Virtual Distro Dispatcher: A costless distributed virtual environment from Trashware

Flavio Bertini, D. Davide Lamanna, Roberto Baldoni

{flavio.bertini, davide.lamanna, roberto.baldoni}@dis.uniroma1.it

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università degli studi di Roma "La Sapienza", Italy

Abstract. Obsolete hardware can be effectively reused through intelligent software optimization, which is possible only when source code is available. Virtual Distro Dispatcher (VDD) is a system that produces virtual machines on a central server and projects them on a number of costless physical terminals. VDD is the result of an extreme software optimisation based on virtualization and terminal servers. VDD creates and projects Linux distros that are completely customizable and different from each other. They are virtual desktop machines that can be used for testing or developing and are completely controllable directly from each terminal. Memory consumption has been strongly reduced without sacrificing performances. Test results are encouraging to proceed with the research towards clustering.
Keywords: Trashware, LTSP, User Mode Linux, Clustering, Virtualization.

1 Introduction

The massive diffusion of Information and Communication Technologies (ICTs) has as a consequence that a huge quantity of obsolete computers are widespread around the world. 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 vice versa. Such a vicious circle causes a damage for users, who are obliged to buy new hardware even if they could use the old one in a much more efficient way. In this scenario, Trashware movement [8] is spreading worldwide to give to computers the correct time of obsolescence, thus facing the ecological problem related to e-waste from its very roots. Doing Trashware means working on a sustainable adoption of hardware resources which are still effectively usable and are instead destined to dumps. Trashware is deeply related to Open Source and Free Software movements. Open Source software represents an indispensable tool, as it enables full control on hardware/software optimisation, whereas the sadly common practice of not distributing software source-code prevents users from almost any possibility of optimising their systems. The optimisation we propose in this article is based on two main concepts: virtualization and terminal

server. Virtualization is normally used to supply for servers (e.g., FTP or HTTP servers), which guarantee security and protection as they are virtual. Instead of this, we propose to run proper virtual desktop machines and then project them to scarce and old computers (working as terminals), which are literally costless thanks to Trashware. Such a distributed virtual environment is excellent for testing Linux distributions. The Linux Terminal Server Project (LTSP) [3] is a very cheap solution for building computer labs by just buying one good PC and using a number of old machines as terminals. Virtual Distro Dispatcher (VDD), which is based on it, supplies for several Linux distros that are completely customizable and different from each other. They are virtual machines that can be used for testing or whatever purpose, as they are completely controllable from each terminal, whereas LTSP provides terminals that are all identical, lacking in control, and only configurable *una tantum* by the server administrator. VDD can dramatically reduce the cost of hardware for such a complex development environment and it is licensed as GPL software. The virtualization system used by VDD implies five main constraints to be respected:

- **C1** - Open Source software (for integration with other systems and maximum optimisation);
- **C2** - Modularity and file-based structure;
- **C3** - Easy and quick restore;
- **C4** - User level kernel execution;
- **C5** - Linux only emulation (no need for emulating other operating systems).

The choice is among five popular virtualization systems: VMWare [9], Xen [10], Qemu [11], User Mode Linux [1],[4] and Bochs [12].

	C1	**C2**	**C3**	**C4**	**C5**
VMWare		√			
Xen	√	√	√		√*
QEMU	√	√	√		
UML	√	√	√	√	√
Bochs	√		√	√	

∗ Microsoft Windows® license does not allow modifications to the operating system. In the future, hopefully, it could be that we will be able to emulate that operating system too

UML is very easy to manage, because we only need three files: an executable guest kernel image, a root filesystem and a swap filesystem. Also for this reason, we chose to start our research with UML, that is the only one satisfying all the five constraints. That is also the main reason why our final choice was UML (see the table above). The aim of the paper is to present an innovative system which allows users and developers to approach Free Software in a secure and inexpensive way. We use LTSP as a mean for distributing virtual sessions, as it

is quite stable and provides an easy and fast bootstrap for a large number of thin clients. User Mode Linux is also very fast in starting and it is quite efficient performance-wise. In section 2, we present some of the related works that are the starting point of our piece of research. Section 3 explains what virtualization is and how to get a virtual system working, focusing particularly on UML. In section 4, we talk about the opportunity for joining UML and LTSP and in section 5 we explain how to do it. Section 6 describes security and performance issues that are relevant for a virtual laboratory made through VDD. Section 7 presents performance tests in a number of cases and section 8 deals with future directions of our work, especially in order to speed up the whole system.

2 Related work

In Italy, there are several research groups doing Trashware, like Binario Etico[1]. Their work is strictly focused on Free Software. For this reason, they provide an excellent technical support, both on software solutions and configuration, without which our research work would have not been possible. We were able to set up a quite sophisticated testbed without any significant charge, except from the purchase of two new machines, being able to support the working charge coming from a network of obsolete computers (12 terminals). An important contribution to Trashware was given by [5], describing how to integrate clustering with LTSP. An extensive performance analysis shows how clustering can be useful, especially in a distributed environment. This is the starting point for the present work, which take into account virtualization as a new direction to be explored and integrated. About virtualization, User Mode Linux (UML) [1],[4] appears appropriate for integration with other technologies, in particular the ones we are interested with. This is mainly due to its filesystem based structure, which allows for easy managing its components as separate modules [1]. Our piece of research is tightly based on UML, through which we have been able to obtain a fast and easy to use virtual distribution. Many research groups [1],[4], and individuals [6] are contributing to UML development[2]. Main contribution in [6] is the introduction of skas0 mode (see section 6), as an alternative to tt-mode, which presents strongly reduced security risks. Linux Terminal Server Project [3], is also a good idea when Trashware is concerned. LTSP is already helping in several circumstances. For example, schools can benefit of what Trashware and LTSP does [7], when licenses and new hardware are not affordable. As the use of LTSP and Trashware is growing everyday, this piece of research intends to take to the extreme their possibilities. HPC clustering is an effective way to increase performance [5],[8]. Our model of LTSP/UML based laboratory would benefit a lot from clustering in order to reduce the physiologic slow-down caused by intense use of several virtual machines. An HPC system like OpenMosix has been

[1] Further informations at: http://trashware.linux.it/wiki/TrashWiki - http://www.binarioetico.org - http://www.isf-roma.org

[2] On [6], it is possible to get host and guest kernel-patches maintained by Paolo Giarrusso.

already used with LTSP, as described in [5]. We now need to join clustering with Terminal Server concept and Virtualization. Another important contribution to this effort has been brought by a research study on cluster solutions, aiming at scalability and high availability, based on OpenSSI, which we acknowledge in section 9. In order to get into details of what VDD is and what is based on, we need to have a look on what LTSP and UML are.

3 Virtualization

The present piece of research is strongly based on virtualization. The system we set up is able to run a number of real and complete GNU/Linux virtual distributions. Designing a virtualization system involves the evaluation of a number of issues concerning the use it is called for. There is a fair amount of software available which supports the emulation of an operating system. Among to most famous, one can cite: VMWare, Bochs, Qemu and Xen. Our first requirement was to implement an Open Source system under GNU/GPL License[3]. The choice of User Mode Linux (UML) as the engine for VDD appeared to us to be the most appropriate. "User Mode Linux is a safe, secure way of running Linux versions and Linux processes"[4]. UML is composed of well-defined basic parts, including an user-mode kernel image, which is executable in a command line shell, a root filesystem (i.e. a single file fits all) and a swap file. UML is normally used to run buggy software, including new kernels and distros, and to hack around without exposing the physical Linux box. The fact of being based on filesystem images is particularly suitable in a distributed environment like ours, where failover can not be disregarded. Recovery is easy and fast. Other virtualization systems work in such a way[5], but do not offer opportunities for integration with the rest of our system, which is completely Open Source. Also, the possibility of setting up very easily a highly configurable development system is particularly attractive when different virtual workstations needs to be created and distributed in the same environment. Besides, thanks to its modular structure, one can recover or substitute all its parts separately. If a different filesystem becomes necessary, for instance, one can just operate on it, without influencing, e.g., the executable kernel file. Another important aspect we took into account in our choice is that it is possible not only to shape the host kernel, as we already highlighted, but also the guest (UML) kernel. In such a way it is possible to supply for a huge quantity of virtual hardware, much more than what real hardware can do. The possibility to specify, inside the UML kernel configuration file, options like the CPU architecture used by the host system, makes UML flexible and adaptable pretty much in any circumstance.

[3] General Public License is a Free Software license

[4] Directly from http://user-mode-linux.sourceforge.net/

[5] For example, VMWare works using image files containing the full installation of an operating system.

4 UML and LTSP: Virtual machines and physical terminals

UML is often being used to create operating systems that are entirely dedicated to accomplish only one task. For example, it is possible to dedicate a Linux machine entirely to a FTP Server or a Web Server. This is particularly useful when it is necessary for security reasons to limit the number of services managed by a server at the same time. Apart from preventing attacks, such a feature is attractive for us because a stable version of a machine is fully recoverable in any time, which makes it ideal for a development workstation. UML allows to consider guest systems as real workstations. If one looks the machine from outside, within the LAN topology it is part of, she will be able to see all UML machines as normal PCs belonging to a computer network. If one is using a PC in the same LAN, she will never know if she is communicating with a real machine or a virtual one. Normally, all the virtual machines are physically run and used on a single machine. Remote login to a particular virtual machine is of course possible, but this is not at all what we mean by dispatching machines on nodes of a LAN. We are indeed interested in projecting the whole machine to a different node as opposed to just gain access to it from outside. For this reason, we considered the integration of LTSP in our system. LTSP allows to connect several diskless thin clients to a Linux Server. LTSP is based on four-services: TFTP (Trivial File Transfer Protocol), DHCP (Dynamic Host Configuration Protocol), XDMCP (X Display Manager Control Protocol) and NFS (Network File System). These four services, together, allow for file transferring between nodes, remote login, managing IP addresses without conflicts, and sharing the minimal filesystem used by all nodes to work. The best way to join LTSP with UML seemed to us to be through the Xorg[6] Server support. As we said before, we need to project virtual sessions to all the LAN thin clients. Xorg is actually a server, even if it runs on a thin client. For this reason, we used it to show graphical virtual sessions (i.e. KDE executed in a virtual terminal by an UML session). Our approach is to put all thin-client Xorg servers in listening mode on a specific socket using a minimal shell. Once UML virtual machines are up and running, one is able to project their KDE sessions by simply doing an environmental variable export. The previous value of `DISPLAY` variable needs to be overridden in order to indicate what is its new X server address and socket. Finally, our window manager can be run by launching `startkde` command.

5 Dispatching on terminals: Virtual Distro Dispatcher

As we said, the purpose of this piece of research is to find the contact point to unify virtualization to Terminal Server concept. The basic idea is to project every virtual session (possibly completely different from each other) on every thin client, in a perfectly transparent way. Virtual Distro Dispatcher provides

[6] The X.Org Foundation Open Source Public Implementation of X11.

the possibility to obtain several different environments, one for each terminal. In such an effort, there are two main approaches to consider: the first one is quite easy to understand and to implement and consists of installing UML and setting up LTSP inside it. VDD logic is a simple solution to realize a join between these two technologies, but it has driven us to a fundamental issue: using a virtual distro in order to supply for a needed service[7], i.e. LTSP itself. Figure 1 shows the architecture for this first approach. As it appears clear at a glance, this is

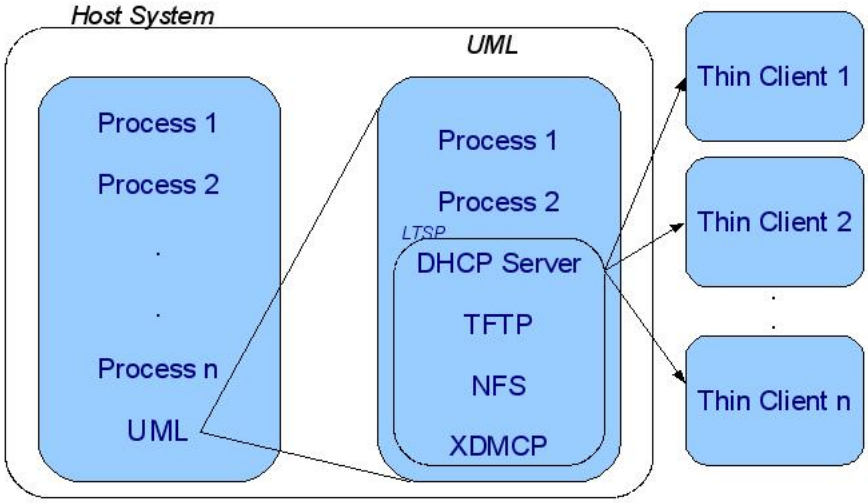

Fig. 1. The first approach

a restrictive methodology. It does not allow neither to obtain satisfying performances, nor to get a scalable and customizable system. The reason why we do not like this approach is that it is not possible for developers to have a choice. Our work is mainly oriented to developers and debuggers necessities. What we intend for customizable, in this context, is to give to developers the possibility to choose what distribution to use for their tests or works. The other approach is certainly more effective and proposes the opposite of what we have just described: UML inside LTSP. In this case, LTSP plays the role of an intermediate mean, which allows all terminals to use a virtual session. This approach is implementable in two ways: Dynamic Assignation and Static Assignation (Figure 2 and 3). In the first mode, Server decides what is the distribution to send to a thin client. In the second mode, instead, the generic thin-client user chooses what to use herself.

In our laboratory, we used the Static Assignation. This choice is based only on a practical reason. The laboratory is used also by inexperienced people (e.g.

[7] In §4, e.g., we talk about installing a FTP server into a virtual machine.

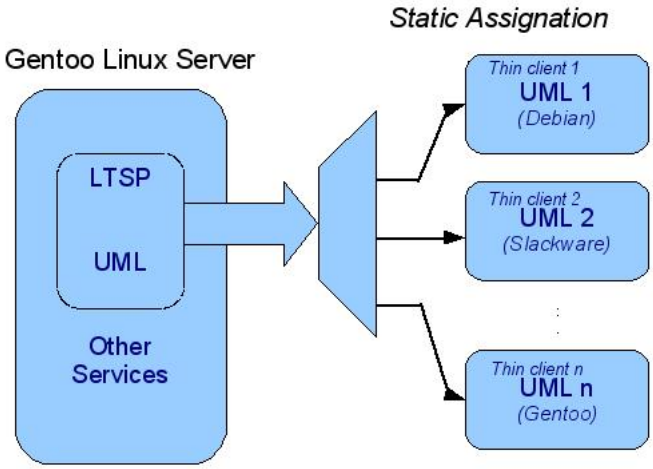

Fig. 2. The second approach in *Static Assignation* case

students) and we thought it was better to decide ourself what distribution to send on a thin client. The strong point of the last methodology (UML inside LTSP) is that it is possible to switch easily among two assignation modes. The second mode is recommended in a development environment in which node users are expert system administrators. In fact, if some developer would like to do a new kernel testing or try buggy software, she will decide what distribution is better to use. Practically speaking, all what we need is to start UML on the host, directly from the Terminal Server and, once a thin client has logged in, we can get access to its minimal shell. From there, we can open an X Server socket. Next step is to start a windows manager (e.g. KDE) from the Terminal Server (into UML) after specifying the `DISPLAY` environment variable in order to tell UML where its display is and to run the graphical session. The final result is that it will be possible to use Linux by KDE on a thin client screen, all in a virtual session.

6 Performance and security issues

UML development has pursued both memory usage efficiency and protection. There are currently three working modes available, each of which presents different performance and security issues: TT-Mode, SKAS3-Mode, SKAS0-Mode.

TT-Mode: a first way of working of UML is based on a tracing thread mode (tt-mode), whose job is to listen guest System Calls and to forward them to the host system. Besides, in this mode, memory is basically shared between processes and UML kernel. Typically, the guest kernel is allocated in the upper bound of the address space. Processes instead, are allocated in the following 5

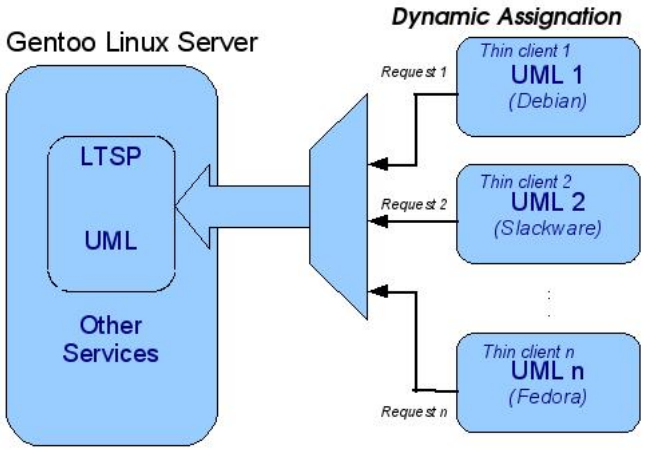

Fig. 3. The second approach in *Dynamic Assignation* case

GB of memory. This causes a serious security problem, because the memory is shared between processes and kernel code. In fact, due to this situation, a false process can easily execute code in kernel mode and prevent the guest kernel to be a real (good) process. It is sufficient for someone to know the UML internal structure and she can get into the host system directly.

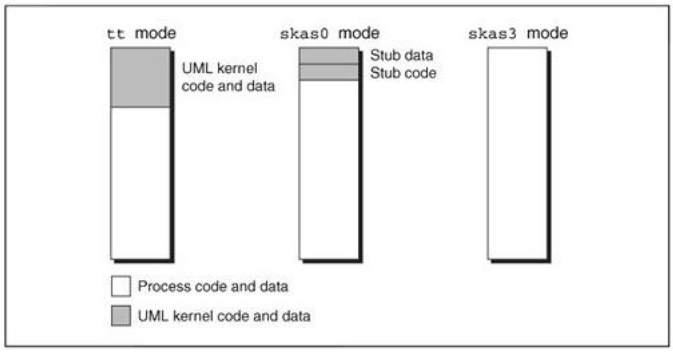

Fig. 4. Address space division in each situation: tt, skas0 and skas3 modes

SKAS3-Mode: Problems encountered in tt-mode can be solved by using a patch, whose aim is to modify the host kernel as well as to manage the address space in a different way. Skas3, or *Separate Kernel Address Space*, has now got to the third version. When applied to the host kernel, it adds a new file in `/proc` (`/proc/mm`) and creates a new address space to hold the guest kernel. Regarding performance aspects, there is a big amount of wasted memory working in this

mode. We can say that a lot of slow-downs can occur because, for every process on the guest system, the host creates a new one straight away. The reason why such a waste of memory occurs is that every process can live in a separate address space. Every process has got a separate address space, which it shares with kernel code and data. Skas3 prevents false processes to deliver an attack. This is possible because of the isolation between guest kernel and processes. In this case, we do not have memory sharing. There are no tracing threads and system calls are intercepted by the guest kernel itself. For every guest process, there is only one host process. There are two advantages doing so: no waste of memory and no risk for UML crashes, resulting in a strongly reduced possibility to compromise the host system integrity. As a matter of fact, what happen inside UML is confined in itself. The only Skas3 con is that it is not always supported. It depends on an old UML version and the host CPU architecture is not already supported.

Skas0-Mode: This mode can be used when Skas3 mode is not supported. For example, if we had an X86_64 host machine, skas3 would not be able to work.

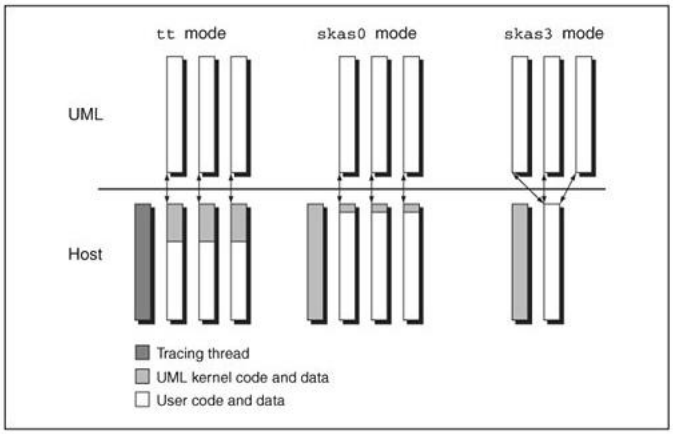

Fig. 5. Processes behaviour in guest and host system for each mode

As explained in [1],[6], it is possible to make something between Skas3 and tt-mode. In fact, like the tt-mode, for every guest process, there is an host process and, as for the Skas3 mode, the guest kernel will be executed in a separate address space without a tracing thread. In this way, we can take advantage of Skas3 performances even though there is an amount of wasted memory like the tt-mode. Skas0 mode is less efficient than Skas3 but it is certainly better than tt-mode. The Skas0 basic idea is to insert a little quantity of kernel code in the shared memory in order to reduce opportunities for attacks. The system we developed has been built upon two of three previously described modes:

Skas3 for the x86 CPU architecture and the Skas0 for the x86_64 CPU (only for performance tests). Mainly, we have to consider the Skas3 mode. We do not use UML versions which are not supporting Skas3 mode. Regarding the filesystem image, which contains the GNU/Linux distribution, we chose to create a new one by means of *debootstrap* utility. Our UML distribution is Debian GNU/Linux. We got kernel sources from the official website[8]. After configuring the Debian base system, we could start to use User Mode Linux.

7 Performance analysis

Our testbed has got the following characteristics:

- 1 Intel Pentium 4 3000 MHz
- 1024 Mbytes RAM
- 4 SATA 80Gb Hard Disks
- 13 100Mbps Ethernet cards
- 12 diskless thin clients[9]
- 1 100Mbps Ethernet switch

After several tests, the final system resulted to be quite usable, without significant slow-downs. In order to analyse system performance, we chose to evaluate computational time of a CPU bound process. The right test for us, seemed to be a kernel compilation on its *defconfig*[10]. Our tests, whose results are shown in Figure 6, consist of a vanilla kernel compilation in three cases: firstly on the host system (A), then on a thin client (B) and finally on a thin client inside UML (C). This test has been repeated ten times for each case in order to estimate a mean value for any measurement.

The most significant case is C. As it is shown in Figure 6, the compilation time in this third case is higher with respect to case A and B. These results confirm the actual benefit that one would get by integrating a clustering system, in order to increase performance (see section 8). The whole system could be part of a bigger one, in which it is possible to integrate further technologies to get a fully functional and faster final system. Although UML is host-CPU optimized, the aim of who is writing is to go ahead. We are trying to find out further techniques to improve the whole service. As a matter of fact, results persuaded us to use Clustering HPC systems in order to improve performance hits, even if we obtained a fully usable system with no significant slow-downs. HPC Clustering oriented technologies we are considering are OpenMosix[13] or OpenSSI[14]. Another possibility is to use an alternative virtualization system. The next step can be to experiment Xen.

[8] http://www.kernel.org

[9] Generally we used many different architectures for thin clients, but the typical one is a Pentium II@233 MHz with 64 Mbytes RAM, floppy disk, CD-ROM reader and an Ethernet card.

[10] For each test, we did `make defconfig` on the kernel tree.

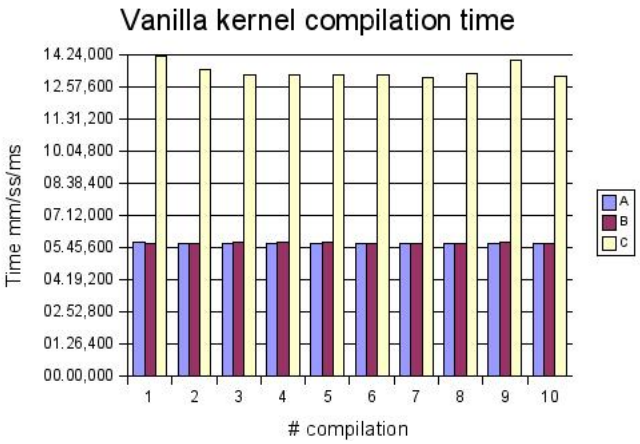

Fig. 6. Benchmark results

8 Conclusion and future works

The combination of different technologies makes it possible to realize new systems, aiming at incrementing supplied services and/or their performance. Open Source paradigm is essential in such an effort. The main innovation we presented here is the fusion between a distributed system and virtualization. LTSP already allows us to distribute login and GNU/Linux sessions to a thin-client LAN. But systems showed on the screens are not virtual machines. They are the remote execution of a single physical machine. Whereas we succeeded in distributing virtualization on to a diskless node of a LAN. Virtualization dispatching allows users to use their own distros and developers to do their real work in a costless, secure and comfortable virtual environment. Performances analysis is an important aspect to be considered when joining LTSP and UML. Virtual Distro Dispatcher could be much lighter if supported by a distributed server instead of a central one. In fact, if we consider large scale service distribution, the central server could be lagged. Since we would like to enlarge VDD environments, we think is absolutely necessary to boost up our system by using HPC clustering. Figure 7 shows a prototype of a full VDD based laboratory, which is also supported by an HPC Clustering system.

In this case, we have two clusters and two possible LANs: one for virtual-distro based thin-clients, and another one for LTSP based thin-client network. We are hence proposing something scalable, accommodating different necessities. This is a multiple sections example, whose aim is to show how one can easily change and personalize the global system as needed. We intend now to set up an HPC Clustering system, in order to enlarge virtualization dispatching without any lag, so that more users can access our system.

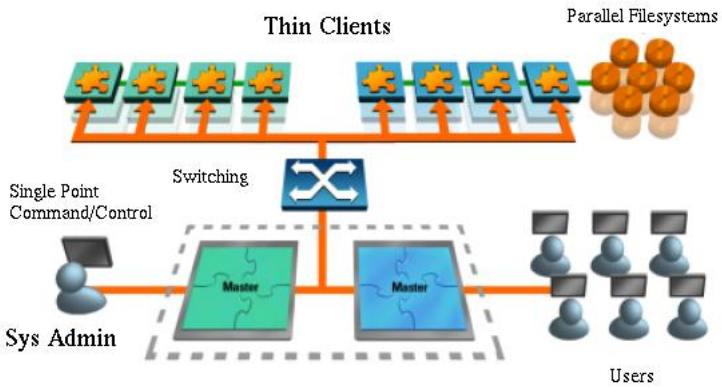

Fig. 7. Future laboratory prototype

9 Acknowledgements

We wish to thank Andrea Leone. His work on scalable and high-available clusters on a distributed platform is the presupposition and the natural course of our piece of research. The support of our colleagues Daniele Carcasole and Alessandro Di Stefano has been essential to mastering LTSP, a system on which they conducted in-depth studies and research. Finally, we are grateful to Binario Etico that provided hardware resources, expertise and a laboratory to conduct research and tests.

References

1. Jeff Dike - User Mode Linux® (Bruce Perens Open Source)
2. William Stallings - Operating Systems: Internals and Design
3. Linux Terminal Server Project - http://www.ltsp.org
4. The User-Mode-Linux Kernel Home Page
   http://user-mode-linux.sourceforge.net/
5. Ruggero Russo, Davide Lamanna and Roberto Baldoni - Distributed software platforms for rehabilitating obsolete hardware
6. Paolo Giarrusso: Skas and Guest patches
   http://www.user-mode-linux.org/ blaisorblade/
7. Tina Gasperson, Old school cuts ties with Windows,
   http://business.newsforge.com/business/05/09/28/1843234.shtml?tid=37
8. http://trashware.linux.it/wiki/TrashWiki
9. VMWare - http://www.vmware.com/
10. Xen - http://www.xensource.com/
11. Qemu - http://fabrice.bellard.free.fr/qemu/
12. Bochs - http://bochs.sourceforge.net/
13. OpenMosix - http://openmosix.sourceforge.net/
14. OpenSSI - http://openssi.org/