IT and Applied Mathematics Department

Technical Project 2008

VIRTUALIZATION FOR AUTONOMOUS ADMINISTRATION OF SERVERS

Presentation of Technical Project

Client supervisor : Daniel Hagimont - Professor INPT/ENSEEIHT - Daniel.Hagimont@enseeiht.fr

Client co-supervisor : Laurent Broto – Ph. D. student at UPS - Laurent.Broto@irit.fr

Industrial Supervisor : Emmanuel Murzeau - emmanuel.murzeau@airbus.com

Team Leader: Ezequiel Geremia - ezequiel.geremia@etu.enseeiht.fr

Students :

- Julien Louisy
- Julien Clariond
- Hery Randriamanamihaga
- Ezequiel Geremia
- Mathieu Giorgino

Summary

1 Abstract

Server administration research aims to develop efficient policies to manage clusters of servers. The virtualization paradigm offers a theoretical frame to handle operating systems on a hardware device as applications on an operating system. As a performing paravirtualization software, Xen provides a stable environment to manage virtual machines. Moreover, Xen migration commands can be used to move operating systems between several hardware devices. TUNe provides a high-level environment to perform autonomous software reconfiguration. The Xen AttiTUNe project experiments Xen and TUNe software components in order to implement server administration policies.

Keywords : virtualization, paravirtualization, autonomous server administration, Xen, TUNe, server administration policies.

2 Xen AttiTUNe project description

2.1 Objectives

Consider a company which owns a cluster of computers, *i. e.* many network connected computers which are able to work together. This company rents these computers to customers who need to process large amount of information. Each computer is able to host one single server which processes the client's data flow. One obvious strategy to maximize profits consists of renting all the computers.

However, several studies revealed that most regular customers do not use computers efficiently, *i. e.* the processor remains inactive for long-term periods because the hosted server does not process anything. These periods seem to be a very severe loss for the company. Indeed, the inactive machines could have been rent to other customers.

The Xen AttiTUNe project aims to build an architecture which minimizes the number of inactive computers. This architecture will enable to host active servers on inactive computers.Therefore computers should be able to host two servers : one active server and one inactive server. As a consequence, considering that the company owns *n* computers, if it rents *n* places for clients' servers among whom *m* are shown to be inactive, then *m* more places can be rented. The total number of rented places equals *n+m*. This kind of argument leads to conceive optimized strategies, *e. g.* overbooking the cluster of computers ensures that all computers are used efficiently.

However, if two active servers are running on the same computer, the customer may complain about the lack of performance on his servers. Indeed, one single machine cannot support two servers without slowing down the data processing on both servers. Therefore the company which rents the cluster of computers should ensure that it will always be possible to pool active servers with inactive servers. The Xen AttiTUNe project will enable to implement such cluster management policies. Figure 2.1.1 sums up further similar policies.

	Policy 1	**Policy 2**	**Policy 3**
Context	pay to use computers	high electrical consumption	highly distributed clusters
Metric	server reservation price	per computer electrical consumption	network load and distance between computers
Goal	minimize costs = bring together inactive servers	minimize electrical consumption = shut down inactive computers	minimize communication time = pool communicating servers in a local network

Figure 2.1.1. Administration policies depending on the context. *Each policy applies to a specific context, observed according to a certain metric, and achieves a goal. From customers' point of view, the first policy leads to minimize the cost of computer hire. The second one minimizes costs due to electrical consumption by saving energy. The third one speeds up communications between distant servers by pooling them into a local network.*

2.2 Requirements

Implementing the management policies presented above requires resolution of two functional issues. On the one hand, the final architecture should provide multiple server hosting on one single computer. On the other hand, such an architecture should implement an autonomous server administration.

In order to achieve multiple server hosting on one single computer, each computer should be able to host several operating systems. Yet each operating system needs the complete physical environment provided by a computer, including resources like processors, memory, hard drives and network interfaces. To overcome this problem, each operating system should be given virtual resources. Each physical resource should be shared out among many virtual ones.

In order to implement an autonomous server administration, the final architecture should allow to move a server from one computer to another within the cluster. This kind of move is called migration. Migrating the operating system hosting a server is not an obvious operation. Indeed, the execution of the server should resume transparently. As a consequence, the execution context , including the memory context, should be the same before and after a migration. As the memory context is managed by the operating system, the operating system and the server running on it should be migrated together.

Once operating system migration is shown to be context conservative, the associated file system distribution remains necessary. Indeed, data stored on hard drives cannot be migrated with its operating system because it would needlessly increase the network load. In fact, all file systems should be pooled on a data server and accessed through the network.

2.3 Project architecture

The following analysis discusses functional and conceptual means in order to satisfy the requirements presented above. The first section presents the virtualization design pattern. The second and the third section explains how the Xen tool satisfies resource sharing and migration functionalities. The fourth section explains how to use the NFS (Network File System) protocol to share file systems. Finally, the fourth section explains how to perform autonomous server policies with the TUNe administrator.

2.3.1 Platform virtualization design pattern

Sharing physical resources among several operating systems is a software design problem which can be resolved by the virtualization design pattern. Virtualization provides an interface to allow several applications to share the same hardware.

Platform virtualization is performed on a given hardware by a host software operating system, which creates a simulated computer environment, called virtual machine, for its guest operating system. The guest operating system runs just as if it were installed on a stand-alone hardware platform. Such virtual machines are generally simulated on a single physical machine and their number is limited only by the host’s hardware resources.

2.3.2 Xen related architecture

The Xen AttiTUNe Project is based on a paravirtualization tool called Xen. Xen based virtual machine does not need to simulate hardware. Instead, the guest operating system is modified to be able to use a special API (Application Programming Interface). That is why Xen is the fastest and most secure virtualization software available today, enabling every server to support multiple virtual servers each with resource guarantees.

With Xen virtualization, a thin software layer called hypervisor is inserted between the server’s hardware and the operating system. This provides an abstraction layer that allows each physical server to run one or more virtual servers decoupling the operating system and its applications from the underlying physical server. A Xen domain contains an operating system and its execution context but does not contain the associated file system. Xen offers interfaces to manipulate two kind of domains : host operating systems, called dom0 domains, and virtual, or guest, operating systems called domU domains. Figure 2.3.1 illustrates the abstraction of physical resources.

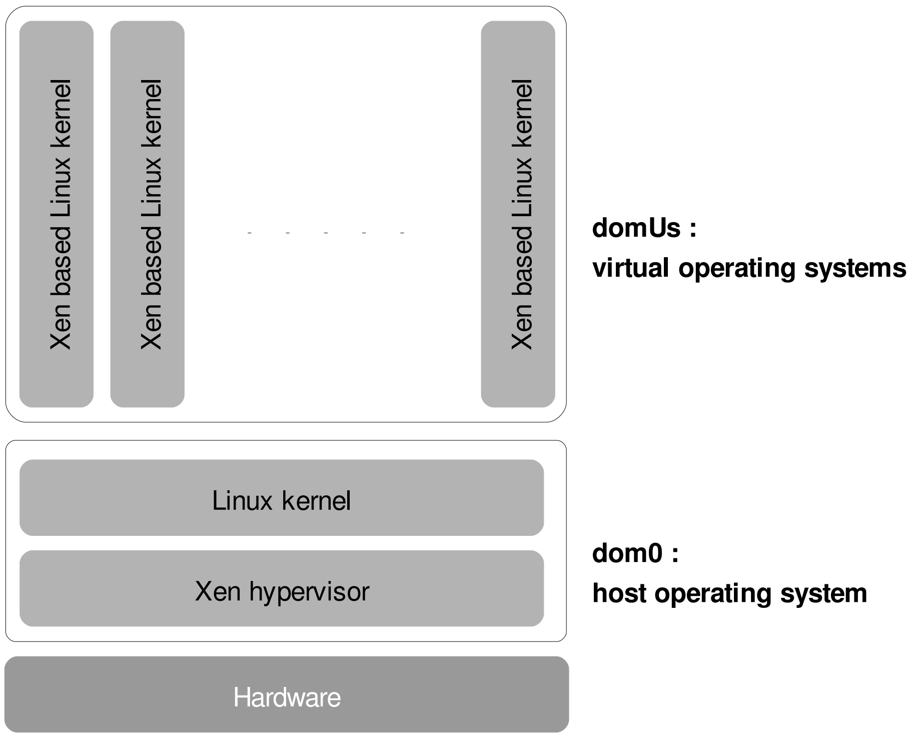


Figure 2.3.2.1. Hardware sharing through software abstraction. *The host operating system, called domain dom0, contains a Xen hypervisor to run one or more virtual operating systems, called domain domU. The hardware layer stands for the physical environment running the dom0 layer. It includes processor, memory and web interfaces. If a physical server implements the hardware layer, then the dom0 layer is a host operating system supporting the domU layer, which pools server applications.*

2.3.3 Xen domain migration

Migration is used to transfer a domU domain between dom0 hosts, *i. e.* virtual machines between physical hosts. There are two varieties : regular and live migration. The former moves a virtual machine from one host to another by pausing it, copying its memory contents, and then resuming it on the destination. The latter performs the same logical functionality but without needing to pause the domain for the duration. In general when performing live migration the domain continues its usual activities and, from the user's perspective, the migration should be imperceptible. Moreover TCP (Transmission Control Protocol) connections of migrated domains are not broken off. Figure 2.3.2 presents the migration of a server from one computer to another.

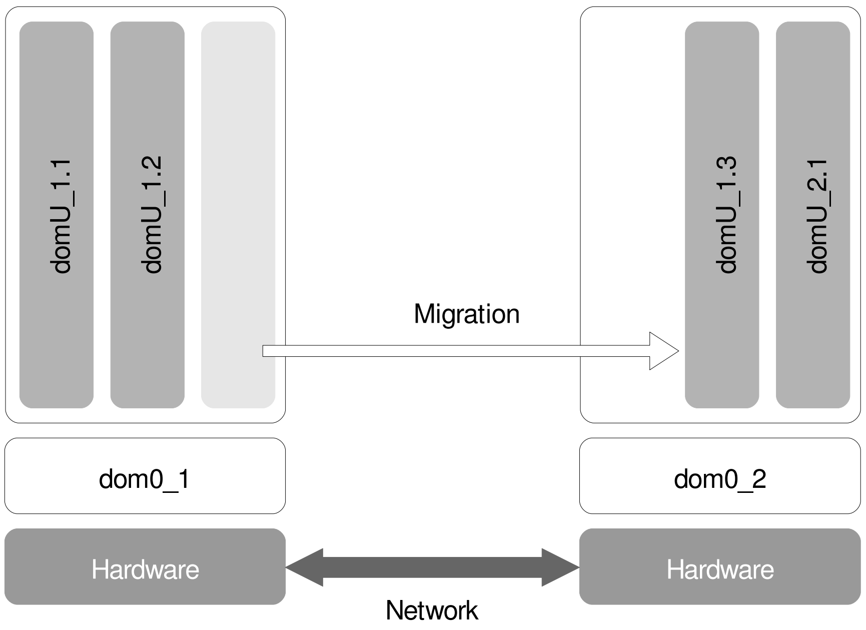


Figure 2.3.3.1. Migration of virtual domains between physical machines. *Initially, the first physical machine running dom0_1 hosts three virtual machines or domU, and the second one, running dom0_2, hosts only one virtual machine calles domU_2.1. The two hosts dom0_1 and dom0_2 are network connected. After the migration of domU_1.3, dom0_1 and dom0_2 host both two domU.*

2.3.4 File system distribution

Xen does not allow to migrate file systems. However the migrated domUs need to keep a reference to their file system to execute stored programs or to save data. Every domU can access its file system located on a distant NFS server. Therefore, the migration will appear transparent to the user because the execution context of the migrated domU remains exactly the same. Figure 2.3.3 shows how file systems should be distributed among the network.

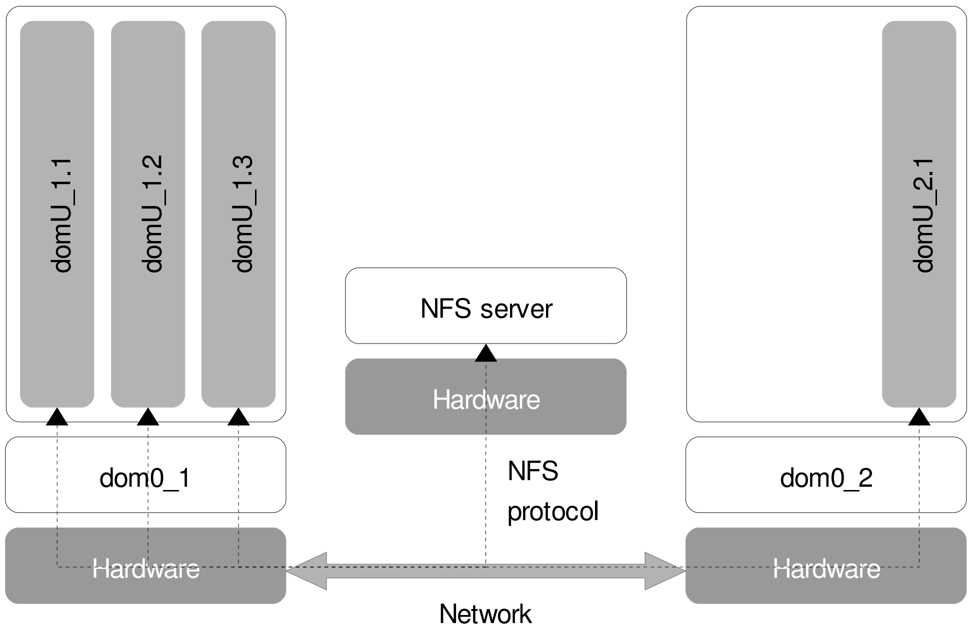


Figure 2.3.4.1. NFS server as file system distributor. *Every domU accesses the NFS server through the network in order to recover its file system after migration.*

Physical architecture should include an NFS server, a DHCP (Dynamic Host Configuration Protocol) server to handle the local network connect the physical machines.

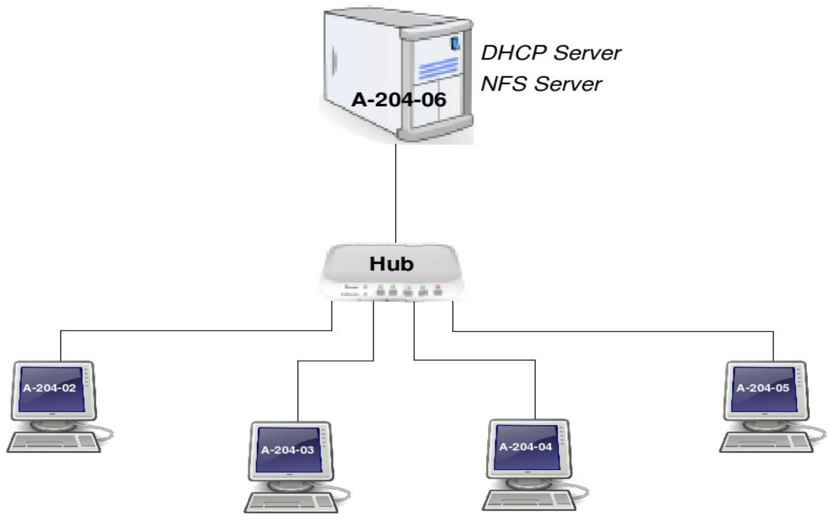


Figure 2.3.4.2. Server cluster architecture. *The network is implemented by a hub and a DHCP server. Four physical are network connected. The DHCP and NFS server are hosted on a single physical machines. Each network component is identified by its specific location.*

2.3.5 Server administration with TUNe

Distributed software environments encounter increasing complexity and management difficulties. Such architectures integrate various legacy software with specific management interfaces and need human-performed management tasks. Several configuration errors and low reactivity are the major consequences. TUNe (Toulouse University Network Environment) offers an implementation of a global autonomous management system, designed to solve these problems. The main idea is to administrate a software infrastructure as a component architecture, by wrapping legacy software pieces in components. Moreover, a high-level formalism has been introduced for specifications of architecture deployment and for autonomous management policies.

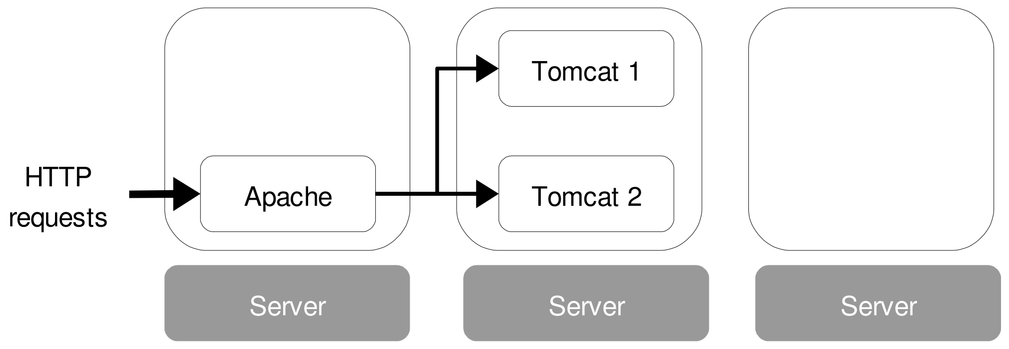
Integrating TUNe in the project leads to several reconsiderations of the current architecture. The first idea is to wrap entire virtual machines into TUNe components and use reconfiguration diagrams to lead migrations. However, it will be necessary to conduct a study in order to validate or invalidate such a choice. Results will define a more precise architecture which will be used until the end of the project.

The next step will be a simple deployment of a defined system based on the final client request. One Apache web server and two Tomcat application servers will be launched on a TUNe architecture. The idea is to check whether TUNe enables migration or not. If it does not, a phase of specification and resolution of the issue should be considered.

Finally, an autonomous policy administrating live migrations will be implemented to satisfy client demand. As CPU consumption will be the determinative parameter, TUNe probes will monitor system activity and therefore their implementation will be added to the final tasks.

Basically, this scenario should show a migration monitored by the physical machine CPU rate. The Apache/Tomcat architecture defined below is reused: one Apache and two tomcats are hosted on a single physical machine. The Apache server distributes the requests to the two Tomcats. When request number rises, one of the Tomcats moves to another physical machine. When request number decreases, the Tomcat moves back to its previous host.

First step :



Second step :

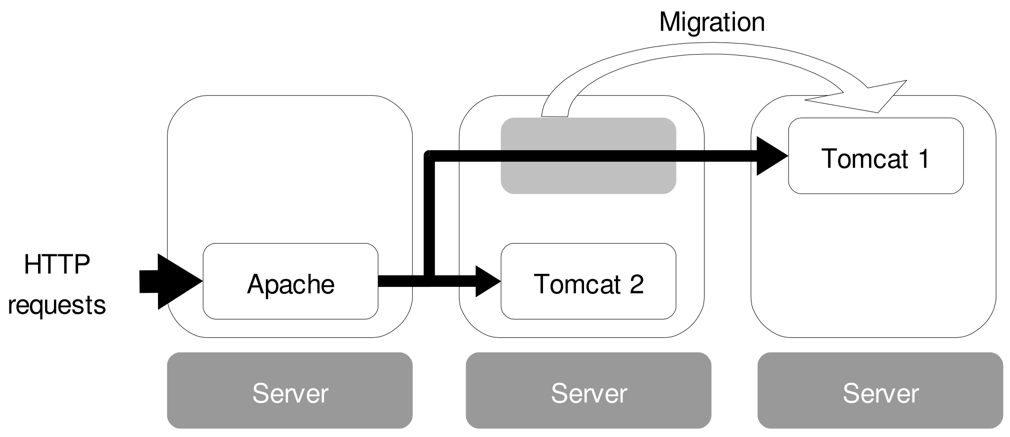


Figure 2.3.5.1. Autonomous server administration policy. *Initially, each Apache and Tomcat is wrapped in a TUNe component. The two Tomcats are on a single server and receive the same number of HTTP (HyperText Transfer Protocol) requests. If the number of HTTP requests increases then the TUNe component hosting one of the Tomcats should be migrated to another servre. Finally, the two Tomcat applications run on separate servers to ensure efficient processing of entering requests.*

3 Role distribution

The project has been divided into three parts: first of all a study of Xen, its architecture, mechanism and performance, then an integration in TUNe and finally set up a test scenario. The team is currently working on the second part. Work distribution has not been an easy task as this project is more a matter of reflection than of coding: team brainstorming is an unavoidable, systematic and efficient way to reach a group agreement.

3.1 Xen study

The first step was to study Xen API to be able to build up a physical architecture that would suit the needs of the project. This has been a collective task as everybody in the group had to fully understand how Xen works.

At the same time, Ezequiel and Julien L started a preliminary study of a Xen installation on the computers provided by the CCRI. It lasted about three days and ended with the working architecture described below (cf. section 2.4.4).

The next step was to deploy this architecture on each machine and write different kinds of documentation: specification and architecture sheets, risk analysis, action list, test plan, schedule and various tutorials (how to install Xen, how to build an NFS server, how to set up a DHCP server). The whole group has worked on these last points so that everybody has experimented both project management and Xen installation and utilization.

The final step was to develop the tests described in the test plan sheet. As it was obvious that the different tests could be run separately, the group has been divided in two according to the development difficulty of each test: Julien C and Hery were assigned to the two most complicated tests and Ezequiel, Mathieu and Julien L worked on the five less hard to implement tests.

At this point of the project, a functional architecture has been built. The team's main goal is now to integrate this architecture into TUNe.

3.2 Integration in TUNe

The very first step will be to learn how to use TUNe. Mr Hagimont has given a presentation on it and the whole team is currently working on the API in order to study how the integration will be done.

The team will then have to agree on a working TUNe architecture, update the architecture sheet and deploy the simple example described below (cf. section 2.3.5). All the members of the team will probably work together on this part as it is more likely to be a brainstorming than a coding phase.

The final step will be to measure the new architecture performance (compared to the previous one) by applying the tests developed in the Xen Study part. The team will be divided again into the two traditional groups to perform this test bench in order to save as much time as possible.

3.3 Validation scenario

This is the very final part of this project. The goal is to have a working scenario summarizing all the work that has been done. This scenario is described in section 2.3.5. The precise distribution of work has not been decided yet, but roughly, the coding part will be assigned to three people and the testing part to the rest of the group. The result will bring the project to its maturity and provide the client what he is expecting.

4 Bibliography

- *Xen User's Manual.* University of Cambridge. 5 December, 2005. <http://tx.downloads.xensource.com/downloads/docs/user/>.
- Xen Networking. 14 September, 2007. <http://wiki.xensource.com/xenwiki/XenNetworking>.
- Ubuntu documentation : Xen. David Tangye. 16 January, 2008.<https://help.ubuntu.com/community/Xen>.
- Debian administration : Booting Xen 3.0 guests using NFS. Steve Kemp. 8 Mars, 2007. <http://www.debian-administration.org/articles/505>.
- TUNe: Toulouse University Network (Middleware). Thierry Monteil. 10 September, 2008. <https://www.grid5000.fr/mediawiki/index.php/Special:G5KExperiments?menu=renderexperiment&id=444>.