PROCEEDINGS OF SCIENCE

A High Availability Solution for GRID Services

Álvaro López Garcia[*a], **Mirko Mariotti**[b], **Davide Salomoni**[c], **and Leonello Servoli**[ab]
[a] *INFN - Perugia*
Via A. Pascoli, 06123 Perugia (Italy)
[b] *Physics Department - University of Perugia*
Via A. Pascoli, 06123 Perugia (Italy)
[c] *INFN - CNAF*
Viale Berti Pichat, 6/2 - 40127 Bologna (Italy)
E-mail: alvaro.garcia@pg.infn.it mirko.mariotti@fisica.unipg.it leonello.servoli@pg.infn.it davide.salomoni@cnaf.infn.it

In this work we describe a prototype to obtain the High Availability of GRID services (CE, SE, WN), using the XEN paravirtualization paradigm, coupled with NAGIOS and Cfengine packages. The architecture of the prototype is outlined together with some of the first results; noticeably a dead time of the order of one minute to restart a server either in case of software or hardware failures has been achieved. No failure of running jobs due to CE downtime in this configuration has been observed.

XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research
April 23-27 2007
Amsterdam, the Netherlands

[*] Speaker.

PoS(ACAT)013

1. Introduction

In this section it is described the origin of this work: the main elements of the problem to be tackled and the virtualization concept will be briefly discussed.

1.1 The LHC experiments

In 2008 the Large Hadron Collider (LHC) at CERN will start to collide high energy proton beams and its High Energy Physics Experiments will start to collect and record huge amount of data, in the order of petabytes per year. These data should be analysed by thousands of scientists belonging to hundreds of research institutes scattered all around the world. In such scenario the GRID computing paradigm has been adopted as the preferred framework to organize the computing effort, as it should make easier for the single physicist to access to higher computational power and storage resources than the capacity granted by a single institute.

1.2 The Grid

The *Grid*, as defined by their biggest promoters in the nineties is "(...) a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [1]. This concept is similar to the model applied to the electric power or the telephone grid, in which an end-user gets plugged, uses the resources needed just for the necessary time and then gets unplugged.

Based on this concept, the *Grid* –or more correctly the *Computing Grid*– concept gets shape as a distributed computational paradigm in which an unspecified, heterogeneous, geographically distributed and variable number of computing resources are glued together, forming an unique supercomputer seen by the end-users.

The goal of this model is to provide the users –either individuals, scientific and research institutes and enterprises– with the necessary resources to resolve huge computational problems (for example on high energy physics, meteorology, earthquake simulations, protein folding, etc.) that otherwise could not be treated. The Grid also hides all the complex structure behind it by providing some kind of interface, thus making theoretically possible for the users to worry just about their problem, and to ignore how the system is organized to help then.

For a more in-depth explanation of the Grid architecture, please refer to [1, 2].

1.2.1 The LHC Computing Grid

In this context the LHC Computing Grid (LCG) has been developed from the first prototypes (2000) to the first deployed infrastructure [12] in 2003 [3], to the actual configuration, with the target of building a computing infrastructure for the LHC experiments. The middleware adopted is the one from the EGEE European project, called gLite –first based on the European Data Grid middleware, then on the LCG middleware, and now renamed gLite with the end of the second EU funded research program–. The LCG is today defining the basis of the European Computing Grid, both in infrastructure and middleware.

The Italian INFN[1] [13] is an important contributor to the project, participating not only with its more than 40 sites, but also developing some essential middleware components. The INFN also

[1] National Institute for Nuclear Physics

Table 1: Downtime values for a High Available system

Percentage	Time
99%	3.6 days/year
99.9%	8.78 hours/year
99.99%	52.6 minutes/year
99.999%	5.26 minutes/year

makes its own customizations to the gLite middleware, releasing the INFN-Grid [14] middleware, a fully gLite compatible distribution with modifications to satisfy INFN's sites peculiarities.

1.3 The High Availability

The term *Availability*, when talking about computer resources, refers to the level at which a system is able to guarantee the continuity of its operations during a specified time period (usually a year). Thus, a system that implements *High Availability* should ensure an elevate degree of functionality, typically over 99% of the time for a given period. On Table 1 are shown the downtime periods corresponding to increasing levels of High Availability; to pass from one level to another is increasingly demanding, starting from the very first levels.

The candidates for highly available systems are critical systems that provide services to the end-users –both directly or indirectly–, in which an extended period of downtime (intending downtime as a "*not available period*") means loss of data, time and/or money.

Inside a Grid site there is a certain quantity of different nodes, each one with different roles and with different requirements. For example, inside a LCG site there are some "vital" elements –i.e. needed by the Grid to work– like the *Computing Element*, the *Storage Element*, the *Resource Broker*, etc. that, in case of failure, may cause the malfunctioning of the entire Grid or of a relevant part of it; whilst there are other components –as the *Worker Nodes* and *User Interfaces*– which are not so critical.

As said in the Section 1.2.1, the INFN takes part on the LCG project and is interested in finding a reliable solution to make highly available its Grid sites, with the possibility of extending it to to the whole LCG project.

There are several ways to implement the high availability concept. A classical approach is the replication of the hardware, coupled with a data synchronization and a recovering mechanisms (as the one provided by the well known Linux-HA solution) providing a reliable solution in which the downtime of the service is minimal –as there is a second machine ready to start working when the original machine fails. This approach, however, has some inconvenient as the duplication of the hardware and the increase of the human resources needed to manage the system. Also, the computing power of these machines is wasted, as they are inactive until something fails.

1.4 Virtualization

One of the definitions for virtualization[2] is the following: "the use of a specific software – *Virtual Machine Monitor* (VMM) or *hypervisor*– to create different execution environments on a

[2]Intended as platform virtualization, defined as the simulation of a whole computer.

given machine –*host* or *physical machine* (PM)–, making possible to execute another operating system –*guests* or *virtual machines* (VM)– on top of them, in an isolated, independent and secure way".

As easily understood from the above definition, the virtualization of a given machine should introduce a performance loss, because the hypervisor needs to ensure that the guest systems do not access the hardware directly, i.e. it needs to discover and to trap the instructions attempting to do this. The complexity of these operations depends on the CPU architecture and it will influence the performance loss introduced by the virtualization. For example, to virtualize the x86 architecture[3] there are 17 instructions that need to be trapped, and this is introducing a non negligible overhead.

Recently another approach has emerged to implement the virtualization concept, the so called *paravirtualization*[4], in which the guest operating system needs to be modified (or designed ad-hoc) to use a special API[5] provided by the hypervisor running on the host, instead of using the architecture system calls. Doing so, the hypervisor does not worry about the trapping of any instruction, as it relies on the guest system to use its API.

One of the most known products for the x86 paravirtualization is Xen [15], in which the native system performances are reduced by only few percent due to the overhead [4], at the price of porting the operating system to the paravirtualized Xen architecture.

In the last year Intel and AMD presented their x86 virtualization extensions (named Intel-VT [6] and AMD-V [7]), which were aimed to allow an hypervisor to run a guest operating system without the need of being modified. These technologies made possible to virtualize systems under the x86 architecture, removing the overhead with respect to the native solutions.

In general the virtualization technique brings several benefits:

Hardware and Software detachment: By virtualizing a machine, it is no more dependent of the underlying hardware. Doing so, a VM could run on any host without the need of being reconfigured. Furthermore, an old software which may not run by itself on a cutting-edge hardware could be virtualized and then run on any physical machine.

Isolation: Several virtual machines may run on a single host, each one independent from the others.

On demand machine creation Single-use machines for testing purposes could be created easily.

Migration: Some hypervisors allow a virtual machine to be migrated from one host to another (if its filesystem is reachable by both hosts).

Security: All the network traffic will pass through the physical hosts, so some security policies and tools can be used to monitor the guests systems on a stealth way.

Execution pausing: Snapshots of virtual machines can be made, allowing to pause the execution and then restarting it.

[3] In this case the hardware is fully (or almost fully) simulated.

[4] Which is not a new technology, as it was originally developed by IBM in the '70s with the development of their CPM/VM operating system family.

[5] Application Program Interface

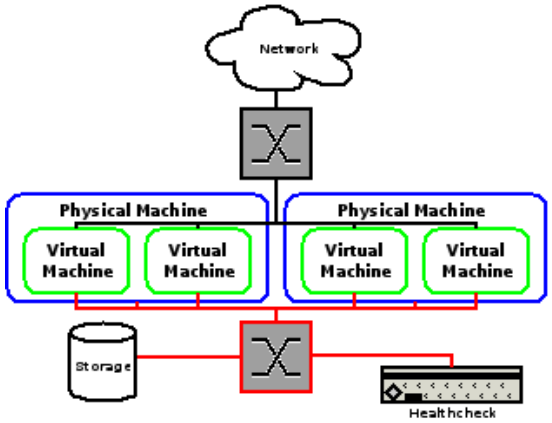


Figure 1: Conceptual schema of the HA using virtualization

Few spare machines: As more than one virtual machine can be booted on a given host, there is no need to have spare machines ready to substitute a broken one, as every machine could be used to do so.

1.5 Motivation

The scope of our work is to take advantage of the virtualization techniques and its benefits, basing on the Xen hypervisor, to present a new conceptual model and the related prototype for making highly available the critical components of a INFN-GRID site. Some tests of the prototype performance are also presented.

2. HA using virtualization

2.1 The architecture

The prototype architecture we are presenting (Figure 1) has three main components:

Virtualization of the components: The hypervisor will be installed on every physical machine and a number of virtual machines will run on top of any hypervisor; on any virtual machine will run a service. In this configuration the physical machines are just a layer where to deploy the virtual ones, which will be the ones that do the "*real work*".

Use of network storage: The images of the virtual machines are going to be stored on a storage device over the network which will be reachable by any physical machine. By doing so, any virtual machine is able to run on any physical machine.

Monitoring and control: In our approach is present a component, distributed or centralised, which is dedicated to the monitoring and the management of the machines, both physical and virtual.

In this scenario, when a physical or virtual machine has a faulty behaviour, for any reason, both hardware and software, the monitoring component should discover it. It should find out which

virtual machines where running on that specific host, and then start the procedure to migrate, in case they are still running, or boot up these machines to another host(s), if the execution has failed.

As seen in the Figure 1, a separate network, possibly Gigabit Ethernet, is needed, in order to be used for storage and monitoring traffic.

2.2 Components

The presented architecture needs the following components:

2.2.1 Hypervisor

The hypervisor that is going to be used is Xen. This *paravirtualizer*, developed as Free Software by the Cambridge University –and now by the spin-off Xensource– has been chosen as it presents a very close performance with the native system [4].

On the Xen jargon, a host and a guest system are called *dom0* and *domU* respectively. Please note that through the rest of the article the terms *virtual machine*, *VM*, *guest* and *domU*, and the terms *physical machine*, *PM*, *host* and *dom0* are going to be used without any distinction.

2.2.2 Network storage

On the schema shown on Figure 1 the virtual machines are loaded from a network shared storage, allowing the loading of any domU from any dom0.

This is a really important component in our approach, as the idea is based on the possibility of loading any guest system on whatever host system. Thus, finding a reliable solution with a good performance will be one of the main goals on the testing phase.

There are several technologies that could be used:

Filesystems In this case the storage would be based on the use of any network-based filesystem. More precisely it should be one of the so called "shared filesystems" –GFS, GPFS, OCFS2, etc– or of the "distributed filesystems" –SMB, NFS, AFS, v9fs, Coda, GlusterFS, etc–.

The filesystems listed above differ greatly on their characteristics, performance and fault tolerance; for example GlusterFS is a distributed filesystem, while NFS is a client-server one.

Remote block devices The remote block devices are based on the use of some protocols (iSCSI, ATA over Ethernet, Fibre Channel, Infiniband) to export block devices over a network from a host called *target*; then the block devices will be imported and used as local devices by a host called *initiator*. The protocols are mainly based on the use of computer bus protocols, like SCSI and ATA, over network protocols, like TCP/IP, Ethernet, etc.

There could be both hardware based solutions, in which a specific hardware is designed and attached to the network, or software based ones in which a generic hardware is used –i.e. a generic fileserver–, with a software that implements the protocol and enables the hardware to behave as a "target". This second solutions makes possible to use the existing hardware without the need to buy a specific solution. Some of the undergoing tests are presented on Section 3.2.

2.2.3 Monitoring

The monitoring component should be able to discover problems due to services, network and machines –either physical or virtual– and start some recovery mechanism according to the problem discovered.

For example if there is a service failure (which usually is caused by a software problem), the system should just try to reload that service. Then, if it is not able to recover, it should inform the network administrators.

Another case is when there is a hardware failure, causing not only a problem in some services, but also on the physical machine and to the domUs that where running on top of it. In this case, the recover mechanism should consist on the following steps:

- ignore the virtual machines errors, as they are produced by the physical machine failure;
- discover which virtual machines were running on the physical machine;
- get sure that these machines are no longer running on the original host;
- look for another dom0 (this could be just one host, or more than one if there are involved several VMs) and start the domUs on the new dom0.

This component could be designed and implemented ad-hoc or could be based on some existing solutions. After an initial study phase we have chose a solution based on the following software packages:

Nagios: [16] It is a network, host and service monitoring tool which provides checks, notification mechanisms, error handling and notification. It is a widely used and reliable tool, which is also highly customizable, as the checks are performed by plugins –both official and third-party ones.

One important feature of Nagios is the possibility to define different check policies for different machines, in order to allow different monitoring priorities.

Cfengine: [17] It stands for *Configuration Engine* and is a tool used to manage the configuration and administration of large computer networks and clusters of computers. It is based on a bunch of configuration files written in its own high level language, which define the "ideal" state of a given host. Then, these configuration files are compared with the actual machine's state and, if there are some deviations, Cfengine tries to solve them. Cfengine also provides mechanisms to distribute these files that are modified, when needed, on the healthcheck host, then distributed to the Cfengine hosts.

GHAX: Under this name it is enclosed a number of custom Phyton programs, developed by our group, that glue together Nagios, Cfengine and Xen, enabling them to implement the healthcheck mechanism described before. The architecture of this part is implemented in a modular way, easily allowing the substitution of a component with a better one.

3. Preliminary Tests

In order to provide a stable solution we have extensively tested the components of the prototype.

3.1 SW tests

As a first step we have tested the software packages that are involved in our approach: Nagios, Cfengine and Xen. Whilst the first two packages are going to be tested more intensively on Section 4.1 a note on the preliminary Xen tests should be introduced here.

3.1.1 Xen

We have tested Xen, first on its version 2, then on its version 3, with different systems and architectures, in both the dom0s and the domUs

dom0 We have tested both NetBSD and several GNU/Linux distributions (such as Debian, Gentoo, Slackware, Ubuntu, Scientific Linux 4); several kernels (all of them on the 2.6 branch) have been tested, either using the pre-compiled binary packages provided by Xen than the source code.

The problems noted on the first versions of Xen (mostly dependent on distributions specific issues) have been solved on the current release of Xen. As for now, all the major GNU/Linux distributions provide binary Xen packages ready for the deployment.

It should be noted that NetBSD is the only operating system outside Linux that provides Xen support.

domU Also on the domUs side several distributions have been tested, but the work here was focused on Scientific Linux 3 and 4, as they are the distributions for which the LCG middleware is being released; actually the production release of the middleware is available for Scientific Linux 3, but it is intended to be released for Scientific Linux 4 in the second half of 2007.

Not only the vanilla operating system for the domUs has been tested, but also the operating system with the LCG middleware installed. In all cases no specific problems have been found in the virtualized LCG components, (Computing Element, DPM and dCache Storage Element, Worker Nodes, User Interfaces, etc). A part from specific kernel issues (as said before there is the need to install a modified kernel) no other problems have been found.

3.2 HW tests

As said on Section 2.2.2, the storage component is an important element of the system. It has been put under extensive tests, in order to get a reasonable comparison between the different solutions to help deciding which is the best suitable technology for the final solution.

The tests presented in this work are based on the iSCSI and ATA over Ethernet (AoE) protocols, both of them using its software implementation, instead of the hardware one. We have not only tested the performance of these systems, but also the compatibility between the client software (called *initiator*) and Xen.

3.2.1 Fibre Channel

The Fibre Channel is a gigabit-speed network technology primarily used for storage networking that can run on both twisted pair copper wire and fibre-optic cables, with full duplex capability links. Several topologies are available: point-to-point, fabric, arbitrated loop, etc. [10].

3.2.2 GNBD

GNBD stands for *Global Network Block Device* and provides block-level storage access over an Ethernet LAN. GNBD components run as a client in a GFS node and as a server in a GNBD server node. A GNBD server node exports block-level storage from its local storage (either directly attached storage or SAN storage) to a GFS node [9].

3.2.3 AoE

AoE stands for *ATA over Ethernet* and it consists on the use of the ATA storage devices over an Ethernet network, as defined by [11]. It does not use any other network layer above the Ethernet protocol, thus is not routable. Both hardware and software solutions are available. The software implementation used (and the only one existent) is the one from Coraid Inc. [20], which is also integrated in the Linux kernel tree.

3.2.4 iSCSI

The iSCSI[6] protocol, defined by the RFC 3720 [8], is a transport layer protocol that allows the use of SCSI devices, mapping their commands over TCP/IP networks, with support integrated in the Linux kernel tree. There are several GNU/Linux (and other operating systems) userspace tools to manage the iSCSI devices.

3.2.5 Results

Some of the results obtained are presented in Figures 2 trough 6, and are commented below. The tool utilized to perform those tests were IOzone [18] and Bonnie++ [19]. All the test were made using the same fileserver, provided with Gigabit Ethernet network cards and using an isolated network segment; for the tests with the Fibre Channel hardware also a Fibre Channel interface was added to the fileserver.

Only the most relevant results will be presented here. The first result obtained is the independence of the performance from the record size, as can be seen in Figure 2 obtained for the iSCSI case; similar results have been obtained for all the other cases as well as for the read performances.

As expected, the best results are obtained for the hardware solution, i.e. the Fibre Channel one, as can be seen in Figure 3 where are reported the results for a single Virtual Machine for the different options as a function of the file size. If instead we compare the only software options, the iSCSI solution is the best choice.

One of the important tests is the measurement of the performance when more than one Virtual Machine with the Operating System on the remote storage, is active. Even in this respect iSCSI is the preferred software solution, as can be seen for example in Figure 4 where the throughput for a test made with a 512MB file is reported for iSCSI and AoE as a function of the number of

[6] *internet* ISCSI

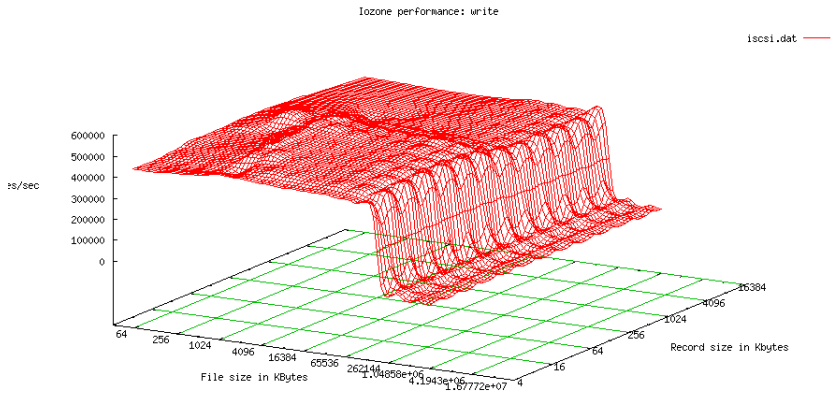


Figure 2: iSCSI write performance as a function of file size and record size

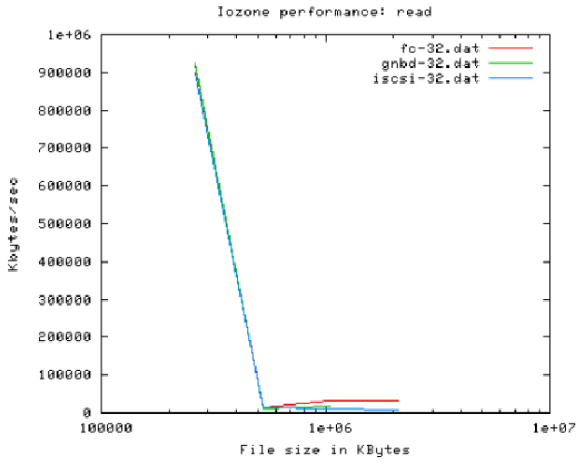


Figure 3: Comparison among FC, iSCSI and GNBD read performance as a function of file size.

Virtual Machines. The result is similar for both solutions for the *write* operations, while for the *read* operations iSCSI is clearly superior than Ata over Ethernet (almost a x5 factor).

When using Ata over Ethernet, we have experienced several problems –i.e. physical machine freezing and weird behaviour– when using a device directly to boot a domU using Xen. The literature consulted suggested some problems within the Xen networking support and the Ata over Ethernet drivers, not resolved as for now. The solution we have found to solve these problems is the use of LVM[7] between them. With its use we introduce a layer between Xen and the device, so Xen is no longer dealing with the *real* block device, but with a virtual one. Also, LVM provides with the mechanisms to manage the domU's images –shrinking/growing, deleting, etc.

On the other side, AoE doesn't need an special configuration, which makes its deployment an easy task whilst loosing the powerfulness of iSCSI, which needs more time and effort to get configured.

[7]Logical Volume Manager

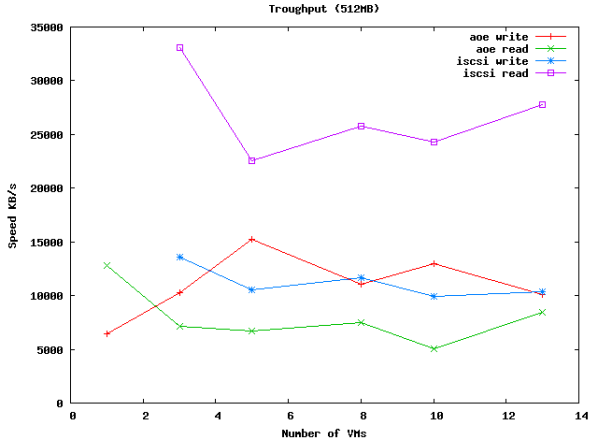
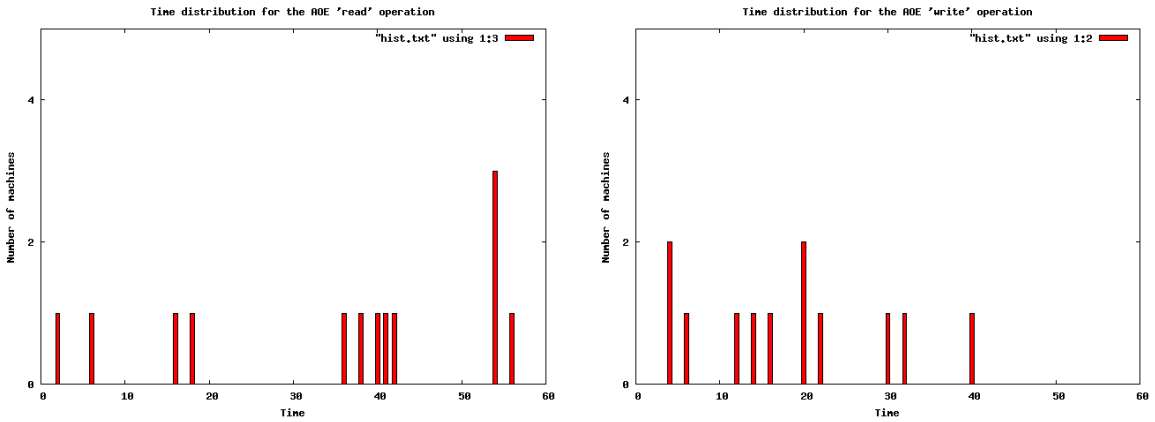


Figure 4: Throughput

Probing in a deeper way the difference between AoE and iSCSI, for example looking at the distribution of the time needed to finish the test on the different VM, as reported in Figures 5 and 6, it is evident that AoE has a very irregular behaviour, because the time needed to finish the same test varies from 2 minute to 2 hour in some cases, while the iSCSI is more homogeneous. Apparently these irregularities are due to a software problem of AoE, because the first 8 VM are faster than the remaining ones, independently from the physical machines where they are running.



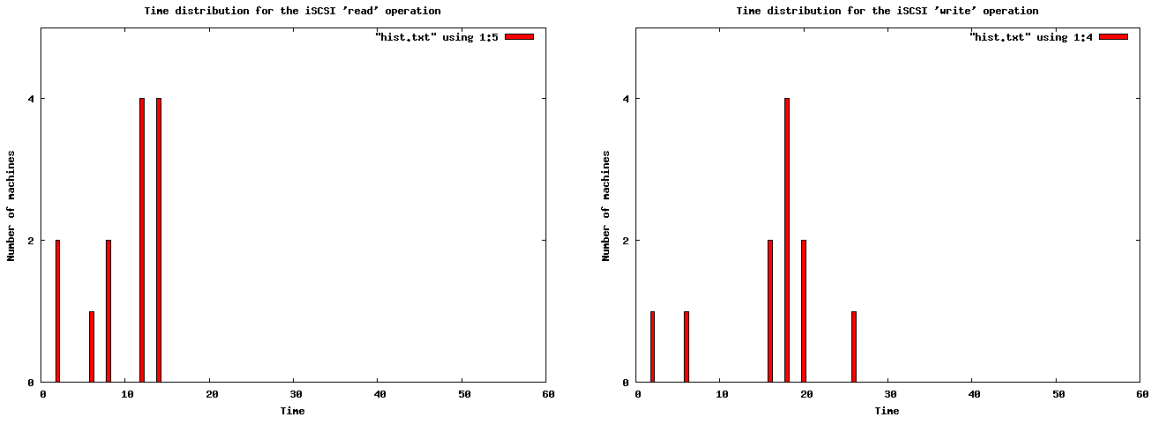
(a) read operation (b) write operation

Figure 5: Time distribution for AoE tests

As a conclusion, the iSCSI solution looks as the most stable and reliable one.

4. HA Prototype

Based on the architecture explained before, we have deployed the prototype shown on Figure 7. This prototype consists of several physical machines each with one or more virtual machines. On top of them there were deployed several LCG Grid components to form a small Grid site: a Com-



(a) read operation (b) write operation

Figure 6: Time distribution for iSCSI tests

puting Element, a Storage Element and several Worker Nodes. The number of virtual machines per physical machine was varied during the different tests, as well as the number of Worker Nodes.

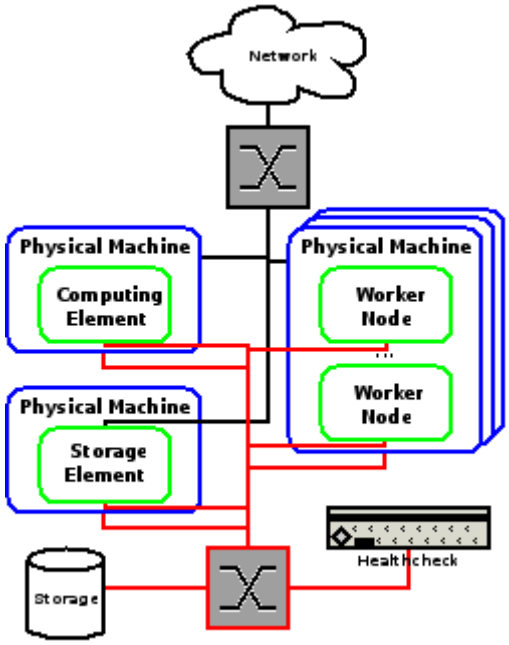
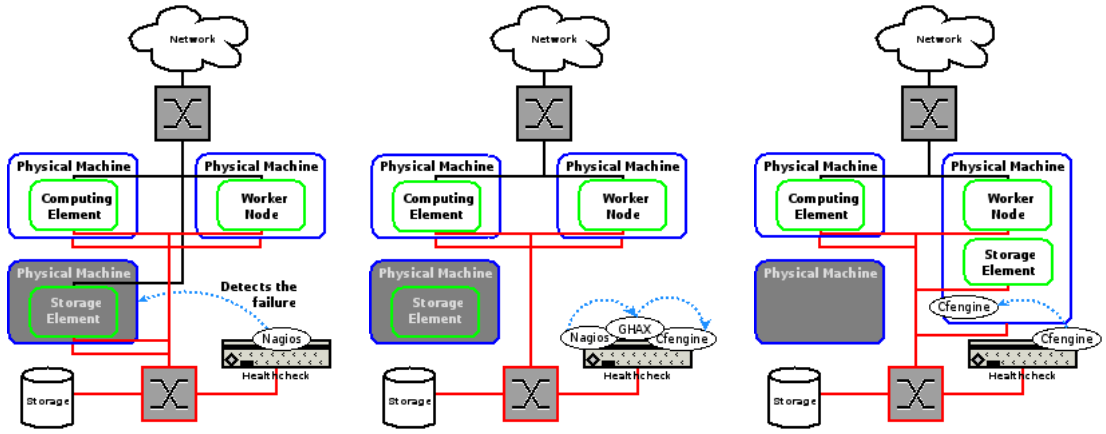


Figure 7: Prototype testbed

The healthcheck host checks the services/machines using Nagios. When a failure is detected, an agent is invoked and decides what to do: if the faulty component is a service or a virtual machine, it tries to reboot it invoking Cfengine on the remote host. Instead, if it is a host problem, then the healthcheck component will (Figure 8):

- Look for the virtual machines running on the given host.

- Disable the checks for these dependent virtual machines.
- Look for another physical machine –or more than one if there are more than one VM involved– where to boot the virtual machines.
- Modify the dependencies on the Nagios and Cfengine configuration files.
- Invoke Cfengine on the invoked (physical) machines:
  - Cfengine gets the new files from the healthcheck node.
  - Detects that the host has new dependent virtual machines, that are not running.
  - Boots the new machines.



(a) PM-3 failure (b) VMs migration (c) Recover

Figure 8: Workflow example

4.1 Prototype test

Two type of tests have been performed on the prototype:

Performance tests, to measure the time needed to recover from a failure: Nagios makes possible to define certain time intervals in which perform the checks for both the hosts and the services. We have defined the following intervals:

- Critical machines: 20sec;
- not so critical machines: 40sec;
- no critical machines: 60sec.

As reported on the Table 2, for the case of a single VM, the minimum time measured is 50 sec., while for multiple VM, an average delay of about 30 seconds per virtual machine is needed, which is almost the time needed to boot a Linux box.

	Critical	Not-so-critical	Not critical
1VM per PM	50 sec	1 min 10 sec	1 min 30 sec
5VM per PM	2 min 30 sec	3 min	3 min 30 sec

Table 2: Average times needed to solve a physical machine problem

Efficiency tests, to measure the percentage of jobs lost before a CE recovers when is artificially killed: We have launched a certain amount of jobs (in the order of 3000) with a fixed duration (45 minutes), first one by one and then –when the queues were saturated– all together. Then, simulating at random time intervals between 1 and 10 minutes the shutdown of the CE we have measured the amount of jobs for which its execution failed or its the output didn't return to the CE. The batch system chosen is the Maui+Torque combination.

During the tests performed we have found that just one job was "undelivered". This was the expected behaviour, as the CE downtime is really low (as it is on the *critical machines* group defined on the Table 2) and Torque provides a mechanism to retry the delivery of the job to the CE, being the time between attempts is lower the downtime measure.

5. Conclusion

In this work we have presented a reliable solution to obtain Highly Available systems, with the accent on GRID services. One of the goals reached by our work is the fully –or almost fully – exploitation of the resources, allowing to reduce to a small number the spare machines. Also we focused on the easiness of the machines management, providing tools to automate the work that usually should be done by a person. By using virtualization, the virtual machines created are going to be almost clones, so the management of these machines (even if deployed on heterogeneous hardware) is an easy task

The downtimes measured –Section 4.1– was of the order of 1-2 minute, for the various configurations, which is an acceptable downtime for Grid services. Also, this work opens the doors to several workfields, in which the experience gained during it could be applied. These would be discussed on the next Section 6.

6. Future plans

The prototype implemented opens the field to several developments and research lines in the near future.

XenAPI Integration: With the last releases of Xen (at the time of the writing of this article, Xen 3.1) the Xen API project is taking shape as an XML-RPC protocol for remote and local management of Xen systems.

At the moment, it is a project under heavy development and modifications and with a great lack of documentation; but it should be taken into account for the next future, as it would simplify the way in which the management of the virtual machines is done, and is should become the standard way to manage Xen systems.

Quattor integration: To provide an easy way to deploy this solution on the LCG sites, works on the Quattor software are under development. Also the integration with other Scientific Linux tools (i.e. apt-get and yum) is under development.

Dynamic computing environments: The automatic and on-demand deployment of customized virtual machines is an interesting subject, to develop kin the next future, taking advantage of the existing work. This could solve the problem of the different Virtual Organizations working on the Grid that have different software requirements, which, sometimes, could be also incompatible between them. One possible solution could be the implementation of *Dynamic computing environments*: There is a pool of different types of virtual machines, each one satisfying different requirements. Then, when any job arrives, *n* virtual machines satisfying the requirements are deployed on top of some physical machines.

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 2004.

[2] I. Foster, C. Kesselman and S. Tuecke, *The Anatomy of the Grid*, `cs/0103025v1`

[3] L. Boyanov, P. Nenkova *On the employment of LCG GRID middleware*, in proceedings of *International Conference on Computer Systems and Technologies*.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, A. Warfield, *Xen and the Art of Virtualization*, in proceedings of *Nineteenth ACM symposium on Operating systems principles*, ISBN: `1-58113-757-5`.

[5] G. J. Popek, R. P. Goldberg, *Formal requirements for virtualizable third generation architectures*, *Communications of the ACM* Volume 17, Issue 17.

[6] G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig, *Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization*, *Intel Technology Journal* Volume 10, Issue 3.

[7] *Putting Server Virtualization to Work*, AMD Whitepaper no. 32951.

[8] J, Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka and E. Zeidner
RFC 3720: Internet Small Computers System Interface
`http://www.ietf.org/rfc/rfc3720.txt`

[9] `ftp://sources.redhat.com/pub/cluster/releases`

[10] `http://www.fibrechannel.org/`

[11] S. Hopkins and B. Coile
ATA over Ethernet protocol description
`http://www.coraid.com/documents/AoEr10.txt`

[12] LHC Computing Grid
`http://cern.ch/LHCGrid/`

[13] *INFN* - Italian National Institute of Nuclear Physics
`http://infn.it`

[14] *INFN* Computing Grid
`http://grid.infn.it`

[15] Xen virtual machine monitor
`http://xen.sourceforge.net`
`http://www.xensource.com`

[16] Nagios host and service monitor
`http://nagios.org`

[17] Cfengine: an adaptive system configuration management engine
`http://cfengine.org`

[18] Iozone Filesystem Benchmark
`http://www.iozone.org`

[19] Bonnie++ benchmark suite
`http://www.coker.com.au/bonnie++/`

[20] Linux ATA over Ethernet implementation
`http://www.coraid.com/support/linux/`