UNIVERSITE DE YAOUNDE I
****
ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE
****
DEPARTEMENT DE GENIE INFORMATIQUE

UNIVERSITY OF YAOUNDE I
****
NATIONAL ADVANCED SCHOOL OF ENGINEERING
****
DEPARTMENT OF COMPUTER ENGINEERING

OPTIMIZED RESOURCE ALLOCATION FOR THE PRIVILEGED DOMAIN IN SERVER VIRTUALIZATION

Applied to the domain 0 in the Xen virtualization system

End of course dissertation/Master of Engineering

Presented and defended by

NOM PRENOM

In partial fulfilment of the requirements for the award of a:

Master of Engineering in Computer Science

Under the supervision of:

XXXX xxxx, PROFESSOR, NATIONAL POLYTECHNIC INSTITUTE OF TOULOUSE

XXXX xxxx, ASSOCIATE PROFESSOR, NATIONAL POLYTECHNIC INSTITUTE OF TOULOUSE

XXXX xxxx, ENGINEER, TOULOUSE COMPUTER SCIENCE RESEARCH INSTITUTE

In front of the jury composed of:

President: **XXXXX XXXXX, ASSOCIATE PROFESSOR, UNIVERSITY OF YAOUNDE I**

Examiner: **BERNABÉ BATCHAKUI, SENIOR LECTURER, UNIVERSITY OF YAOUNDE I**

Academic year 2016-2017
Defended the 08th September 2017

Institut de Recherche en Informatique de Toulouse

OPTIMIZED RESOURCE ALLOCATION FOR THE PRIVILEGED DOMAIN IN SERVER VIRTUALIZATION

Applied to the domain 0 in the Xen virtualization system

End of course dissertation/Master of Engineering

Presented and defended by

XXXX XXXXX

In partial fulfilment of the requirements for the award of a:

Master of Engineering in Computer Science

Academic year 2017-2018
Defended the 08 July 2018

Dedication

To my parents

Acknowledgements

This work is the culmination of many efforts and sacrifices and would never have been accomplished without the help and support of:

- **XXXX xxxx**, Associate Professor at UY1, who has made us the honor of presiding over this jury;
- **Bernabé BATCHAKUI**, Senior Lecturer at UY1, for agreeing to examine this work and for his remarkable dedication to teaching;
- **Alain TCHANA** and **Daniel HAGIMONT**, respectively Associate Professor and Professor at INPT, members of the IRIT laboratory

Glossary

Benchmark In computer science, a benchmark is a test to measure the performance of a system to compare it to others. 3

Bus In computer science, a bus is a communication system that transfers data between components inside a computer, or between computers.. 9

. 8

. vi, 1, 2, 10, 17, 19, 21, 27, 28

. 1, 11, 34

Interrupt An interrupt is a signal from a device attached to a computer or from a program within the computer that requires the operating system to stop and figure out what to do next. 12, 34

. 11, 34, 39

Open source The open source designation, or "open source code", applies to software whose license meets criteria precisely established by the Open Source Initiative, that is, the possibilities of free redistribution, access to source code and the creation of derivative works.. 1, 6, 9, 34

ABSTRACT

SNulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Keywords: Virtualization, Resource, Memory, Processor, Location.

Résumé

L Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Mot clés: Virtualisation, Mémoire, Ressource, Processeur, Emplacement.

LIST OF TABLES

LIST OF FIGURES

Table of Contents

INTRODUCTION

CONTEXT

SNam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

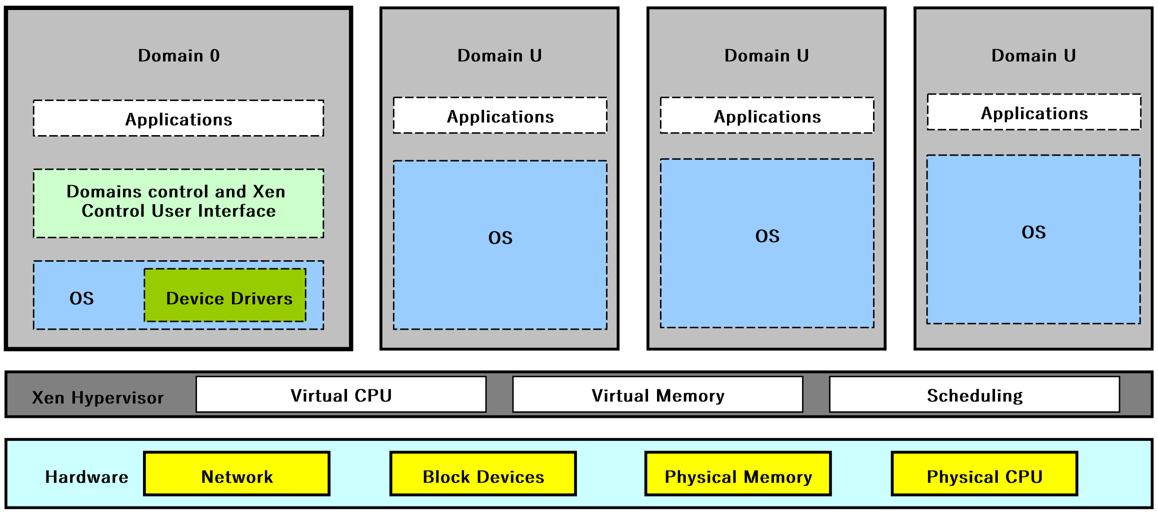

Figure 0.1: Simplified Xen Architecture

PROBLEMATIC

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Questions

How should resources be allocated to the dom0 and organised in aNUMA architecture ?

OBJECTIVES

The main goal of this work is to establish a **resource allocation strategy for the dom0** that will take into consideration the two fundamental questions mentioned above. Our resource allocation strategy should enable:

1. the dom0 to have exactly the amount of resources it needs at every moment,
2. the dom0 processes that work on behalf of a virtual machine use the resources of this virtual machine and
3. those processes must be executed as close as possible to the virtual machine for which they work.

To achieve this, we explored the issues associated with these two questions, analyzed the limitation of previous solutions addressed to this problem and compared our resulting model to the existing solutions on a series of benchmarks well thought out for the problem.

ROAD MAP

The rest of the document is structured as follows:

- **Background** in which we enrich our vocabulary and present some mechanisms in the Xen architecture for a better understanding of the work and what follows;
- **Problem statement and assessment** where we discuss about the questions raised in the problematic and prove it is relevant;

CHAPTER 1

Background

In this chapter, we present basic concepts such as server virtualization, the types of server virtualization systems, NUMA architectures and I/O mechanism in virtualized environments such as Xen for a better apprehension of the work done. The plan of this chapter is as follows:

Contents

1.1 Server virtualization

Most servers use less than 15% of their resources [**?**]. Server virtualization addresses these inefficiencies by allowing multiple operating systems to run on a single physical server as virtual machines, each with access to the underlying server's computing resources. To achieve server virtualization, we need a server virtualization system (hypervisor). There are mainly 4 types of server virtualization systems which are: **full virtualization, para-virtualization, OS level virtualization and hardware assisted virtualization**. Depending on the needs, one type may be better than another. In the next section, we discuss the different types of server virtualization systems and give some examples.

1.1.1 Types of server virtualization systems

1.1.1.1 Full virtualization

Here, the hypervisor is a software that runs on top of the host[1] OS as shown on Figure 1.1. The latter virtualizes/emulates the hardware for the guest OS (virtual machine), and is believed to communicate directly with the hardware. This solution is very similar to an **emulator**, and sometimes even mistaken for the latter. However, the CPU, the RAM, as well as the storage memory, are directly accessible to the virtual machines, while on an emulator the CPU is emuled, and the performance is considerably reduced compared to virtualization.

This type of server virtualization is offered by software such as **Virtualbox, VMWare Workstation.** Hypervisors enabling this type of server virtualization are said to be of **type 2**.

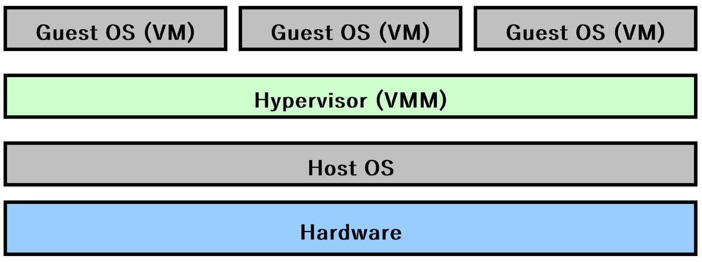

Figure 1.1: Full virtualization architecture

1.1.1.2 OS level virtualization

Here, the hypervisor is called an **isolator**. An isolator is software that isolates the execution of applications in **contexts**, or **execution areas** as shown on Figure 1.2 . The isolator thus makes it possible to run the same application several times in a multi-instance mode (several instances

[1]Here we refer to the physical machine

of execution) even if it was not designed for that. This solution is very efficient, because of the **small overhead**, but the virtualized environments are not completely isolated.

There is a visible performance. However, we can not really talk about virtualization of operating systems. Only connected to Linux systems, isolators are actually made up of several elements and can take many forms. This type of virtualization is offered by software as **Linux-Vserver, Chroot, BSD Jail, OpenVZ and LXC (Linux Container)**.

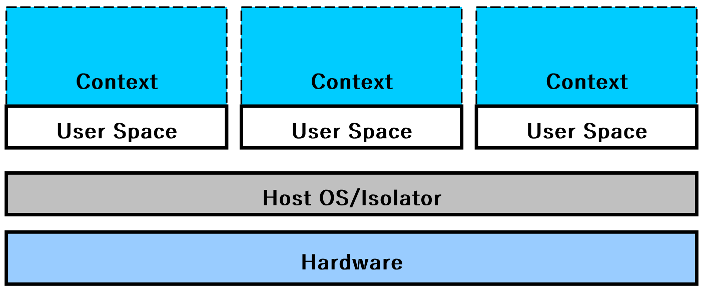

Figure 1.2: OS level virtualization architecture

1.1.1.3 Hardware assisted virtualization

This is a particular case of server virtualization where the hardware is designed to ease virtualization. Here the virtual machines are called **Hardware Virtual Machines (HVM)**. The CPU designers such as **Intel** and **AMD** modify the hardware to enable technologies such as **Intel VT** and **AMD-V** which enables material assisted virtualization. Hypervisors such as **VMWare vCenter, Xen and** supports material assisted virtualization.

1.1.1.4 Paravirtualization

Here the hypervisor replaces the host OS and acts as an intermediate between the guest and the hardware. The host OS is considered a privileged virtual machine and used by the hypervisor to complete a set of tasks as shown in Figure 1.3. However, this type of server virtualization requires modification of the guest OS to be virtualized for it to support **system calls** from the hypervisor. Paravirtualization offers better performance than the other types of server virtualization even though the modification of the guest OS is not always possible. For example, it is not possible to modify the **Windows** kernel for its designer does not allow it since it is not open source.

Hypervisors enabling this type of virtualization are said to be **type 1**. Some examples of these hypervisors are **Xen, VMWare ESX**.

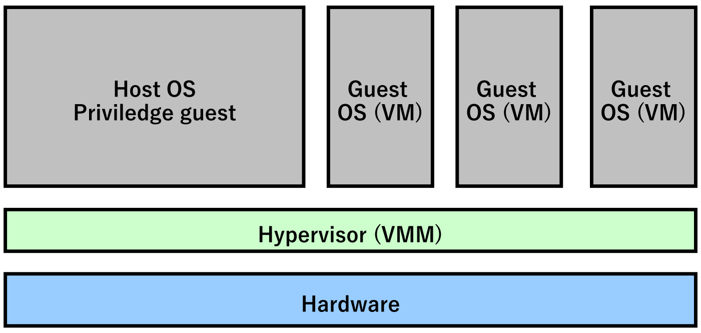

Figure 1.3: Paravirtualization architecture

1.1.2 Advantages of server virtualization

Server virtualization is not just for big companies. s can also take advantage of this technology, the advantages of which are presented here.

Reduced physical servers. The first benefit of server virtualization is that since many virtual machines can run on a single physical server, the number of servers to buy and maintain will be reduced. With a traditional infrastructure, many servers are oversized in order to cope with possible peak loads. Combining multiple virtual machines in one place, a virtualized server is better exploited. The total number of machines required for the smooth functioning of the can thus be revised downwards.

Better availability. Server virtualization solutions now allow the migration of virtual machines which is the process to move a virtual machine from one physical server to another without even needing to stop it [**? ?**]. This feature is a key element in improving the availability of services. The use of two physical servers thus makes it possible to easily double a virtualized infrastructure (redundancy). In case of a failure of one of the two servers, the virtual machines will be automatically moved to the second.

Better performance. Another advantage of hot-migrating virtual machines between physical servers is that they can be used to distribute the workload between servers [**? ?**]. When a virtual machine rises to extreme load, others will be able to move to a less demanding physical server. Critical tasks will also work within a virtual machine that has more virtual CPU, virtual RAM and virtual hard disk cores than the others. It is thus possible to modulate the size of the virtual machines according to the tasks they will have to perform.

Security enhanced. In a traditional / infrastructure, , file sharing, the mailing system, or even the web server, all run on the same server. However, if the mailing system is infected with a malware, all applications hosted on the machine are endangered. Server virtualization can be used to separate the different tasks of a physical server into many separate virtual machines, which will then be isolated from each other, thus dividing the services.

Anti-obsolescence warranty. A part of an , for example, an , is likely to work on a dedicated server, due to a specific configuration. Ensuring the renewal of this dedicated server, which is used by only one application, is often not very profitable. tools make it possible to transform most physical servers into virtual machines. Once this is done, the virtualized server will be able to switch from an end-of-life machine to a new server by simply migrating the virtual machines.

Gain on licensing costs. One operating system license per server is required. However, virtualization sometimes allows you to take advantage of license packs covering the OS of the physical server and its virtual machines. For existing virtual machines, it will not be necessary to repay a license when the physical server running them is changed. A plus, which also applies to applications running within these virtual machines.

Simplified backups. With a virtualized infrastructure, the physical server is the only one physically present in the engine room. Virtual machines are pure software. This aspect greatly simplifies data backup operations. It is indeed possible to directly perform a backup of the contents of the virtual hard disk of a virtual machine. And even during its operation, by creating a snapshot of the virtual machine and its data. In case of a problem, this snapshot will restart the virtual machine in a previous state.

easier to manage [?]. Virtualization can simplify the , making it easy to implement complex recovery plans. For example, launching a database server before the server that accesses it.

Test without paying. Virtualization allows you to create a blank virtual machine in minutes. As long as your physical server does not display full, it will be possible to add new virtual machines to manage. Developers or system administrators will be able to exploit this feature to try new services without spending a single coin. Where a test server was previously needed, a simple virtual machine on an existing server of the company is enough today.

A stepping stone to the private cloud [?]. Server virtualization allows you to deploy new services within your computer system in the form of virtual machines. But also to dimension these virtual machines according to the criticality and the expected use. From there, why not offer a catalog of services and sizes of virtual machines to your business teams? And turn your into a service center.

1.1.3 Some server virtualization systems

In the table 1.1, we compare some server virtualization systems based on their characteristics.

Table 1.1: Comparison of virtualization tools based on different factors

Virtualization tools	License	Mode supported
Bosch	Open source	OS level
KVM	Open source	Full
LXC	Open source	OS level
Microsoft Virtual	Commercial	Full
Parallels	Commercial	Full
QEMU	Open source	Full
VMWare Vcenter	Commercial	Full
VMWare Workstation	Free (not open source)	Full
VNUML (Virtual Network User Model Linux)	Open source	Full
VServer	Open source	Para
Xen	Open source	Para

1.2 NUMA architecture

When using a server virtualization tool on a physical machine, for better configurations, it is essential to understand and master its architecture. Here we will try to give an outline of NUMA architectures and present its impact on the performance of applications.

For the past decade, processor clock speed has increased dramatically. A multi-gigahertz CPU, however, needs to be supplied with a large amount of memory bandwidth to use its processing power effectively. Even a single CPU running a memory-intensive workload, such as a scientific computing application, can be constrained by memory bandwidth.This problem is amplified on systems, where many processors must compete for bandwidth on the same system bus. Some high-end systems often try to solve this problem by building a high-speed data bus. However, such a solution is expensive and limited in scalability.

NUMA is an alternative approach that links several small, cost-effective nodes using a high-performance connection [**?**]. Each **node** contains processors[2] and memory, much like a small system. However, an advanced memory controller allows a node to use memory on all other nodes, creating a single system image as shown on Figure 1.4. When a processor accesses memory

[2]The group of processors in a node is usually called **socket**

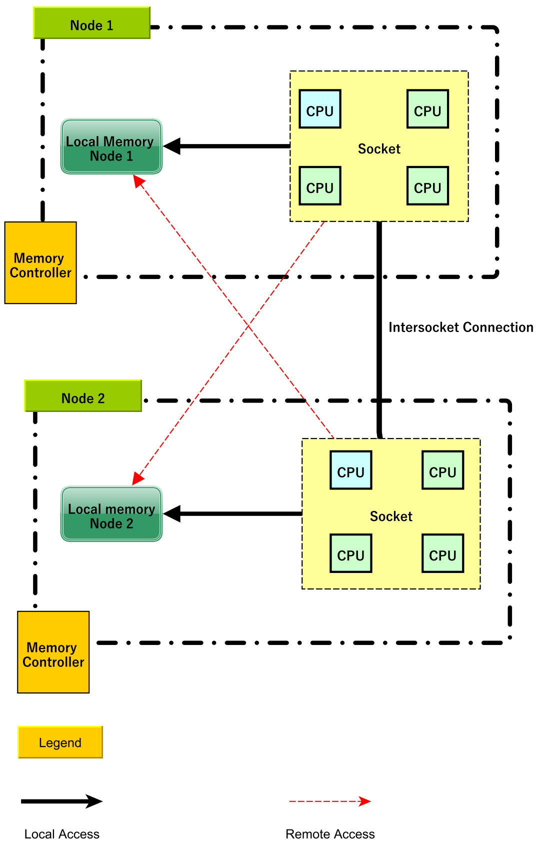

Figure 1.4: An example of a numa architecture

that does not lie within its own node (remote memory), the data must be transferred over the NUMA connection, which is slower than accessing local memory. Memory access times are not uniform and depend on the location of the memory and the node from which it is accessed, as the technology's name implies.

A lot of work was done to optimize existing algorithms to take into account NUMA [**? ? ?**], this work led to many changes in existing OS to take into account NUMA. NUMA is relevant to multiple processors and means that memory can be accessed *quicker if it is closer*. This means that memory is commonly **partitioned** at the hardware level in order to provide each processor in a multi-CPU system with its own memory. The idea is to avoid an argument when processors attempt to access the same memory. This is a good thing and means that NUMA has the potential to be more scalable than a **UMA** (multiple sockets share the same bus) design, particularly when it comes to environments with a large number of logical cores.

Now that we have a good idea of what is concretely server virtualization, and the predominant architecture in s, let us present how our server virtualization system handle I/O tasks for the virtual machines to helps us better understand the role played by the privileged domain (here the dom0) in these crucial tasks.

1.3 I/O mechanism in Xen

I/O tasks are handled by physical devices on a native OS. There are two major types of I/O activities, **block and network I/O** which use the block devices and network card by the means of s. How the virtualization system Xen manages virtual devices will be the point of interest in this section.

Virtual devices under Xen are provided by a **split device driver model**. The illusion of the virtual device is provided by two co-operating drivers: the **front-end**, which runs in the unprivileged domain and the **back-end**, which runs in the privileged domain (here the domain0) as shown on Figure 1.5. The front-end driver appears to the unprivileged guest as if it was a real device, for instance, a block or network device. It receives I/O requests from its kernel, as usual. However, since it does not have access to the physical hardware of the system it must then issue requests to the back-end.

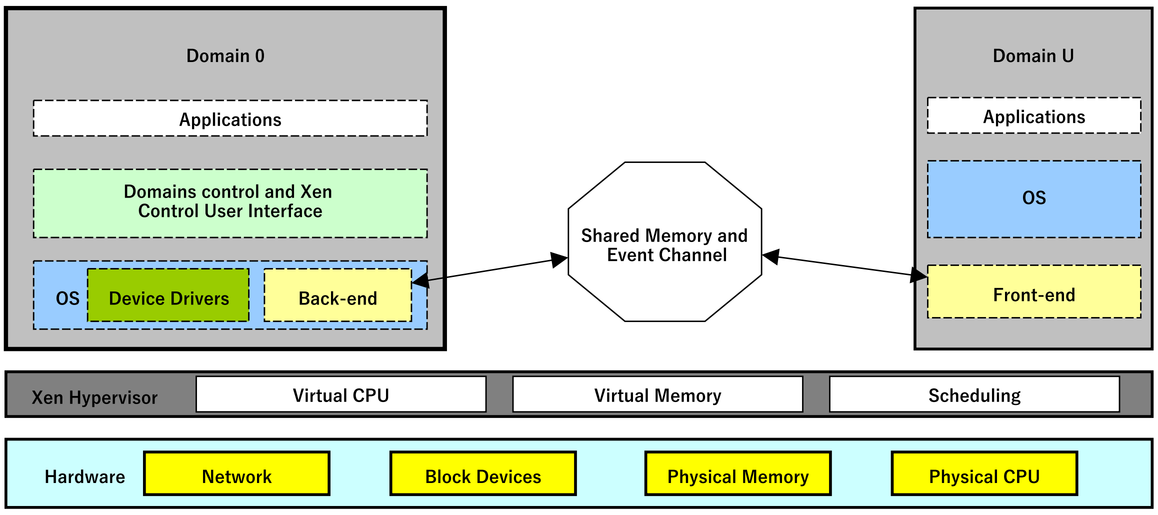

Figure 1.5: Split driver model

The backend driver is responsible for receiving these I/O requests, **verifying** that they are safe and then issuing them to the **real device** hardware. The back-end driver appears to its kernel as a normal user of in- I/O functionality. When the I/O completes the back-end notifies the front-end that the data is ready for use, the front-end is then able to report I/O completion to its own . Split drivers exchange requests and responses in **shared memory**, with an **event channel** for asynchronous notifications of activity [**?**]. When the front-end driver comes up, it uses the **Xenstore** to set up a shared memory frame and an interdomain event channel for communications with the backend. Once this connection is established, the two can communicate directly by placing requests/responses into shared memory and then sending notifications on the event channel. In the sections below, we will detail what happens in the case of block I/O and network I/O.

1.3.1 Network I/O mechanism

From the point of view of the back-end domain itself, the network back-end driver consists of a number of ethernet devices. Each of these has a logical direct connection to a virtual network device in another domain. This allows the back-end domain to route, bridge, firewall, etc the traffic to/from the other domains using normal operating system mechanisms [**?**]. Corresponding to each virtual interface in a guest domain, a back-end interface is created in the driver domain, which acts as the proxy for that virtual interface in the driver domain (dom0). Each virtual interface uses two **descriptor rings** (two shared memory rings), one for transmitting, the other for receiving. Each descriptor identifies a block of contiguous machine memory allocated to the domain. Each back-end interface has a number of **queues** to handle requests from a virtual interface.

1.3.1.1 Packet transmission

As shown on Figure 1.6, the transmit ring carries packets to transmit from the guest to the back-end domain. By doing so,

- it writes/copies the packet in the transmit ring (which is a shared memory between the driver domain and the virtual guest),
- then generates a virtual interrupt request (VIRQ) to notify the driver domain (dom0)
- and the return path of the transmit ring carries messages indicating that the contents have been physically transmitted and the backend no longer requires the associated memory pages.

Each transmit request is followed by a transmit response at some later date. This is part of the shared-memory communication protocol and allows the guest to (potentially) retrieve internal structures related to the request.

1.3.2 Packet reception

To receive packets, the guest places descriptors of unused pages on the receive ring. The back-end will return received packets by exchanging these pages in the domain's memory with new pages containing the received data and passing back descriptors regarding the new packets on the ring.

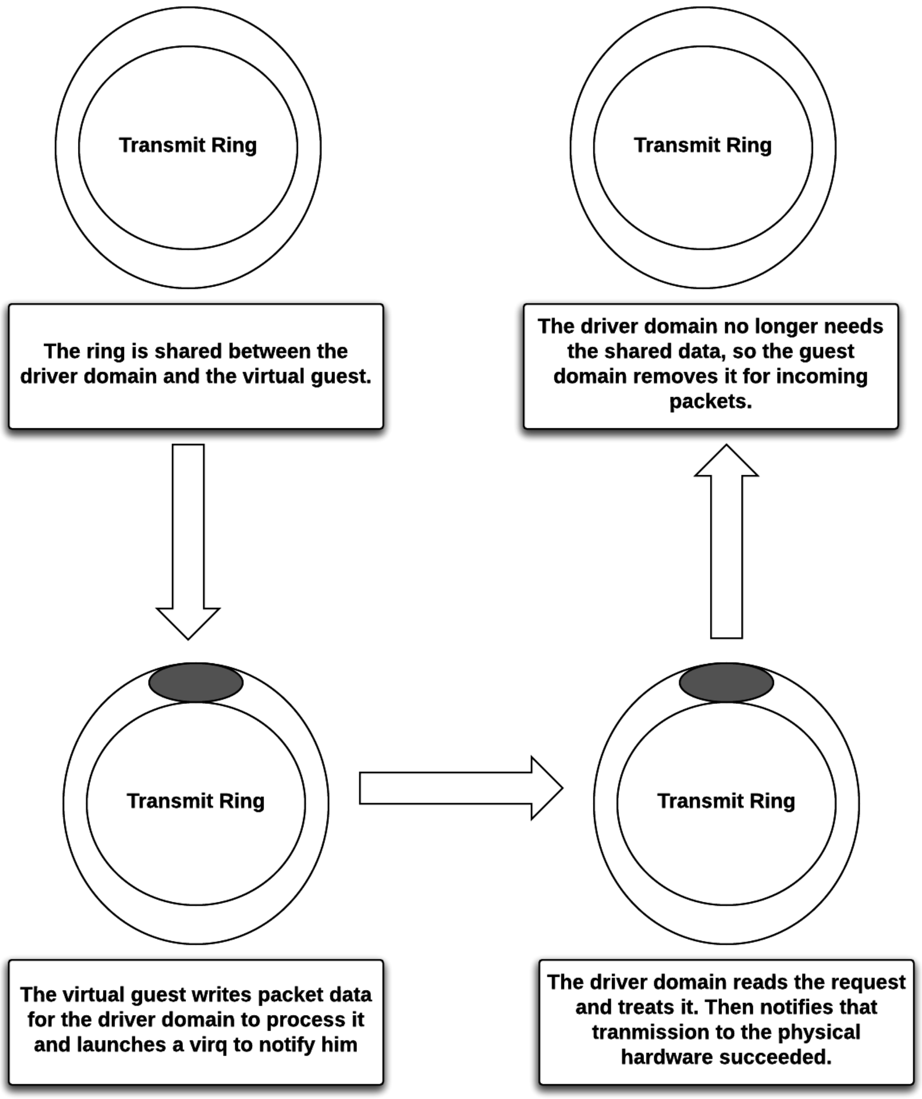
Transmit Ring
The ring is shared between the driver domain and the virtual guest.
Transmit Ring
The virtual guest writes packet data for the driver domain to process it and launches a virq to notify him
Transmit Ring
The driver domain reads the request and treats it. Then notifies that tranmission to the physical hardware succeeded.
Transmit Ring
The driver domain no longer needs the shared data, so the guest domain removes it for incoming packets.

Figure 1.6: Network I/O transmit mechanism between front-end and back-end

This zero-copy approach as shown on Figure 1.7, allows the back-end to maintain a pool of free pages to receive packets into, and then deliver them to appropriate domains after examining their headers. Receive requests must be queued by the front-end, accompanied by a donation of page-frames to the back-end. The back-end transfers page frames full of data back to the guest. Receive response descriptors are queued for each received frame.

Guests are expected to return memory to the hypervisor in order to use the network interface. They must do this or they will exceed their maximum memory reservation and will not be able to receive incoming frame transfers. When necessary, the back-end is able to replenish its pool of free network buffers by claiming some of this free memory from the hypervisor.

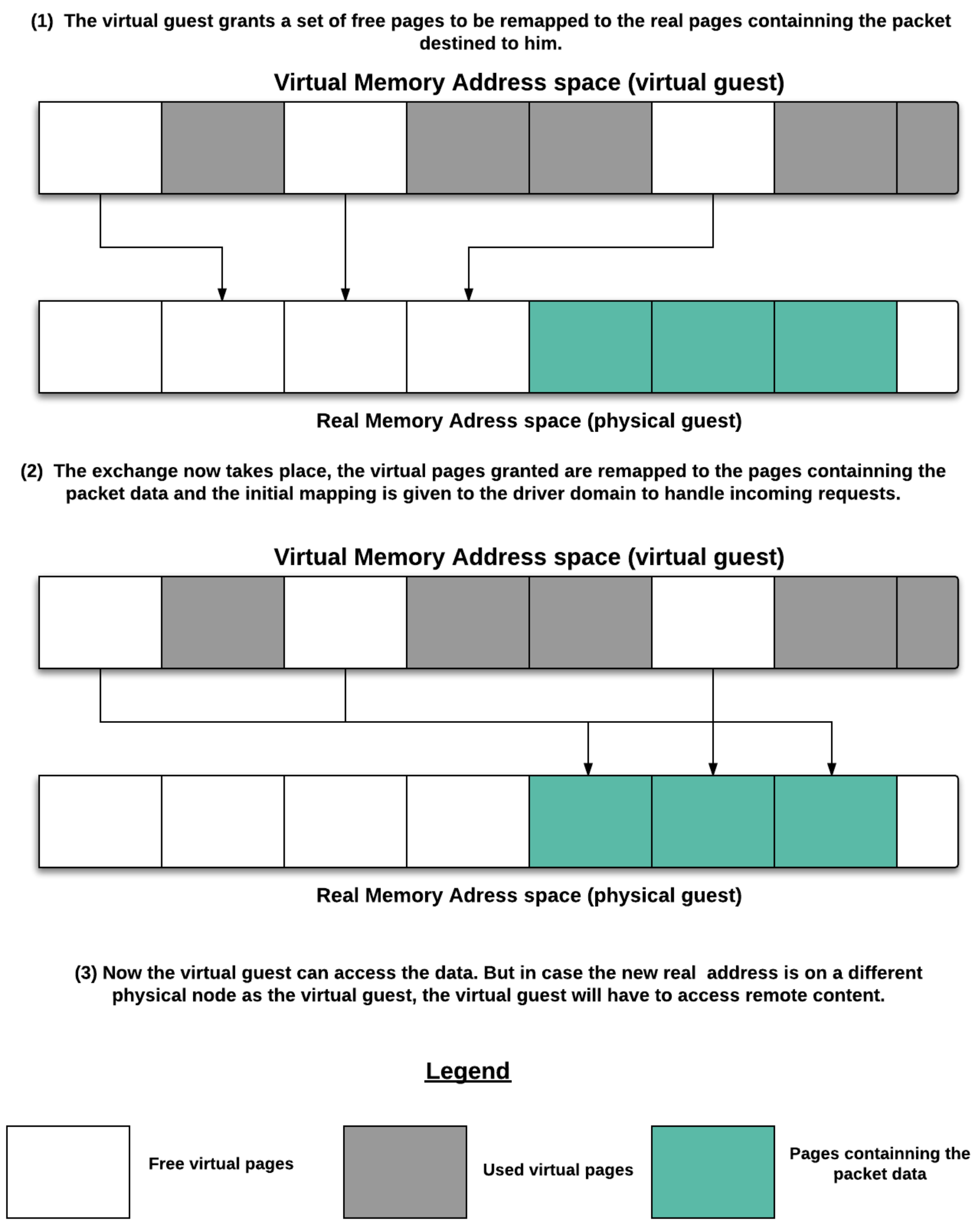
(1) The virtual guest grants a set of free pages to be remapped to the real pages containning the packet destined to him.
Virtual Memory Address space (virtual guest)
Real Memory Adress space (physical guest)
(2) The exchange now takes place, the virtual pages granted are remapped to the pages containning the packet data and the initial mapping is given to the driver domain to handle incoming requests.
Virtual Memory Address space (virtual guest)
Real Memory Adress space (physical guest)
(3) Now the virtual guest can access the data. But in case the new real address is on a different physical node as the virtual guest, the virtual guest will have to access remote content.
Legend
Free virtual pages
Used virtual pages
Pages containning the packet data

Figure 1.7: Zero copy mechanism

1.3.3 Block I/O mechanism

All guest OS disk access goes through the **virtual block device** (VBD) interface. This interface allows domains access to portions of block storage devices visible to the block back-end device. The VBD interface is a split driver, similar to the network interface described above. A single shared memory ring is used between the front-end and back-end drivers for each virtual device, across which I/O requests and responses are sent. Any block device accessible to the back-end domain can be exported as a VBD. Each VBD is mapped to a device node in the guest, specified in the guest's startup configuration.

The per-(virtual)-device ring between the guest and the block back-end supports two messages:

- **READ**: Read data from the specified block device. The front end identifies the device and location to read from and attaches pages for the data to be copied to (typically via from the device). The back-end acknowledges completed read requests as they finish.
- **WRITE**: Write data to the specified block device. This functions essentially as READ, except that the data moves to the device instead of from it.

In the next chapter, we will discuss about the questions raised in the problematic and its assessment.

Conclusion

Review

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Prospects

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Annex

Internship host structure description

IRIT, the Toulouse Research Institute of Computer Science, one of the largest joint research units at the national level, is one of the pillars of research in the **Midi-Pyrénées** with its 700 permanent and non-permanent members. Due to its scientific impact and its interactions with other fields, the laboratory is one of the structuring forces of the landscape and its applications in the digital world, so regional and national level. The unit is structured into 7 research themes grouping 21 laboratory teams. Its main research axes are:

1. Analysis and synthesis of information.
2. Indexing and retrieving information.
3. Interaction, Cooperation, Adaptation through Experimentation.
4. Reasoning and decision.
5. Modeling, algorithms and high-performance computing.
6. Architecture, systems, and networks.
7. Software development security.

Beyond that, IRIT's influence is reflected in various actions at european and international level, for example, the european laboratory IREP, the french spanish Laboratory for advanced studies in information and perennial cooperation with various countries including the Maghreb, Japan, Armenia, the Unites States of America, etc.

IRIT present in all Toulouse universities as well as the Institute of Technology of Tarbes Castres provides both coverage of the local territory and all the themes of computer science and its interactions, ranging from infrastructure enabling it to contribute to the structuring of regional research. The laboratory works to establish a continuum from research to valorization, despite the difficulties and limitations inherent in existing devices (industrial property, patent), by imagining and developing innovative forms of collaboration around the concepts of a joint laboratory or a consortium.