2003.
[2] Amon Barak and Oren La'adan. The openmosix multicomputer operating system for high performance cluster computing. Technical report, Institute of Computer Science, The Hebrew University of Jerusalem, 1998.
[3] Amon Barak and Amnon Shiloh. Scalable cluster computing with openmosix for linux. Technical report, Institute of Computer Science, The Hebrew University of Jerusalem, 1998.
[4] Carlo Buzzi. I numeri ed i rischi della spazzatura informatica. *Punto Informatico*, 2003.
[5] Martin Daniau. *LTSP-Mosix*, 2001.
[6] WWF-Consorzio Ecoqual'IT. L'e-waste ladri di futuro-le cause e gli effetti della mancata gestione dei rifiuti tecnologici. Technical report, WWF, 2002.
[7] Golem. *Trashware HowTo*, 2001.
[8] Alessandro Inzerilli and Igino Gagliardone. Prodigi-il progetto tunisia, 2003.
[9] Davide Lamanna and Daniele Arena. Progetto kosovo,un anno dopo. In *Atti del Terzo Linuxday Italiano*, Firenze 2003.
[10] Davide Lamanna and Stefano Puglia. Il software open source come strumento di diffusione della conoscenza informatica: un progetto in kosovo. In *Atti della conferenza annuale AICA*, Settembre 2003 Trento.
[11] James McQuillan. *LTSP-Linux Terminal Server Project-v3.0*, 2002.
[12] Pippa Norris. Digital divide. *New York:Cambridge University Press*, 2001.
[13] Kuehr Ruediger and Eric Williams. Computers and the environment: Understanding and managing their impacts. Technical report, Kluwer Academic Publishers, October 2003.
[14] Mulyadi Santosa. Checkpointing and distributed shared memory in openmosix, Aprile 2004.
[15] George Sciadas. Monitoring the digital divide...and beyond. Technical report, Orbicom, 2003.