

Live Migration of User Environments Across Wide Area Networks

Cristian Zamfir

Submitted in fulfillment of the requirements for the Degree of Master by Research.

Department of Computing Science
Faculty of Information and Mathematical Sciences
University of Glasgow

July 2009

Abstract

A complex challenge in mobile computing is to allow the user to migrate her highly customised environment while moving to a different location and to continue work without interruption. I motivate why this is a highly desirable capability and conduct a survey of the current approaches towards this goal and explain their limitations. I then propose a new architecture to support user mobility by live migration of a user's operating system instance over the network. Previous work includes the *Collective* and *Internet Suspend/Resume* projects that have addressed migration of a user's environment by suspending the running state and resuming it at a later time. In contrast to previous work, this work addresses *live migration* of a user's operating system instance across wide area links. Live migration is done by performing most of the migration while the operating system is still running, achieving very little downtime and preserving all network connectivity.

I developed an initial proof of concept of this solution. It relies on migrating whole operating systems using the *Xen* virtual machine and provides a way to perform live migration of persistent storage as well as the network connections across subnets. These challenges have not been addressed previously in this scenario. In a virtual machine environment, persistent storage is provided by virtual block devices. The architecture supports decentralized virtual block device replication across wide area network links, as well as migrating network connection across subnetworks using the *Host Identity Protocol*. The proposed architecture is compared against existing solutions and an initial performance evaluation of the prototype implementation is presented, showing that such a solution is a promising step towards true seamless mobility of fully fledged computing environments.

Acknowledgments

I would like to acknowledge the help and advice I received from the following people.

My thesis supervisors Dr Peter Dickman and Dr Colin Perkins for the invaluable guidance, advice, and encouragement I received throughout the project and for the help in improving this thesis.

Professor Joe Sventek for the feedback, help and support I received during this project.

The Xen and DRBD communities for answering my questions and for the feedback I received, especially Keir Fraser for the quick fixes to network device migration in Xen, Lars Ellenberg for the help provided in understanding the technical aspects of block device migration in DRBD and Octavian Purdila for the help regarding *udev*.

The technical staff, especially Douglas MacFarlane and Naveed Khan for the help in setting up the network testbed.

Students Ross McIlroy, Stephen Strowes, Mark Shannon, Athanasios Pavlopoulos and Dora Galvez-Cruz for the early feedback during the initial steps of my work.

Students Silviu Andrica and Paul Marinescu for proof-reading the thesis.

My parents for their continuous support, without which this work wouldn't have been possible.

Alexandra Covaci for the support and encouragement during the writing up of the thesis.

Contents

1	Introduction	10
1.1	Scenario	11
1.2	Discussion and Scope	13
1.3	Limitations	14
1.4	Thesis Statement	16
1.5	Structure of the Thesis	17
2	Background	18
2.1	Virtual Machines	19
2.1.1	The Xen Hypervisor	21
2.2	I/O Virtualization	23
2.2.1	The Xen I/O Subsystem	23
2.2.2	Soft Devices	23
2.3	Mobility Protocols	25

2.3.1	Survival of TCP Connections for Long Periods of Disconnection .	27
2.4	Storage Replication	29
2.4.1	Mirroring Virtual Disks	33
2.5	Migration of Virtual Machines and Environments	36
2.5.1	Virtual Machine Live Migration	36
2.5.2	Migration of Virtual Environments	37
2.5.3	Persistent Storage Management	44
2.5.4	Xen Security	45
2.6	Quantifying Interactive User Experience	47
2.7	Live Migration of User Environments	48
2.8	Conclusions	49

3 Architecture 50

3.1	Discussion of the Factors that Influence the Total Live Migration Time . .	51
3.2	General Architecture	52
3.3	Specific Architecture	54
3.3.1	Virtual Block Device Live Migration	55
3.3.2	Reducing the Impact of the Migration on Network Traffic	57
3.3.3	Migrating Network Connections across Subnets	58
3.3.4	Discussion	60

4	Implementation	62
4.1	Persistent Storage Migration	63
4.2	Live Migration of Virtual Block Devices	64
4.3	Migration of Network Connections	67
4.4	Testing	69
4.5	Discussion	72
5	Performance Evaluation	73
5.1	Experimental Setup	73
5.2	Network Performance	74
5.3	Disk I/O Performance	81
5.4	Wrap-up	85
6	Future Work	86
6.1	User Activity Traces	87
6.2	Asynchronous Replication	87
6.3	Estimating the Impact of Migration on User Experience	87
6.4	Live Migration across Hypervisors for Mobile Devices	88
7	Summary and Conclusions	90

List of Figures

1.1	Scenario: Professor <i>X</i> 's virtual environment is live migrated between various physical hosts around the world such as her office PC or her in-car PC. Her system is not suspended during the commutes so that network activity such as attending a video conference is not affected.	13
2.1	Architecture of the Xen Virtual Machine.	21
2.2	Host Identity Layer	26
3.1	Architecture for live migration of user environments	53
4.1	Currently, live migration of virtual block devices can be done using SAN.	63
4.2	Live migration of virtual block devices.	65
4.3	Migration of network connections using HIP.	68
5.1	Performance Evaluation Setup	74
5.2	Live Migration across 100 Mbps Ethernet of a VM running an Iperf client using the unmodified Xen live migration (no storage migration was performed). The storage was mounted from a remote server using the ATA over Ethernet protocol. Migration occurs in the same subnetwork. No additional network delays were introduced.	76

5.3	Live Migration across 100 Mbps Ethernet of a VM running Iperf in client mode. The experiment is done using the VBD migration prototype. No additional network delays were introduced.	77
5.4	TCP throughput measured on the server during migration across subnetworks using HIP. VBD migration is also performed. No additional network delays were introduced.	79
5.5	TCP throughput measured on the client during migration across subnetworks using HIP. VBD migration is also performed. No additional network delays were introduced.	80
5.6	I/O performance of sequential input during virtual block device migration over a 100Mbps link measured with the Bonnie++ benchmark.	82
5.7	I/O performance of sequential output during virtual block device migration over a 100Mbps link measured with the Bonnie++ benchmark.	83

Chapter 1

Introduction

Coping with user mobility across pervasive hardware while maintaining access to a highly customized environment is a challenging task and today's computing environments do not provide adequate support for it. The operating system, installed software and user data take up several gigabytes of storage and are tightly coupled with the hardware. Common solutions for user mobility include mounting user profiles stored on distributed file systems, roaming profiles, remote connections to the user's machine, process migration, carrying a heavy and fragile laptop or using a mobile device with limited computing capabilities. All these solutions have drawbacks that prevent a user from resuming work without interruption across various computers. For instance, remote connections to a user's machine are prone to network disconnections and are sensitive to the network delay. Mobile devices such as PDAs typically have a small computing power and a limited interface. Laptops are fragile devices that can compromise user data or be stolen.

Ideally, a user would like to use the same settings and data as well as the same software packages everywhere. This would allow the user to use the software and settings that she is most accustomed to. Moreover, the same data should be available at every location where the user is using the computing environment (i.e. at home and at the work place). Unfortunately, there are many hindrances to accomplishing this ideal. Firstly, even though the user can take advantage of many machines, typically the user environment on each of them is significantly different because the installed software and the user settings are different. Moreover, the user has to take care of synchronizing the data that she modified on one machine so that it is available on the others. Network connections are

restarted once the user resumes working at a new location. For instance, for each Secure Shell Connection (*ssh* [11]) the user has to manually restart the connection and introduce a password. Another disadvantage of the current approaches to the migration of user environments is that during the commutes, the user environment is stopped and cannot perform any useful work, such as batch jobs.

This thesis proposes an architecture to solve these issues by enabling the live migration of user environments across wide area network links. Therefore, the first challenge is to allow the same user environment to be migrated between different PCs. The running state of a user environment is a snapshot of the current activity that includes the running programs and network connections. Seamless mobility implies that the running state of the user environment is preserved during user commutes. Virtual machine (VM) technology makes it easy to seal the state of an operating system and send it over the network. Resuming the virtual machine on the destination can be done easily and without compatibility problems. The virtual machine monitor offers a uniform hardware interface to the VM so that migrating on a platform with another type of hardware does not change anything from the point of view of the guest VM.

Migrating a user's whole environment packed as a virtual machine capsule has been recognized as a way to support seamless user mobility ([75], [77]). The capsule can be suspended and later resumed (non live relocation of the virtual machine) allowing the user to work with minimal interruption.

My approach is to use live migration for the same purpose. This approach does not require the user to suspend the capsule before she starts commuting. Instead, it allows the user capsule to be migrated with a very small and usually unnoticeable downtime. This is a new approach and has the advantage that network connectivity is not restarted, rather it is migrated with virtually unperceived downtime to the new destination. Also, batch jobs such as updating the operating system with the latest security patches or a lengthy download can be performed during the commutes.

1.1 Scenario

This section describes the scenarios that motivate the architecture proposed in Chapter 3. Similar to Internet Suspend/Resume ([77]), for the following scenario, it is assumed that

hardware is pervasively available, and a Virtual Machine Monitor is installed and configured to run the user's VM.

Doctor D is working on some patient records on her computer at home while downloading a tune she just heard on TV from the iTunes website. At the same time she is also updating her Operating System software. She leaves for the hospital without taking her laptop. In her car, she can resume her work on the car's computer while her chauffeur is driving her to work. The state of her computer was synchronized using the car's WiFi connection and all the network connections are still maintained. In the interim, the music file download finishes and she can listen to the new song in the car. Moreover, her OS is already up to date with all today's security patches. She configures her work application to download last night's updates to patient records and to process them with the specialized software installed in her computer.

In her office she is glad to see that her specialized software has almost finished downloading and processing the updates to the patient's records. She is now starting to visit the patients in various areas of the hospital. Using the computer provided in every patient room, where her computer environment is migrated, she can easily record her notes and enjoy the crisp interaction and familiar user interface.

Her computer is permanently monitoring the vital signs of a patient that has a high heart attack risk and signals are being sent from her monitoring device to the doctor's computer. The computer suddenly detects an anomaly in the readings and automatically alarms the doctor. The doctor quickly processes the output with the software on her computer and decides that an ambulance should be sent for the patient right away.

A similar scenario is shown in Figure 1.1. In the morning, Professor X is using her virtual environment at home where she can start downloading her email. She then starts commuting to work so she instructs her virtual environment to switch to the in-car computer. During her commute, she is able to watch the news and to read some of her email using her customized user environment. At the office, she is able to resume work on her presentation slides for her upcoming afternoon course at the university. She instructs her virtual environment to switch to the computer in the University Lecture Hall. During her course she is also able to demonstrate some software to the students, display the slides from her virtual environment and check her email during the break.

Throughout the day, the professor was able to access her personalized environment on

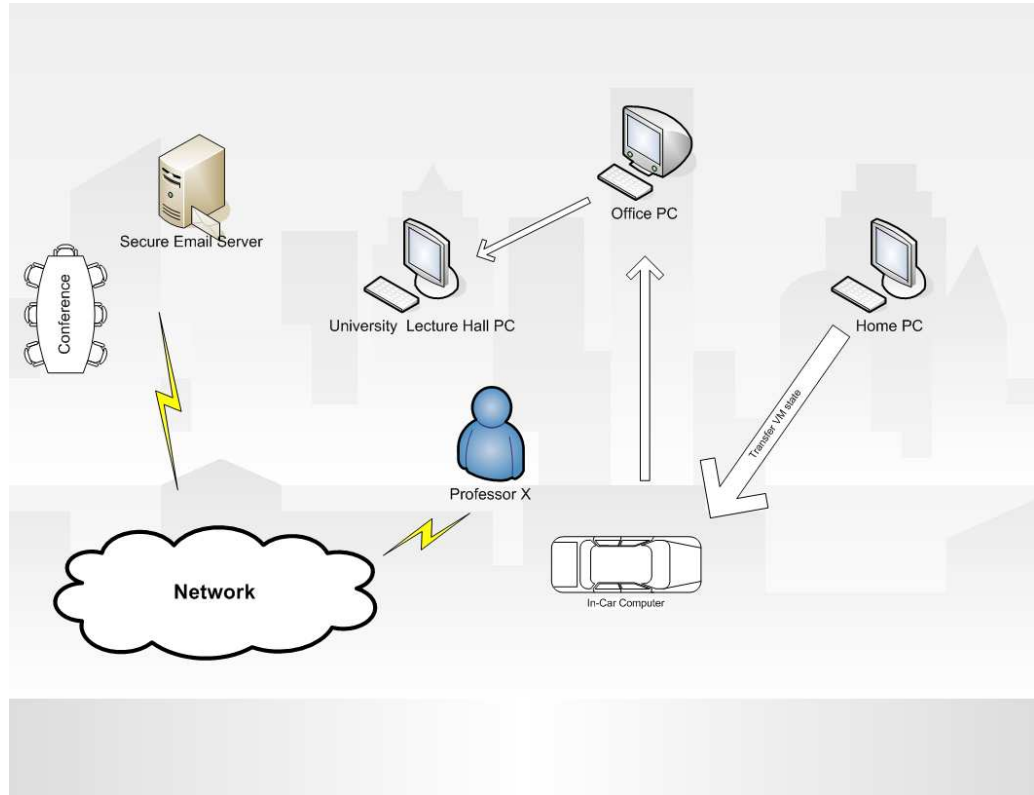


Figure 1.1: Scenario: Professor X's virtual environment is live migrated between various physical hosts around the world such as her office PC or her in-car PC. Her system is not suspended during the commutes so that network activity such as attending a video conference is not affected.

various pervasively available hardware. All her applications worked *out of the box* on the local hardware, without sacrificing performance and providing a good user experience. Moreover, she did not have to restart any network connections at any time so her virtual environment did not notice any downtime.

1.2 Discussion and Scope

This is clearly a futuristic scenario but it highlights some very interesting potential uses of live migration. The biggest problem is live-migrating the user environment's entire state, which may consist of several gigabytes, over the network. The limitations of this work are discussed in Section 1.3.

The typical workload intended for migration is normal home and office load. This work does not deal with server or data intensive workloads. Processor intensive applications on

the other hand are easier to migrate because they do not generate a lot of updates to the persistent state of the user environment. Real-time network applications such as an audio-conference application do not generate large updates to persistent storage but are impacted by the network handover, the new route to the destination, the additional traffic generated by migration as well as the network downtime during migration. Therefore, real-time traffic might be significantly impacted by the migration. The thesis does not explicitly provide solutions to handle the requirements of real time traffic. Instead it provides a best-effort approach to migrate these connections without differentiating between their QoS requirements and the requirements of other network connections.

Security and trust are important requirements of such a scenario. The user should be able to remotely authenticate the destination of a migration. A possible approach is to use the Trusted Platform Module (TPM) [14] to verify that the remote host has not been compromised by an attacker. This approach could leverage existing work such as Terra [38] that builds an architecture for trusted computing. However, these issues are outside the scope of this thesis.

1.3 Limitations

This thesis studies the mobility of whole user environments across the wide area network. Mobility implies migrating the whole operating system on pervasive hardware without interrupting or suspending user programs or disturbing network connectivity. From a usability point of view, the user is responsible for setting the destination where she wants her user environment to run next. This decision is taken before the user starts to commute.

However, there are some limitations:

- **Downtime:** Migration of a running user environment, from a hardware machine to another, certainly incurs a certain amount of downtime. It is up to the architecture and actual realization prototype to make sure this downtime is small enough to have an unnoticeable effect on user experience and network connections. An acceptable bound is discussed in Section 2.3.1.
- **Availability:** Sometimes it is not possible to predict the new commute location for a user due to unpredictable user behavior or unforeseen circumstances. If a certain

next destination is decided upon, but the user changes her mind in the interim and wants to resume work at a different destination, this will make the current state of her machine unavailable. This happens because live migration is used: once the live migration process finishes and an existing copy is ready to run at the destination, the copy running at the source is suspended and becomes unavailable.

Because the user environment is running at all times, suspending a remote copy of it is a complex issue. In the following scenario, user A is commuting from work to home. During her commute, her running computing environment is gradually transferred to her home hardware. Once the migration succeeds, the home location will be running her computing environment and the one at her work location will be suspended and will be no longer active. If an urgent call from work makes the user change her mind about going home and return to her work place, she will find that her computing environment is running in a remote location. The user will need to either resume from the state present at the work location or to trigger a reverse migration. Such scenarios will impact the degree of mobility.

- **Resources:**

Because the architecture is based on the concept of live migration, the user environment starts executing at the destination and consumes available resources there. Moreover, the user has to authenticate twice in order to use a destination. First she has to authenticate to the remote host before starting the migration. This step also checks for available resources. A second authentication phase takes place when the user logs in at the destination.

Depending on the storage and memory size of the user environment, the workload performed and the network capacity available, it may not always be possible to provide the required degree of mobility. For some static resources, such as memory size, it can be determined before migration starts if a destination is not suitable. In this case, migration will fail and the user will be notified immediately. However, for dynamic aspects such as workload and network capacity that can change during migration, it is not always possible to tell beforehand if migration is possible. In this case, migration is attempted and if it is unsuccessful, the user will not be able to resume work at the destination from the most up to date copy of her environment. In this situation there are two options. The default behavior in case of a failed migration is to not allow the user to start at the destination. Therefore, the user will have to use the copy of her environment that runs on the source because that is the most up to date. The source environment can still be used via a remote connection to the source host. However, there is the alternative that the user

resumes execution from a previous version of her environment, if such a version already exists at the destination (for instance from a previous migration). The disadvantage is that her copy is not up to date. Resuming from a previous snapshot is entirely the user's choice. The copy running at the source will have to be suspended in this case. Given the data integrity issues raised by the second approach, this thesis only considers the default behavior.

- **Hardware Compatibility:** Ideally, a solution for mobility that runs on pervasive hardware should take into account the differences between various hardware capabilities and features of different platforms. There is an obvious tradeoff between the degree of similarity of hardware platforms and the subset of features and degree of customization offered to the user. The more similar the hardware platforms are assumed to be, the larger the subset of features and the number of peripherals available to the user will be. On the other hand, being able to accommodate all mobile and desktop platforms will limit the functionality the user will benefit from. For instance, stripping down some of the functionality of the computing environment also implies that the user will make marginal use of the computing power of desktop PCs.

Apart from these limitations, the approach described in this thesis is useful in providing a high degree of mobility to users. Users can perform the live migration of their environment across various machines, assuming the hardware is pervasively available and compatible, and enough resources are available to facilitate the migration of their environment.

1.4 Thesis Statement

This thesis discusses the limitations behind building a system that allows a user to use her unmodified environment at the various locations where she travels without having to suspend her environment or disconnect the network connections. I will propose an architecture for live migration of user environments across wide area networks, designed to have reasonable downtime, high availability and a low impact on user experience. The architecture fits in with the usability scenarios described in Section 1.1.

I assert that live migration of user environments across wide area networks offers a highly

desirable degree of user mobility and can effectively support user mobility in pervasive computing environments. I demonstrate this by building an architecture for live migration of user environments, building a proof of concept implementation of this architecture and evaluating its performance.

This thesis proves that live migration of user environments, including persistent storage and network connections is possible by doing a real system implementation. The proof of concept implementation shows that this new approach to user mobility does not impact performance before and after migration, that it has reasonable performance during the migration and that indeed, users can resume work and network activity without restarting their environment or restarting any applications.

1.5 Structure of the Thesis

In Chapter 2 relevant background work is presented. This gives an overview of the basic blocks involved in the proposed architecture such as virtual machine technology, mobility protocols and storage replication. In the same chapter, in Section 2.5 similar approaches to migration of whole user environments are overviewed. Chapter 3 describes the architecture of the solution. First, a general architecture is described as well as a specific one based on system virtual machine technology. The architecture describes solutions to the live migration of persistent storage and network connections. Chapter 4 shows the implementation details of an initial proof of concept. It discusses the implementation challenges involved in minimizing the network downtime and replicating block devices as well as the solutions proposed for solving each of these challenges.

The performance of the initial prototype is evaluated in Chapter 5. The evaluation is done using existing benchmarks as well as a benchmark created specifically for the purpose of simulating various user workloads. Future work is detailed in Chapter 6. This chapter describes what is needed to improve the prototype, how to make a more realistic evaluation as well as how to better handle failure scenarios. The summary and final conclusions of this thesis are outlined in Chapter 7.

Chapter 2

Background

The main issues involved in the live migration of user environments are how to create a portable user environment and how to perform the migration of the necessary resources, including active network connections. Thus, a broad range of related work about virtual machines, storage replication and mobility solutions will be overviewed in this chapter. Section 2.1 gives an overview of virtual machine technology since this is an essential building block of the architecture for live migration. Section 2.1.1 further discusses the Xen hypervisor used in the implementation in this thesis and section 2.2 discusses some related aspects of I/O virtualization. Section 2.3 gives an overview of the mobility protocols such as Mobile IPv6 and Host Identity Protocol (*HIP*). Mobility protocols are used to allow transparent migration of network connections across subnetworks and are another important building block of the architecture for live migration of user environments across wide area networks. Section 2.4 overviews related work about replication of persistent storage. This topic has been mostly researched in the context of storage replication for server storage backup and fast disaster recovery. In this thesis, the same principles are applied and studied in the context of migration of the user's persistent storage across wide area networks. Sections 2.5 and 2.6 discuss similar work in the area of user mobility. Finally, Section 2.7 discusses the motivation for proposing live migration of user environments and Section 2.8 concludes.

2.1 Virtual Machines

A virtual machine is an environment created within another environment and executes software in the same way as the machine for which the software was developed. Virtual machines are divided into two categories: *process virtual machines* and *system virtual machines*, depending on the type of software they are executing. Process virtual machines execute individual processes while system virtual machines execute whole operating systems. In most cases, the virtualization layer must emulate several resources that are not present in the hardware, and hence the software executed by a virtual machine has lower performance than the same software running on the real hardware.

For process virtual machines, the virtualization software, usually called the runtime, is placed on top of the operating system and is capable of emulating userspace instructions as well as operating system calls. Process virtual machines are capable of running processes compiled for a different instruction set by performing emulation or binary translation.

High-Level Language Virtual Machines such as the Java Virtual Machine (JVM) [55] are a type of process virtual machine that offer cross-platform portability by implementing a VM capable of executing a virtual Instruction Set Architecture (ISA) on various different platforms. The high level code is compiled to a portable code format that contains instructions for the virtual ISA. The portable code format can be run without the need for recompilation on all hardware that is capable of running the virtual machine. The advantage of a high level language VM is portability at the expense of lower performance. VMs such as the JVM encode instructions as *bytecodes*, assume an unbounded memory size and use garbage collection.

System virtual machines provide the abstraction of a complete operating system. A virtual machine environment consists of an additional software level called the Virtual Machine Manager (VMM) or *hypervisor* that manages the hardware resources and provides virtual resources such as processor, storage and peripherals to the *guest* systems. Applications running in virtual machines are completely unaware of the fact that they are not running in fact on the bare hardware. The advantages of system VMs are that multiple OS environments can run at the same time on the same computer, in strong isolation from each other. Moreover, the virtual machine can provide an instruction set architecture (ISA) that is different from that of the real machine.

Systems such as VMware [16], KVM [4], QEMU [9] and Virtual PC [7] virtualize commodity PC hardware allowing a machine to host multiple guest operating systems and implement a full virtualization of the underlying hardware. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc.

There is an overhead associated with full virtualization. VMware dynamically rewrites portions of the hosted machine code to insert traps wherever VMM intervention might be required. This translation is applied to the entire guest OS kernel. The VMM must also create and maintain data structures such as shadow page tables. These data structures must be updated for every access by the VMs.

The Xen hypervisor on the other hand uses paravirtualization which means that it presents a modified interface to the guest operating system. Guests have to be ported to be able to run on Xen. This will be further discussed in Section 2.1.1.

Another approach to paravirtualization is Denali [86]. Denali is designed to support thousands of machines running network services. The Denali implementation does not support the full x86 ISA and thus does not target existing ABIs. It does not address the problem of supporting application multiplexing, or multiple address spaces, within a single guest operating system.

Another approach to virtualization is one in which the operating system kernel allows multiple isolated guest capsules (usually called jails) to run at the same time. This approach is usually used in virtual hosting and has very low overhead. However, the disadvantage is that a number of resources have to be shared by the guest and the host. Moreover, it cannot allow guest operating systems to have a different kernel than the one of the host operating system. Examples of this approach are Linux-Vserver [5], OpenVZ [8] and FreeBSD jail[48].

From the point of view of migrating user environments, almost all the systems described above provide a good degree of isolation and run at almost native performance on the underlying hardware. In this thesis, the implementation is based on Xen which offers excellent performance and is open source and thus was a good option to extend in order to perform live migration. The choice was further motivated by the fact that some of the required live migration functionality was already part of Xen [28].

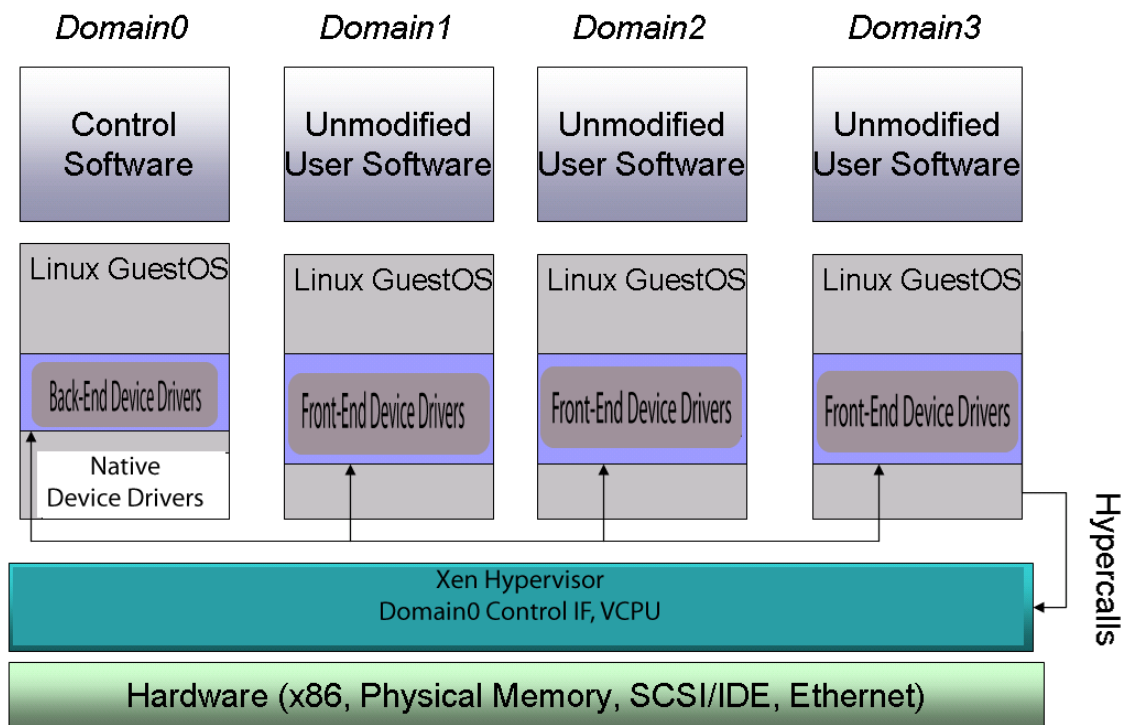


Figure 2.1: Architecture of the Xen Virtual Machine.

2.1.1 The Xen Hypervisor

Xen is an open source system level virtual machine monitor. It allows a virtual machine to run multiple independent OSs, each in its own isolated environment. The created sand-box environment gives the guest virtual machines the illusion that they are running on real hardware. As opposed to VMware that does full virtualization, Xen uses *paravirtualization* which minimizes the overhead of virtualization. However, it does that at a cost. Guest OSs have to be ported to be able to run on Xen, that is their kernel has to be modified to prevent them from running in ring 0 of an x86 architecture. It also exposes a set of highly efficient device driver abstractions that decrease the overhead of virtualization. Applications however can still run unmodified because the Application Binary Interface is not changed. More recent versions of Xen also support full virtualization using the hardware assisted virtualization provided by Intel VT-x and AMD-V extensions. These extensions enable unmodified guest OSs to run in Xen with a low overhead.

In Xen parlance, each VM is called a domain (Figure 2.1). Dom0 is the domain started during boot and it proxies hardware requests from the other domains which are called

domUs. User level management software running inside Dom0 is allowed to use the control interface to the other domains. Dom0 is responsible for controlling scheduling parameters, creating, migrating and destroying the other domains as well as creation and deletion of virtual network interfaces and virtual block devices.

The domains communicate with Xen through hypercalls, which are synchronous calls similar to system calls and allow the guest OS to perform privileged operations. Communication from Xen to the domains is done using asynchronous event channels which deliver notifications to domains much like the hardware interrupt mechanism.

Data transfer between domains and Xen is done using asynchronous producer-consumer I/O rings aimed at reducing the transfer overhead. These data structures are general enough to accommodate different types of devices; they allow reordering of requests and decoupling notifications from the production of requests or responses. Therefore, both virtual interface cards and virtual block devices can use them efficiently.

Only Domain0 has direct access to disks while the other unprivileged domains use a virtual block device (VBD). Xen has more knowledge of the disk structure than domUs, and so a scheduling algorithm within Xen may reorder the disk requests. This may lead to responses being delivered out of order to the domUs. Dom0 manages a translation table for each VBD. Xen then checks permissions and produces a zero-copy transfer using DMA between the corresponding sector of the physical storage device and the memory shared between Xen and the unprivileged domain.

Xen was designed with the goal of running up to 100 virtual machine instances on the same machine. The overheads are considerably lower (less than 5%) than in the case of virtual machines that perform full virtualization.

The architecture presented in this thesis is based on Xen owing to the good performance it achieves and because it is open source and was straightforward to extend. However, it should be possible to use other virtualization technology such as the full virtualization approach of VMware, provided access to the hypervisor source is available. There is nothing conceptually specific to Xen in the architecture, and other virtualization technologies could also be used.

2.2 I/O Virtualization

The specific architecture presented in this thesis is based on using virtual machines to encapsulate the user environment. In a VM, the persistent storage and network devices are usually virtual resources backed by real devices. Since migrating the persistent storage and network connections of a user environment are the main focus of this thesis, this section will give an overview of I/O virtualization.

The approach for virtualizing a specific I/O device is to construct a virtual version of the device and to virtualize the I/O activity between the physical device and the virtualized one. Most of the time, the virtualized device is not necessarily similar to the physical one. For instance, the virtual version of a block device is usually a file residing in userspace but can also be a dedicated physical disk drive.

2.2.1 The Xen I/O Subsystem

The Xen I/O subsystem is based on split device drivers. A virtual machine is granted access to the physical hardware device. It makes use of the unmodified operating system driver for that device. It shares the device with other guest virtual machines by granting access to it through a *back-end* device driver. The other guest VMs will be using a *front-end* device driver to gain access to the physical device (Figure 2.1). Interactions between the back-end and the front-end are carried out via a shared memory primitive called a device channel. The migration of an I/O device means, roughly speaking, simply reconnecting the front-end and the back-end upon migration, therefore, this decoupling makes it easier to perform live migration and the implementation in this thesis heavily relies on this decoupling to migrate persistent storage.

2.2.2 Soft Devices

Adding software functionality to the driver level can benefit system performance and functionality in many ways. Since the architecture provided by this thesis can be regarded as adding live migration functionality at device driver level, it is important to describe the related concept of *soft devices* [84]. Soft devices are device drivers that can

be partially implemented as userspace applications. Userspace applications are easier to write and can reuse a plethora of libraries, compared to kernel code. One possibility to implement live migration is to use soft devices. This was not used in the specific implementation from this thesis but could be the subject of future work. Therefore, this section describes soft devices in more detail.

At the block level, software devices could extend the set of features offered by traditional devices by adding functionality such as data compression or encryption. This normally implies creating an OS specific device driver but with the aid of a hypervisor this can be implemented in an OS agnostic way. For instance, Xen implements the *blktap* to facilitate the development of soft devices for block devices. It allows soft devices to be constructed as userspace applications within the privileged domain and provide the same regular interface to any of the guest operating systems.

Soft devices are specific to Xen. The architecture of block soft devices consists of a kernel message switch between 4 device channels. It interposes between the back-end of the device VM and the front-end of the guest VM. Therefore it can access all the block requests passing from the guest virtual machine towards the driver VM. The message switch can operate in several configurations. The *MODE_PASSTHROUGH* bypasses userspace device channels in order to achieve the lowest overhead and to allow processing within the kernel. The *MODE_INTERPOSE* routes requests through userspace, allowing all requests to be processed by userspace libraries. The *MODE_INTERCEPT_FE* routes all requests from the guest VM front-end to userspace. This means that userspace applications can process the requests and then commit them to any OS specific backing storage such as a distributed file system, a partition, etc.

The advantage of implementing soft devices in userspace is that many already existing userspace tools can be used without any modification. Moreover, this can be achieved without modifying existing Xen back-end and front-end interfaces. Even though the soft device application is implemented in userspace and incurs additional overhead, the block soft device sustains a high level of performance.

Live migration of persistent storage could be implemented as a soft device extension and is a viable alternative to the implementation used in this thesis. Moreover, it could bring the benefit of easily adding functionality such as disk encryption which would be a very desirable feature to mobile users. The implementation in Chapter 4 is a kernel based implementation and does not use soft devices, therefore even though it does not incur the

overhead of userspace processing, it may be harder to extend with additional features. This work chose a kernel implementation mainly for the low overhead.

2.3 Mobility Protocols

Historically, hosts in the Internet were static and remained at fixed locations. Host mobility introduces a new challenge to Internet protocols because hosts can move from one network to another network. This is especially true for small mobile devices with Internet connectivity but also for whole system user environments, as stated in Section 1.1. IPv4 was not designed with mobility in mind and the mobility scheme was developed later in Mobile IPv4 [64]. However, Mobile IPv6 [44] was designed in parallel with the IPv6 [31] protocol. Network connections mobility is an essential feature that needs to be present in a realization of the architecture to support the scenarios described in Section 1.1. I will further discuss how two mobility protocols achieve this goal.

In Mobile IPv6 and Mobile IPV4 nodes have a home address that does not change while the node moves to a different network. A multi-homed node obtains a *care-of-address* from the foreign network and packets are tunneled through its home network to the care-of-address in the foreign network. The tunneling is transparent to upper layer protocols. Thus, Mobile IPv6 allows a node to migrate to another network and still keep network connections active because they are tunneled through the node's home network. The registration of the care-of-address is done either with Registration Request/Reply messages for IPv4 or with Binding Update/Acknowledgment for IPv6 messages.

However, tunneling introduces backward dependencies to the home network, which is not a reliable solution for live migration of user environments. A route optimization can be performed between two nodes that support Mobile IPv6 so that the peer node can be informed of the new care-of-address and use the care-of-address instead of the home address of the mobile node. This means that if the route optimization can be applied there is no need to tunnel the traffic through the node's home network. Tunneling still has to be used for in-flight packets until the route optimization is performed.

Within the current Internet architecture, the IP address is both an identifier for the host as well as the topological locator. This is a problem for the migration of network connections because after migration the node will obtain a new IP address which acts as a

new identifier as well. The second solution to network mobility discussed in this section, the Host Identity Protocol (HIP) [58] solves this problem. It decouples the locator and identifier by introducing the Host Identity namespace. Thus, the IP address is only used for routing, and multi-homed hosts are identified in a secure way using a new cryptographic name. The HIP protocol acts as a layer between the Network Layer and the Transport Layer (Figure 2.2).

HIP hosts can have multiple identities which are in fact the public keys from public-private key pairs. The private key is used to prove that the host owns the respective identity to other peers. This adds a level of security to HIP. It is implemented in the following way: the Host Identity can be represented as a 128 bit Host Identity Tag (HIT) by hashing the public key. The 128 bit representation is the same length as an IPv6 address and can be used by applications as an IP address. Applications only use the HIT as the destination host.

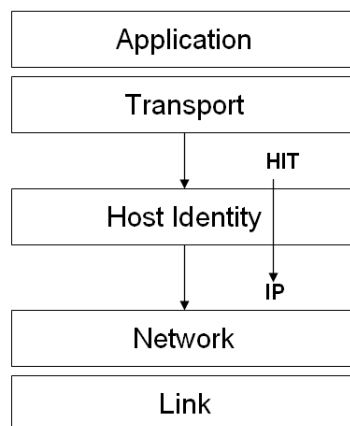


Figure 2.2: Host Identity Layer

To make the migration faster, HIP also defines a mobility management protocol allowing the mobile host to create a re-address packet that contains the new IP address of the node in the foreign network and allow it to resume connections with peer nodes. The vertical handover delays of HIP and MIPv6 are compared in [45] where HIP performs slightly better than MIPv6. The delays depend on the amount of signaling and the network round trip times but certainly even for wireless networks, handover delays between 2-5 seconds are possible. However, to be more useful and more widely deployable, HIP needs some support from the existing infrastructure, including the Domain Name System (DNS).

Because of the better network handover performance of HIP compared to MIPv6, this

this thesis uses HIP for implementing live migration network connections.

The migration of network connections also has to deal with network address translation (NAT [18]). NATs are widely used to allow hosts to connect to the Internet even though they have a private IP address. It does that by asking the gateway server to modify the network address information inside the packet header while the packet is in transit, essentially remapping a given address into another. The implementation in this thesis provides support for live migration of network connections behind NATs and firewalls by using existing techniques provided by HIP [20].

IPsec security associations are bound to the HITs and are not modified when the IP address changes. This way, a mobile host can use a single transport layer connection associated to one HIT. The main issue with traversal of the NAT and firewall middle-boxes is to make the HIP flow identifier available to the middle-boxes. HIP places a flow identifier inside the signaling protocol payloads. The flow identifier is a triplet containing the destination IP address, the Security Parameter Index (SPI) used to encrypt the traffic [19] and the protocol. When the underlying IP address changes, the flow identifier has to be updated for all the middle-boxes. NATs and firewalls are supposed to intercept these messages in order to learn the flow identifier and to forward the packets accordingly. Firewalls introduce the problem of routing asymmetry because packets could be forwarded on one path and return on another path. This is not true for NATs and therefore the problem is much simpler. The challenges involved in creating HIP-aware NATs and firewalls are discussed in detail in [20]. This thesis uses an implementation of NAT traversals that works with legacy NATs that are not necessarily aware of HIP. It involves encapsulating HIP in UDP packets and uses keep-alive messages to maintain NAT port bindings.

2.3.1 Survival of TCP Connections for Long Periods of Disconnection

This section discusses two important issues for live migration. Firstly, it describes some of the factors that influence the survival of a TCP connection during live migration. Secondly, it discusses why TCP performance can be influenced by the live migration, right after the migration ends.

Live migration of user environments implies survivability of TCP connections when the user moves to another network, therefore one of the building blocks involved in the architecture for live migration of user environments is a network mobility protocol. However, mobility protocols such as HIP and MobileIP introduce a new challenge to TCP connections: some hosts are connected to the network at different endpoints and can suffer large periods of disconnection when they move between endpoints.

The TCP standard [65] and current implementations were not designed with mobility in mind and do not tolerate large periods of disconnection. TCP provides a user timeout for aborting the connection in case a segment is not acknowledged. The TCP standard allows implementations to choose a timeout value and for most implementations this is a few minutes [78]. This is normally a system wide value that applies for all connections. Some implementations have different timeout depending on the state of the TCP connection (i.e. ESTABLISHED or SYN_SENT). In order to address this issue, a new Internet-Draft [35] proposes the negotiation of a per-connection Abort Timeout Option as a new TCP option. This option allows hosts to maintain TCP connections across large disconnection periods.

For live migration of user environments, the tolerable downtime is likely to be application and protocol specific. The migration process is best-effort and does not offer any guarantees about network downtime. In the worst case, network connections have to be restarted, which is the approach taken in previous work. In this thesis it is assumed that common applications using TCP can tolerate downtime of a few seconds and still ensure an acceptable user experience. This is based on the experience with some applications such as ssh [11] and wget [17] during migration. In this thesis, it is assumed that if the network connection survives during the handover, the migration is successful. If some connections do not survive, the system can revert to the backup approach of restarting them. It is however desirable to have a downtime as small as possible in order to provide a good user experience for all applications.

An acceptable network downtime for migration is hard to define because it is also application and protocol dependent. For interactive sessions, a downtime of more than 30 ms [37] will exceed the human perception threshold and will impact user experience. However, depending on the interactive application, the downtime can be up to 100 ms. For other types of applications such as a web download, the downtime is less important, and it is more important to ensure the survival of the TCP connection instead. Given that the user is not interactively using her environment during the migration, interactive performance is not a critical aspect. However, network connections need to survive the

migration so that the user does not have to restart them. In extreme network conditions longer disconnection times are possible and the user could benefit from larger disconnection timeouts and could change the abort timeout to a large value just before triggering a live migration.

One other important aspect regarding the effects of live migration on TCP connections, is the performance after a period of disconnection. This is because TCP connections will exhibit low performance after periods of disconnection. One of the reasons is TCP's retransmission behavior. TCP doubles the retransmit timeout in case of an unacknowledged packet for every attempt. It stops increasing the retransmission timeout (RTO) after it reaches 60 seconds. Because of this, upon a node rejoining the network, up to 60 seconds can pass before the connection resumes. The TCP Extensions for Immediate Retransmissions draft [36] addresses this problem by allowing TCP to restart stalled connections as soon as it receives an indication that connectivity to previously disconnected peers may have been resumed. Another reason is the TCP congestion window which is reduced to a small value after disconnection. This window is restored slowly after a period of disconnection and this behavior impacts TCP performance: a small congestion window reduces the throughput of a TCP connection and does not allow it to take full advantage of the available bandwidth. These are some of the reasons TCP exhibits low performance after periods of disconnection which occur during the live migration of user environments.

2.4 Storage Replication

Storage replication is another essential building block for realizing the scenarios described in Section 1.1. The user maintains several replicas of her persistent storage on various locations. Storage needs to be replicated efficiently across the various sites. Moreover, block-level redundancy between peers may be exploited. However, storage replication is normally studied in the context of data centers and is an essential component of building a reliable data center. Some related solutions for persistent data replication are discussed in this section.

Several solutions such as RAID [61] provide local backup. Standard solutions include array mirroring, to either local or remote storage. The preferred method is backup to tape drives because of the low cost of such a solution but often in case of failure, the data can

be up to a day old. Moreover, local backup does not prevent all failures such as the case when an entire data center fails. This means that storage needs to be replicated to a secondary site in a geographically different location. Recovery from failure implies fail-over to one of the secondary sites or data reconstruction at the primary site. Live migration of user environments addresses similar issues and uses similar concepts for migrating persistent storage. However, it is more complex than simply providing local backup because replication is not local, but instead it is over wide area networks.

Disaster recovery has several similarities to the live migration of user environments and will be discussed in detail in this section. Many results pertaining to this topic can be used without any modification to assess a good solution to the live migration of persistent storage for user environments discussed in this thesis. This is mainly because the replication of persistent storage is a very similar problem. There are many available solutions for designing disaster-tolerant solutions that have different tradeoffs in terms of dependability, cost and complexity. The main tradeoff is in terms of bandwidth availability and the degree of dependability offered and the use of either an asynchronous or synchronous protocol. At one end of the spectrum, synchronous protocols offer a high degree of dependability at the expense of network bandwidth used, while asynchronous protocols are less dependable but use less network resources.

A framework for automatic generation of dependability solutions is given in [49]. It takes into account dependability metrics as well as cost and models of data protection alternatives. The degree of dependability refers to the degree of tolerable downtime (*Recovery Time Objective (RTO)*) and the amount of data loss (*Recovery Point Objective, (RPO)*). The data loss metric gives the maximum time window for updates to be lost. Live migration of persistent storage is a particular case of storage dependability with a RTO of less than a second or a maximum of a few seconds and a RPO of 0 (no loss is tolerated).

Storage replication techniques are sensitive to the *average update rate* and to the *unique update rate*. The *average update rate* is the number of written blocks in a certain interval and the *unique update rate* is the number of unique blocks written in a certain interval, not counting the overwrites.

Synchronous mirroring solutions limit write bursts within a certain interval. Depending on the workload, write bursts are typically 3-10 times [49] larger than the average update rate.

For synchronous mirroring, the secondary needs to apply the write and confirm it before the write completes at the primary, therefore a low network latency is required. However, no data loss occurs. Therefore, in order to accommodate short term bursts without slowing down applications on the primary, the following available link *bandwidth* is required:

$$bandwidth \geq averageUpdateRate \times burstMultiplier \quad (2.1)$$

In this equation, *bandwidth* is the link bandwidth, *averageUpdateRate* is the average rate at which data is written and *burstMultiplier* is the short-term update-rate multiplier. This equation that gives the minimum network *bandwidth* required for synchronization is justified by the fact that the link bandwidth must be enough to accommodate the bursts arriving at the *averageUpdateRate*.

Two other replication strategies studied in [49] are relevant to live migration of persistent storage of user environments as well: *write-order preserving asynchronous mirroring*, *async* and *batch asynchronous mirroring with write absorption*, *asyncB*.

The *async* protocol commits the writes to the secondary in the order of arrival without coalescing writes. Therefore, updates are propagated to the secondary at a lower rate than the peak rate they are written on the primary. This protocol can be used to accommodate write bursts that might slow down application workloads. It also requires a write buffer large enough to accommodate the write bursts of the specific workload type. Data losses can occur if the primary fails while the write buffer is not empty as described in 2.2 and 2.3.

$$bandwidth \geq averageUpdateRate \quad (2.2)$$

$$dataLoss = \frac{writeBufferSize}{\min(averageUpdateRate, bandwidth)} \quad (2.3)$$

In this equation *dataLoss* is the data loss probability, *writeBufferSize* is the size of the write buffer used to store outstanding writes, *bandwidth* is the available bandwidth that can be used for replication, and *averageUpdateRate* is the average rate at which data is

written to the disk. Equation 2.2 is justified by the fact that the write buffer must be able to absorb the bursts. The buffer can be drained only if the average update rate is smaller than the available bandwidth. Of course, data losses are possible, depending on the size of the write buffer relative to the average update rate and the bandwidth.

Equation 2.3 justifies that the data lost is the data present in the write buffer at the moment the primary fails. The data is constantly being sent from the buffer to the secondary at a rate which is the smallest of the average update rate and the bandwidth.

The *ascynB* [62] protocol minimizes the bandwidth used by coalescing writes in batches at the primary write buffer before they are sent over the network. The bandwidth required for such a protocol is given by the *uniqueUpdateRate*. However, in case of failure of the primary site, the batch of in progress updates and the one being sent over the network will be lost. The bandwidth requirements for live migration of user environments will be discussed in Chapter 3 as they are different for a number of reasons. Firstly, because data losses are not tolerated and secondly because user workloads instead of server workloads are addressed, a slight slowdown in the performance can be tolerated. Moreover, user workloads are characterized by bursts followed by large pause times.

The previously described *asyncB* protocol is implemented in SnapMirror [62]. SnapMirror uses asynchronous mirroring for Network Appliance file servers. It ensures consistency by taking consistent snapshots from the primary and forwarding them to the secondary site. It takes into account that the destination system must be in a consistent state in case of primary failure. SnapMirror is tailored towards the WAFL file system [40]. It exploits file system data structure information to distinguish changed blocks and to avoid sending deleted blocks (WAFL is a no-overwrite file system) over the network. Thus, SnapMirror is able to reduce the amount of data transferred by 30% to 80% for a loaded server for update intervals of 1 minute and obtains even better performance if the update interval is larger than 1 minute. This suggests a very interesting conclusion for the work in this thesis. Live migration of persistent storage of user environments could use less network bandwidth if a protocol similar to *asyncB* would be used.

Another option is to perform synchronization at file level. Programs like *rsync*, *rdist* [82] and *Csync2* [2] are file synchronization tools. Given that VM storage is commonly backed up by files, this could be an alternative for implementing the live migration of persistent storage. However, these tools have the main disadvantage of being slow. Operating above the file system has some advantages over the block-level

synchronization approach: even a small number of bytes written to a file translate to many data blocks being written at block level (metadata writes). However, a small change to a file usually results in the need to re-sync the whole file. Rsync uses CPU to compute byte range checksums and transfer only a smaller range from the modified file. On the other hand, block layer replication solutions have lower complexity (they only need to traverse a block map instead of the directory structure), can optimize for reads and writes (can read and write blocks sequentially rather than logical blocks at the file level) and are therefore faster than file level replication. This was the main motivation for implementing persistent storage replication at block level in this thesis.

The previous work regarding storage replication, even though it was in the context of storage replication for servers, is very useful in describing the basic principles behind storage replication for migration of user environments. All workloads have a certain degree of bursty behavior and depending on the consistency requirements and the available network bandwidth, it is better to perform asynchronous or synchronous replication. The choices made in this work will be further explained in Chapter 3.

2.4.1 Mirroring Virtual Disks

The previous section described the protocols for synchronous and asynchronous replication and compared the approaches of file level replication and block-level replication. This section overviews existing technology for migration of persistent storage when the storage is represented in a specific way, by virtual block devices. Virtual block devices are in fact virtual disks implemented by virtual machine technology.

For the purpose of live migration of user environments, one option is to encapsulate the entire operating system as a virtual machine as in [77]. In this case, the entire disk of a VM is a virtual block device that can be backed by different types of resources on the host virtual machine, such as a file or a partition. If the user environment is packed as a virtual machine as in previous work and in this thesis, live migration has to deal with the live migration of virtual disks.

This section overviews solutions that can be used to migrate virtual disks. Virtual disks are sometimes easier to migrate because they can be accessed at file level by the host operating system. This is easier because virtual disks are in fact files in the host operating system, thus the migration can be done by userspace tools ([84]).

Solutions for mirroring virtual block devices (VBDs) can either rely on mirroring files or on mirroring block devices using specialized software that requires kernel level code. In the later category fall Software RAID [10] and Distributed Replicated Block Devices (DRBD) [69]. DRBD can be used to replicate block devices over the network and can be used for replicating virtual block devices based on real block devices as well.

The alternative to mirroring VBDs based on block devices, is to replicate file based VBDs using existing tools such as rsync. This is a good option when using non-live VM relocation. However, it is not an appropriate method for live migration as such schemes do not readily provide a way to synchronize the blocks that are dirtied during the migration process. This is the reason this approach was not chosen in this thesis. The rest of the section deals with block device replication of the VBDs.

For mirroring Logical Volume Manager (LVM) partitions for non-live relocation, a setup with Software RAID or DRBD can be used. Live migration is not directly supposed with these tools, although they do offer a number of features that are useful for live replication of virtual block devices. I have implemented a DRBD based virtual block device migration tool, specifically addressing LVM partitions. Therefore, DRBD's features will be discussed in more detail, in order to justify why it is an efficient solution for migrating VBDs using low latency network links.

DRBD is used to build high availability (HA) clusters with replicated storage by mirroring block devices over a network. It uses TCP as a transport protocol to solve issues like packet reordering. It performs intelligent resynchronizations by keeping track of active regions of the disk. DRBD is able to find the up-to-date replica without administrator intervention if the cluster is restarted.

DRBD supports three protocols labeled A, B and C. Protocol C is completely synchronous. Protocol B signals I/O completion to the upper layers as soon as an acknowledgment packet arrives from the secondary node and the local write operation is committed to the local disk. Protocol A signals I/O completion to the upper layers as soon as the write was committed to local storage and was sent to the secondary.

Each DRBD device can have one of two possible roles: *Primary* or *Secondary*. The main difference is that a secondary device is not writable. DRBD supports a configuration in which both nodes are primary, therefore they are both writable at the same time and protocol C must be used in order to ensure consistency.

In the current implementation, the meta-data disk used for this purpose takes 128MB no matter what the hard drive size is. The meta-data disk can be stored internally or on another disk and is an overhead for migrating small disks.

DRBD takes a different approach from SnapMirror to ensure consistency for asynchronous mirroring protocols. It cannot make use of snapshots but it must ensure that the secondary node is always in a consistent state and the write order is maintained. Therefore, there are two approaches:

- Perform each write block request before the next block from the primary is accepted. This approach would not make good use of the disk scheduler and I/O subsystem on the secondary node because it does not schedule large transfers to disk.
- Identify write dependencies on the primary node and insert barriers where appropriate between write requests. When the secondary encounters a barrier it waits for all the requests to complete before processing any further requests.

SecondSite [30] is a work in progress system designed for high availability and disaster protection. Instead of addressing storage replication and backup to a remote persistent storage site, it addresses whole system checkpointing to remote persistent storage. The goal of this system is to virtualize the entire hardware and to provide disaster recovery without any modifications to the operating system. Moreover, it is transparent to the outside network since it aims at minimizing the downtime between failure and service continuation by migrating TCP connections. SecondSite also attempts to minimize the bandwidth requirements for replication by content hashing, dirty page compression and throttling page dirtying rate. However, this is work in progress and actual details about the architecture and results are not currently available.

This work is similar to live migration of user environments because it attempts to resume whole operating systems, including persistent storage at a remote location, with minimal data loss and while maintaining active network connections. It also differs because in the case of user environments, the secondary location does not need to be consistent at all times. Moreover, the workloads and available bandwidth are much lower than the ones likely to be used for disaster recovery of data centers.

2.5 Migration of Virtual Machines and Environments

The previous sections described the background related to my work. This section describes similar projects to the one described in this thesis, such as the Internet Suspend Resume [77] project. This section first describes live migration of virtual machines (2.5.1). The next section (2.5.2) describes how regular (not live) migration of virtual machines has been used in previous work that addresses the non live migration of user environments. The approach used in these works is to pack the user environment as a virtual machine and use a suspend/resume migration approach along with a number of optimizations to reduce the amount of network resources consumed by migration. Since the work in this thesis uses live migration for the mobility of user environments, it is at the intersection of the works described in the next two sections.

2.5.1 Virtual Machine Live Migration

Xen live migration [28] has a number of useful features for data centers, facilitating fault management and load balancing. A VM can simply be relocated with minimal service down time using a pre-copy technique to migrate the memory while the machine is still running. This is an iterative copy process that starts with first copying the whole memory. Xen live-migration benefits from the fact that the hypervisor's control interface has access to the migrating machine's memory pages and is responsible for creating and destroying VM instances.

The hypervisor can keep track of pages that have been dirtied since the last iteration and just copy those at the destination. Finally, a *writable working set* of pages is sent over the network along with a checkpoint of the state of the OS and the machine is resumed at the destination. The downtime obviously depends on the size of the writable working set which is dependent on the workload's patterns of updating page tables.

Xen implements hosted-migration in which the host control software governs the migration process. However, the first approach to live-migration was self-migration [39]. Self migration is done without hypervisor involvement using a similar iterative pre-copy mechanism. The guest OS has to be modified to keep track of its writable working set and the transfer to the migration destination, as well as suspending itself from the originating machine.

2.5.2 Migration of Virtual Environments

This section discusses the related work related to the migration of user environments. It focuses on the techniques developed for migrating user environments, especially on the optimizations aimed at minimizing network traffic.

The Collective: The *Collective* project [75], [74] addresses moving entire x86 operating systems from one machine to the other using the VMware GSX server virtual machine monitor. The x86 machines are encapsulated so that they contain the whole memory and disk space. The most important problem when migrating this amount of data is the network bandwidth. In order to minimize the amount of data sent over the network, the *Collective* made the following optimizations:

- *Copy-on-write disks (CoW)* - only the modifications from the previous snapshot of the system are recorded. Thus, the amount of data that needs to be transferred is proportional to the modifications to user data. Maintaining snapshots of both the disk and the main memory ensures that the filesystem is kept in a consistent state.
- *Demand paging* A capsule's execution can be resumed even if not all the pages are present in memory. The pages that are not on the local disk will be fetched over the network.
- *Ballooning* the unused memory - this optimization decreases the size of the memory being transferred over the network at the expense of losing some of the performance provided by the buffer cache.
- *Hashing* disk blocks for efficiently mirroring virtual disks. Hashing helps to reduce the traffic in the case some of the disk chunks are already on the disk. Checking the hash values for every chunk is not bound by the CPU because the work addresses low bandwidth links.
- *Gzip* all the traffic.

Additional optimizations include CoW disk hierarchies and a hash cache. Their system supports only non-live migration, therefore if the machine is running, it is first suspended to disk and then resumed on the other host. The advantages of their approach and in fact of every VM migration based system are:

- access to the same interface from anywhere in the world
- there is no loss in interactivity
- easiness of maintaining the OS
- an open system that allows users to chose their OS and preferred software

The *Collective* project shares many common ideas with the work in this thesis. Most of the advantages and optimizations done in the *Collective* are also present in my work. The main difference is that they do not support live migration and network connection migration. Some of the optimizations such as copy on write and gzipping the traffic are not implemented in this thesis but are likely to prove useful.

Grid live migration: [81] investigates Xen live migration supporting Grid operations over long haul MAN or WAN high performance networks. The necessity to live migrate machines from a cluster comes from multiple usage scenarios. Firstly, without migration it can be impossible to bring computation closer to the data. Migration mitigates high network latencies to the data centers. Secondly, migration achieves dynamic load balancing of computation workloads over domains that exceed the boundaries of a local data center. Additional gains are limitation of power consumption and better management of workloads.

The network links used are 1Gbps between distant sites like Amsterdam and San Diego, with a RTT of 198 ms. For these types of links, TCP suffers from slow start, therefore live migration with minimum downtime is essential because it maintains network connectivity and high throughput. To ensure external WAN connectivity they make use of Translight [32], which is designed to create IP tunnels for high performance links. The long-haul migration demands that connectivity is always maintained at layer 3. This is to ensure that the machine stays connected before, during and after migration. On the other hand, migrating across domains would generally imply that the VM would get a new IP every time. Since this would break TCP connectivity, a tunneling mechanism invisible to TCP is necessary, which implies that the migrating machine can keep its IP address. This approach has the downside of maintaining residual dependencies to the originating site, therefore it does not meet the goals of this thesis.

Compared to the work in this thesis, this work does not address virtual block device migration. As future work, they propose the use of LVM snapshots in order to make

Copy-on-Write disks and transfer them over the network. Still, for live migration, this is unlikely to work: during migration, before check-pointing the domain on the source host, there are writes that need to be mirrored to the destination host. Therefore, simply copying a CoW disk to the destination would not solve the problem, rather an iterative process, such as the one used to migrate the memory would do. Without the overhead of migrating block devices, the downtime is between 0.8-1.6 seconds. While the RTT was almost 1000 times higher than for the Gigabit LAN used by [28], the downtime grew only about 5-10 times. However, the downtime perceived by the client application - a ping with 50ms between probes - was 3908ms. This project showed some very important results that influenced the design of the the solution presented in this thesis. It shows that live migration of the memory without the persistent state can cope with wide area links.

The project is related to the work in this thesis because it uses live migration, even though it does it in a different environment. They do address the problem of network connection migration using tunnels which introduces the problem of backward dependencies and is not suitable for migration in a mobile environment such as the one discussed in this thesis. Moreover, they do not support the migration of persistent storage.

MobiDesk: MobiDesk [21] is an approach to user mobility based on the thin client approach. The focus of this approach is not on allowing the user to make use of the local hardware at full capacity. Instead, the focus is enabling a scalable fault tolerant mobile virtual computing infrastructure. All the persistent state of the user environment is stored on a cluster of remote servers. The terminals used by the user are simple *input/output* devices. A user's computing session is abstracted into three areas: display, operating system and network. A thin virtualization layer is placed on the hosting servers in order to allow session continuation in case of server fault or maintenance. MobiDesk offers a host independent view of the operating system, allowing a user session to be migrated within the cluster from one host to another without service discontinuation. Network session continuation is enabled by a transport protocol independent network proxy common to all the hosts in the cluster.

The advantages of MobiDesk over single host virtual network computing are that it offers highly available hosting for user environments and persistence and continuity of business logic. The hardware and host's operating system can be upgraded while the user session is migrated to another host. User data is only kept at secure hosting locations and is automatically backed up, thus avoiding current slow practices to back up and restore the operating system and user data. Moreover, MobiDesk does not require the user OS and

applications to be modified or recompiled and only requires running a userspace utility. One important downside is that the user experience and performance in MobiDesk are likely to be dependent on the network quality and available bandwidth of wired network.

SoulPads: Portable SoulPads [26] is an approach to user mobility closer to the one proposed in this thesis. It starts from the assumption that any computer can be used as a hardware platform for running the user's personalized environment. In order to achieve this, the user has to carry a portable bootable self-configuring operating system, as well as a hypervisor and a snapshot of the whole state of the operating system. The self-configuring operating system takes care of detecting the hardware and running the hypervisor which in turn is responsible for starting the guest user environment which is packed as a system virtual machine.

Their approach can take advantage of pervasively available hardware which is the same assumption present in this thesis. However, the traffic is not sent through network connections and the user capsule is instead suspended to a portable storage device. This means that the approach does not support live migration and also implicitly has to restart the network connections upon resuming the user capsule at a new location. This is not a limitation of the work in this thesis which offers the possibility to perform live migration at the expense of generating network traffic.

Internet Suspend/Resume (ISR): *Internet Suspend/Resume (ISR)* [77] was the main inspiration for this thesis. Their approach assumes the following about a future in which hardware is pervasively available: it is unlikely that the need for office work, email reading, etc will go away in the future. Public places are likely to provide customers commodity computers that can be used during waits, such as in cafes, doctor's office, etc. Moreover, the user is likely to work from different machines both at workplace and at home. This *hands-free* approach should present the user with the same interface all the time, just like she would do by carrying a laptop without sacrificing interactivity, interface capabilities and speed.

ISR's approach is to use distributed file system technology to store the suspended state of a user's virtual machine. The machine can be resumed later by reading its state from the distributed file system. Their prototype uses the *Coda* [76] file system for this purpose. Distributed file systems are a reliable way to store data and provide a backup of the user's VM at all times. I argue that keeping the user's state on a distributed file system is a good solution for medium and large organizations such as a university but it is not appropriate

for a large scale use mainly because of the administration involved in maintaining the distributed file system. For the wide area, a completely decentralized self organizing Peer-To-Peer storage solution would scale better for the same purpose. Coda is augmented with a look-aside caching from a portable USB drive. This dramatically decreases the resume times. However, given the fragile nature of such portable devices, they can only be considered an optimization and cannot be relied upon for consistency. Content Addressable Storage is suggested as a possible substitute for portable storage.

Since storage of virtual disk images makes use of distributed file systems, caching of blocks raises the question of finding out the best chunk size, in terms of cache miss rate and the required bandwidth. Each block is stored as a Coda file and makes use of Coda's file caching capabilities. If the chunk size is increased, the cache miss rate will go up. A chunk size of 128KB is found to be a good compromise between the bandwidth use and cache miss rate. Even though this thesis does not use the same techniques, the result itself is very useful if the solution would be extended to exploit block-level similarity between peers.

ISR provisions for situations in which it can be predicted where the next resume is going to take place, by using caching within the distributed file system. This is feasible because the suspended virtual machine is made by files that can be easily updated in the distributed file system after it has been suspended and is no longer being actively changed. However, live migration implies that the image of the guest capsule is continuously changing and such an approach is not feasible due to the large number of updates that need to be mirrored constantly. Moreover, a distributed file system imposes strict consistency on the global state of the file system and is unlikely to scale well over heterogeneous links and a large number of users.

ISR shares the same goals and vision with the work in this thesis. The main difference is that it provides only a stop/resume capability and cannot provide live migration. Moreover, the way to migrate the persistent storage is fundamentally different. ISR uses a file system and my work uses dedicated network connections. One advantage of ISR is that user capsules are safely stored on the distributed file system which is a good way of implementing backup, while the live migration approach in this thesis does not specifically address this. A disadvantage of ISR and an inherent disadvantage of the stop/resume approach is that network connections are not migrated, and they have to be restarted upon resume.

Dimorphic Computing: This work [53] introduces a computing model based on switching between thin and thick client modes. The model improves the performance of applications that alternate between CPU/data intensive and user interactive phases such as 3D intensive applications, visualization, video editing, etc. It achieves this by allowing interactive applications to use the user's desktop GPU for crisp interaction. It supports automatic switching between thin and thick client modes based on Xen VM migration. The type of bandwidth used is 100 Mbps with a 33, 66 and 100 ms delay, typical for high performance WAN links, therefore HPN-SSH [68], a secure transfer protocol optimized for WAN links is used. Migrating a VM's virtual block device over such links is not scalable even when using such optimized protocols. Existing solutions like Storage Area Network (SAN) and distributed file systems do not scale and act as a performance bottleneck. *Dimorphic Computing* uses a peer-to-peer approach for distributing chunks from the machine's block storage device. Virtual block devices are synchronized using *rsync* in the background. Even though this work suggests the use of peer-to-peer approach for storage replication, the gains are not evaluated.

Migration across subnets is supported by VNETs [79], a layer-2 proxy. While using VNETs, both source and destination subnets need to stay connected for the whole period. This means that this solution does not address the *no residual dependency requirement* of live migration of user environments. To ensure that this requirement is met, the migration across domains and thus across subnets should be supported by a solution that is completely independent of routing the traffic through the source's subnet, such as the one proposed in this thesis.

Followup work based on ISR [60] makes use of data gathered from a live deployment of the ISR enterprise client management system based on VMware virtual machines over a period of seven months. Various policies for storing client content are evaluated and the use of *Content Addressable Storage (CAS)* [56] [59] and *gzipped* images are compared. These policies influence the network bandwidth used, as well as the storage requirements but ultimately have an effect on user privacy. This study demonstrates that there is a trade-off between network bandwidth and storage efficiency and privacy. Moreover, it quantifies the effect that chunk size has on the performance to show that CAS has a better compression rate than gzip for small chunk sizes (smaller than 64KB). Given the design of CAS storage, the more redundancy that exists between previous snapshots belonging to all users or between previous versions of the same user's capsule, the better the compression rate.

This shows an important result: splitting block device data and memory content into smaller pieces increases the chance of redundancy which is very well exploited by CAS storage. Thus, CAS seems a scalable solution that exploits data commonality between real workloads. This idea is also exploited in the *Pastiche* [29] Peer-To-Peer backup system for commodity computers. Data commonality is an aspect that is not exploited by the current architecture for live migration presented in this thesis but would clearly improve the performance of live migration and should be exploited by future work. This could be done simply by integrating with *Pastiche*. In *Pastiche*, excess disk capacity is used to store data on behalf of other peers. *Pastiche* builds on three existing technologies: CAS to provide redundancy data within similar files, convergent encryption [24] to allow sharing of encrypted representations of the same data and the Pastry [71] peer-to-peer network for routing and locating data. Thus, *Pastiche* provides block-level data sharing without losing privacy as well as content based indexing and a self organizing network to facilitate finding similar disk blocks from peers with similar installations.

There are several similarities between the scenario assumed by *Pastiche* and the scenario assumed by this thesis. In both cases, peers form a self organizing network and exploit the similarities between persistent storage data. However, in the case of live migration of user environments, the locations where the user is making use of her environment contain replicas with a high degree of overlap. Since *Pastiche* performs better when there is a high overlap between peers, I argue that an architecture based on the same technologies could be used as a scalable underlying backup and replication layer for live migration of user environments.

In parallel with my work, another approach to perform live migration of persistent storage was developed in [25]. This work does not address the same scenario of user environments migration because it deals with server migration. It also does not handle the migration of network connections using a mobility protocol and instead relies on manually set up tunnels. Their implementation is based on Xen. From the implementation point of view, it enables live migration of a Xen guest VM and adds the capability of migrating the virtual block devices. Their work reports that a running web server can be migrated including persistent storage in 3 seconds in a LAN and in 68 seconds in a WAN. The design uses the block tap [84] and also performs synchronization at block level by intercepting write requests and generating a *delta*, which is a communication unit that consists of the written data and the location on the disk. Therefore, even though both our works address the live migration of persistent storage, there are differences in both goals and implementation. Technically, it should be possible

to integrate their work related to persistent storage migration into the implementation (Chapter 4) described in this thesis.

2.5.3 Persistent Storage Management

In the scenario proposed in Section 1.1 user capsules are stored as VM images at various locations. It is not necessary to have a separate hardware machine for each location where the user will migrate, instead it is likely that multiple VM images will be stored on a single hardware machine. This highlights the need to be able to manage a number of VM images, each with their associated persistent storage. This topic has been studied in the context of computer clusters that use a large number of VMs and some of the projects dealing with this are discussed in this section. The topic of managing multiple VM images is not specifically studied in this thesis and is subject to future improvements.

Persistent storage management for virtual machines is a key challenge in deploying virtual machines in commercial data centers and scientific clusters. Parallax [85] proposes a design aiming at managing the virtual disk images for the predicted increase [23] in the number of virtual machines instances used in such data centers. The design goals of Parallax are to keep historical snapshots of the disks and to allow efficient snapshots of an OS's disk and memory every thirty seconds. This feature would be valuable for debugging purposes or for intrusion detection. The fact that most disk images will inherit from a small set of base images and any virtual disk image is associated with at most one virtual machine instance, substantially reduces storage requirements and eliminates the need for a write lock that is inherent in systems using a distributed lock manager. Parallax offers an interesting approach for managing multiple disks pertaining to many users of the live migration approach and suggests a number of improvements to the work in this thesis. For instance, keeping historical snapshots is a desirable feature that could help users of the live migration system presented in this thesis cope with data loss. Another work that aims at providing archival persistent storage support is Venti [67]. Venti is motivated by the growth in storage capacity coupled with dropping prices of hard drives and builds a block-level storage system that uses a write-once [66] policy to create an archival storage system shared by multiple clients without sacrificing too much on the performance side.

Many other approaches can be used for managing the persistent state of a user environment. The ISR project uses networked file systems such as Coda [76], AFS [41]

or NFS [63] which rely on a few servers to export file systems to many clients. Another option is to store the persistent state using LBFS [59]. LBFS is a network file system designed for low bandwidth links. It exploits similarities between file versions to save bandwidth. Similarities can also be exploited at the block level. For example, Petal [54] and Federated Array of Bricks (FAB) [73], [42] exploit block-level distributed storage and provide snapshot facilities. This thesis however does not address storage management at the moment and has no way of exploring similarity between several environments belonging to the same user or between environments of different users. However, this is not a limitation of the approach and could be added as future work.

2.5.4 Xen Security

The main challenge involved in designing a platform for live migration of user environments is how to ensure that a remote host has not been compromised. Users need to be able to perform remote attestation of the hypervisor and host OS running at the destination. This section discusses two approaches to provide security for Xen VMs. These approaches could be used to provide security guarantees to the users of the live migration platform.

[34] discusses the trust management issues that arise from the interaction of VMs in the XenoServer Open Platform (a public infrastructure for wide-area computing). It also presents XenoTrust - a trust management architecture for this platform.

The XenoServer Open Platform consists of execution platforms scattered across the globe and available to the public. The advantage is that users are able to run programs at various locations in order to minimize communication latencies, avoid bottlenecks, etc. The platform can be used to deploy wide area experiments and to provide a point of presence for transiently-connected mobile devices.

Clients pay for the resources they are using and operators are paid for running the programs they are hosting. Given the open nature of the platform, trust becomes a crucial aspect. Both servers and clients can take advantage of numerous ways to cheat. Servers may not run programs faithfully, may over-charge or may miss-handle secrets provided by the clients. Clients can try to cheat the system by trying to abuse the platform or by providing false payment credentials.

This platform introduces the risks that are inherent in P2P and numerous other open distributed systems. It has to balance the security constraints and authentication mechanisms supported by users, the open nature of the platform, and anonymity (how easy it is to create new pseudonyms). It is also exposed to groups of malicious users. One other problem is how to deal with the reputation for newly added clients or servers.

The scenario sets this work apart from Peer-To-Peer and recommendation systems. Firstly, the interaction between servers and clients involves running real tasks. Secondly, there is real money involved in payment for services. Various servers may have different notions of what normal behavior is. This makes it impossible to assign any global significance to trust values. The trust value mechanism could be applied for the live migration of user environments as well but it provides weak guarantees. Malicious servers would in time obtain a low trust value but only after providing malicious services to some user capsules.

IBM [22] has done some work in virtualizing the Trusted Platform Module (TPM) secure storage and cryptographic functions so that they are available to applications running in the virtual machines. This facility supports higher level services for establishing trust in a virtualized environment, such as remote attestation of software integrity. Support for securely live migrating the Virtual Trusted Platform Module (vTPM) is also provided.

The virtual machine monitor ensures that there is a good isolation between the VMs running on a certain host. The TPM is able to ensure resistance against software attacks and can provide attestation for the software running on a physical machine, from the hypervisor itself to the OS and all the applications running inside guest VMs. This can be achieved by virtualizing the TPM so that all the VMs believe that they have access to a TPM platform.

There are two major challenges:

- Integrating the vTPM to another machine while maintaining the secrets intact.
- Maintaining the association between the vTPM and the underlying trusted computing base after a migration.

An approach to secure remote hosts based on the TPM is more likely to provide a good solution for the live migration of user environments. The user would be able to perform

remote attestation of all the software layers which are responsible for running the user capsule at the destination, before triggering migration.

2.6 Quantifying Interactive User Experience

A very popular way to access a user capsule from any host is to use the thin client approach. A thin client such as VNC [15] is a client-server approach to user mobility in which the computer resources are accessed from a remote server and forwarded to the user's terminal. The user's keyboard and mouse input is forwarded to the remote server by the thin client terminal and the output is forwarded over the network back to the user's screen. It has the advantage that clients and servers do not need to run the same OS. However, it is not a good mobility model because the user's software is running at a central location and thus breaks the residual dependencies requirement.

The interactive user experience varies highly with the demands of the application and with the available network conditions. A response time of less than 150 milliseconds is considered ([80]) a limit for crisp interaction that does not interfere with productivity. The response time becomes noticeable or even annoying between 150ms to one second and unusable beyond this. These studies are appropriate for measuring the interactiveness on thin clients where each user action is forwarded over the network to a VNC server and then the client receives the screen updates. Therefore, latency is a very important network characteristic.

The solution for live migration of user environments does not have these drawbacks because at all times the user is running the user capsule locally on available hardware with crisp interaction. The thin client approach is very sensitive to network conditions and delay throughout the entire user experience while the live migration approach in this thesis is sensitive to network conditions only in the initial copy phase of the memory and persistent storage.

2.7 Live Migration of User Environments

Previous work addressed either the live migration of servers or the migration of user environments by explicitly suspending the user environment and resuming it at a later time. None of the previous works addressed the scenario described in 1.1. To the best of my knowledge, there is no other work that studied the live migration of whole user environments including persistent storage and network connections across wide area network links.

This section further motivates why a new architecture that helps users cope with mobility is needed and overviews the limitations of the previous work.

One of the most common approaches to cope with user mobility is to use a laptop. The laptop offers a powerful computing environment but is fragile, heavy and can be stolen. A more lightweight alternative is to use a mobile device such as a PDA. However, these devices are often too limited in terms of computing power and they have small displays and reduced interaction capabilities. Therefore, users prefer to use them only when they are on the move and then synchronize their data with their laptops or computers.

One other common alternative is to use a remote connection from various thin clients to connect to the user's environment. This approach however is very sensitive to network delay and network disconnections. Moreover, it requires an always-on network connection to the network where the computing environment is located.

These shortcomings motivated this work and similar projects such as the Collective and ISR. The key idea is to allow the user's environment to be transferred to the destination where the user can take advantage of pervasively available hardware and run the same user environment on the available hardware.

However, previous work does not provide the option to keep the user environment running during the user's commutes. This means that migration is not done *live*. Live migration of user environments is a highly effective and useful way to use a personal computer environment. This approach does not have the overhead of suspending and resuming which means that the network connections do not have to be restarted, and the user session is exactly the same as if the user simply returned to the same computer to resume work.

The work in this thesis is based on existing work on live migration of servers [28]. Their architecture however targets a specific scenario which is significantly different from the mobility of user environments. The main difference is the fact that live migration occurs within a cluster and there is no need to migrate network connections and the persistent storage. Moreover, the workloads migrated are server workloads which are much more demanding than the typical user environment workloads.

Therefore, a new architecture that enhances live migration and adapts it for what a user would expect from her environment is needed. This entails two main challenges. The first challenge is to migrate persistent storage along with the in-memory state of the user environment. The second challenge is to perform the migration without disconnecting the network connections. None of the previous work addresses both of these challenges. The architecture proposed in Chapter 3 provides a solution to both of these challenges by incorporating live migration of user environments including persistent storage and at the same time migrating the network connections.

2.8 Conclusions

This chapter gave an overview of the related work. It discussed the building blocks of the solution presented in this thesis: virtual machine technology, mobility protocols, storage replication and related work about the migration of user environments. The work in this thesis is based on all these technologies and aims to provide some features that are missing from the previous work on migration of user environments. These features are live migration, the lack of backward dependencies on the source host or on a centralized server, the migration of network connections across the wide area. My work is most similar to projects such as ISR and the Collective which were the first to propose the migration of user environments. The main goal of these works was to provide migration on pervasively available hardware, to allow the user to resume work seamlessly and to minimize the amount of network traffic generated. My approach also assumes pervasively available hardware but tries to improve on allowing the user to resume work seamlessly by providing the live migration capability. The next chapters will further discuss the live migration of user environments and will describe the architecture and implementation of live migration of user environments, which are contributions of this thesis.

Chapter 3

Architecture

This chapter describes the general architecture I propose for live migration of a user's operating system across wide area network links. An initial proof of concept realizing this architecture is presented in Chapter 4. The architecture facilitates the task of *live migrating* fully functional operating systems, including memory, persistent storage and network connections across various user commute locations, with unnoticeable downtime: to the outside world, it seems that the user environment is never disconnected from the network.

I will refer to the migrating operating system as a *user capsule* which is in fact a guest virtual machine running on top of a hypervisor. As described in Section 2.5, live migrating an operating system is in fact a process of iterative replication. However, the final step of live migration implies a small downtime which depends on both the workload and the available network bandwidth. [28] shows that live migration times of 165 ms are achievable for a loaded web server if migration is performed between two hosts residing inside a cluster.

Since this thesis addresses wide area networks, the downtime is expected to increase but should typically be no more than a few seconds. Therefore, the impact on the guest user capsule should be tolerable by most userspace applications. Moreover, given that the migration occurs while the user is commuting, the downtime due to the migration should not impact the user experience significantly. Migrating the network connections means that they are not restarted during the live migration. This is a significant usability advantage. The solution takes into account that commuting locations are not necessarily

in the same subnet and therefore provides the means for migrating between different subnetworks.

The scenario addressed is seamless user mobility across various locations such as home and work computers. The user will have the option to trigger migration of her capsule towards the required location and the migration can occur during her commute. The important aspect is that the user capsule will remain connected to all services, thus eliminating the need to suspend, hibernate or shutdown any of the operating system, userspace applications, authenticated sessions and network connections. The architecture will describe the various software components that needed to be created and integrated in order to make such a scenario possible.

3.1 Discussion of the Factors that Influence the Total Live Migration Time

The total time for live migration is expected to be dependent on several factors:

- network bandwidth.
- size of the persistent storage and main memory to be transferred.
- workload characteristics.

The proposed architecture uses a point to point replication between the host where the guest capsule is currently running and the destination. This means that only one possible location is being updated at any given moment. Even though it is not addressed in this work, a suggestion for future work addressing this shortcoming is to use a proactive replication mechanism. By tracking the movements of a user using a GPS device, the next resume target can be predicted. This should allow a proactive replication mechanism to update the possible locations.

Once most of the state is synchronized, the regular live migration process can begin and only copy the final blocks that are not present on the destination as well as the check-point data. This should normally incur small delays even for low bandwidth links.

Using a peer-to-peer technology to sync all copies of a user's capsule across various domains should prove very effective. However, this is an optimization and currently the architecture proposed in this thesis describes only the unicast approach.

The architecture proposed avoids all the problems of using the thin client approach overviewed in Section 2.6, such as backward dependencies to the server and poor interactivity that depends on the network delay between the client and the server (the network delay influences the user experience at all times).

However, it is still possible to degrade user performance. Firstly, network performance will be affected during migration due to the additional network traffic generated by the migration. Secondly, if write requests have to be throttled to allow live migration to be tractable, user performance might be impacted severely. Only write intensive processes will be slowed down by migration. Instead of quantifying interactive user experience, the performance evaluation that will be employed to evaluate the prototype relies on measuring the performance of various I/O and network tests and benchmarks.

3.2 General Architecture

The general architecture I propose for live migration of user environments is shown in Figure 3.1. The user environment containing application software accesses the resources provided by the underlying virtualization layer. Depending on the type of virtualization layer below the user environment, the user software and available resources are different. The virtualization layer manages the access to the backing hardware resources which can be general enough and portable, such as network interface card or block device, or specific such as speakers or a DVD drive. The specific backing resources cannot be migrated, while the general resources can be migrated to a different machine.

A virtualization technology was chosen for two reasons. First, it makes the migration process simpler: the environment already comprises user files and settings as well as the current state of the system. Secondly, existing virtualization technology allows systems that run on the bare hardware to be encapsulated as a virtual machine. This means that users can use a system they are already familiar with. Moreover, virtual machine technology is able to virtualize a set of features and devices of the real hardware and it does so without incurring too much overhead. These make virtualization technology a

perfect candidate for encapsulating a user environment, which is also the approach that was taken by previous work.

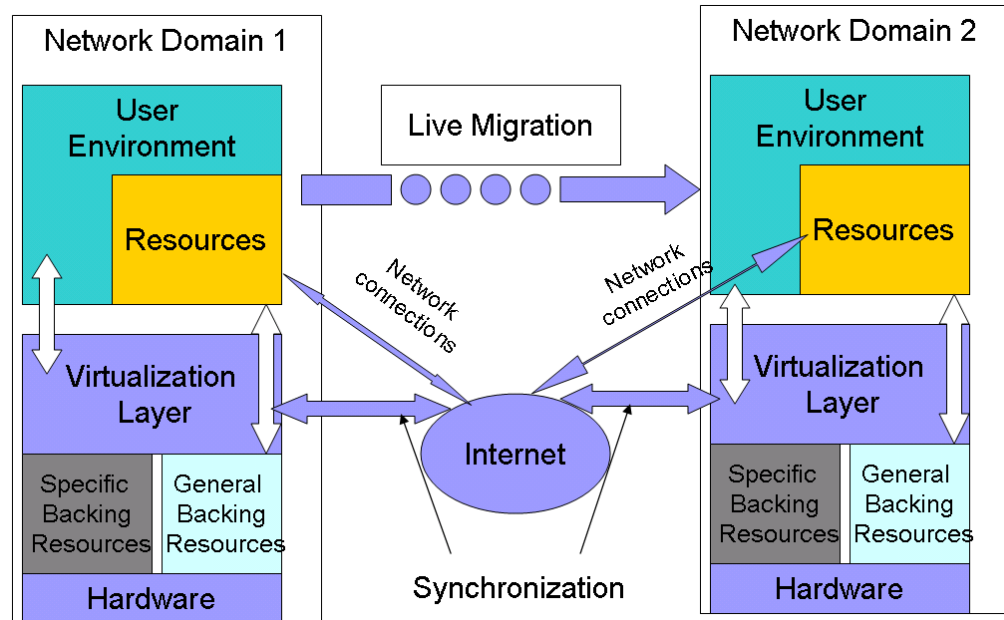


Figure 3.1: Architecture for live migration of user environments

For a system level virtual machine such as *Xen*, the resources available to the user can be a network interface card or a block device. However, a higher level virtual machine such as JVM, that runs entirely in userspace cannot work at kernel level and does not have full control of the network interface card or a block device. This gives approaches using JVM less control over the available resources but also a smaller amount of state that needs to be transferred over the network. In this thesis, it was found that extra level of control over the network and storage resources is beneficial in designing and implementing a live migration system. Moreover, one of the goals of the thesis was to provide the migration of user environments that have an identical interface to the existing operating systems which the users are already accustomed to.

Higher level virtual machines restrict the options available for user software (for instance a JVM approach means the user is only able to use software that compiles to JVM bytecode). Moreover, many system virtual machine implementations allow users to run whole commodity operating systems such as Linux or Windows and the whole range of applications supported by these operating systems at near native speed. A specific architecture using system level virtual machines is described in Section 3.3, leveraging their strengths, performance and ability to run a broad range of software.

Migration assumes that a user environment residing on a host is migrated to a different host, therefore the general backing resources that are available at the destination are made available to the user's capsule. The virtualization layer facilitates the synchronization between the state of these resources so that they are in a consistent state throughout and after migration to the new host.

One of the general resources available to the user, if the migration destination belongs to a new network domain, is likely to be a networking capability. Therefore, the virtualization layer makes it possible for the user's networking capability to be restarted and used inside the new network domain. This implies a setup stage in the new network domain as well as insuring that all network connections are resumed with minimum possible downtime.

This architecture is appropriate for the goal of live migration of user environments because it decouples the resources used by the user environment from the actual hardware resources, by using a virtualization layer. Moreover, it allows the virtualization layer to control the migration process and to easily reconnect the resources inside the user capsule to the specific backing resources present on the migration destination host. Another advantage from previous work is that live migration is performed instead of the stop/resume approach which also facilitates the survivability of network connections during migration.

3.3 Specific Architecture

This section describes the specific architecture for live migration of user environments in which the user environment is packed as a system virtual machine. This means that it offers a high degree of generality because it does not restrict user software. However, it does this at a cost. The user capsule is the whole operating system and the state can contain several GB of memory and persistent storage. Replication of virtual resources such as network connections and virtual block devices are specific problems pertaining to live migrating system virtual machines, and come at the cost of copying the state over the network.

This section describes the architecture for a solution for live migration of user environments using system virtual machines such as *Xen* or *VMWare*. It describes the building blocks that the specific architecture relies upon. Subsection 3.3.1 addresses the

live migration of persistent storage. Subsection 3.3.2 discusses ways to reduce the impact of the migration on the network traffic and subsection 3.3.3 describes how to migrate network connections for the proposed specific architecture. Subsection 3.3.4 concludes the section with further discussion.

3.3.1 Virtual Block Device Live Migration

Migrating the underlying persistent storage of a guest virtual machine can only be done from outside the guest because my design choice is that the guest operating system is not aware of the migration. This can be a misleading claim. Obviously the user has to have a way to trigger migration. However this is done by pressing a default combination of keys that switches focus to the host operating system. The user then triggers the migration process from the host operating system. Therefore, the migration is transparent to the guest operating system in the sense that the guest operating system does not have to be modified to allow migration. The virtualization layer takes care of the whole migration process including automatically synchronizing storage and memory state between the source and the destination. This fits very well with the hosted migration approach in which the privileged virtual machine takes care of pausing the guest virtual machine at the origin, copying the state to the remote destination and resuming it at the destination. Of course, there are situations in which the same resources are not available at the destination, such as a display or sound card. In this case, the user capsule will simply observe that the hardware resource is no longer available. It is beyond the proposed scope of this work to treat such error scenarios. A possible solution is to silently disconnect the device when resuming at the destination. Another solution is to simply check compatibility before migration starts and to ask the user if she wants to proceed.

I refer to the live migration described in Section 2.5 as regular live migration [28]. It refers to migration of the memory and CPU state. Live migration of persistent storage implies coordinating the regular live migration process with the replication of persistent storage. The privileged domain has access to the block-level write and read requests made by the guest domain. A replication layer between the guest virtual machine and the actual physical block device ensures that write requests are replicated to remote machines, thus performing efficient synchronization at block level.

Placing the logic at the block level means losing some of the upper level semantics from the file system. Therefore, synchronization will not be very fine grained. For instance,

writing just one byte of data will result in a few write requests to various blocks which will need to be mirrored entirely. The overhead is necessary for a general solution that does not depend on the file system or operating system being used in the guest. Moreover, some disk blocks are likely to be overwritten often, making it unnecessary to copy them to the destination very often. This is because overwritten blocks are expected to change again before the final step of the iterative live migration process and it is more efficient to copy them in the final steps of migration.

The buffer cache in the guest machine will be migrated along with the memory therefore I do not need to explicitly handle the buffer cache in the guest domain. This is done by the hosted migration daemon.

A separate connection between the two endpoints is maintained while the two virtual block devices on the source and destination are connected for synchronization. The establishment of this connection is independent of starting the process of live migration which can start at a later point. The *delta* between the persistent storage at the source and the destination depends on how many bytes and files have been modified since the last synchronization between the source and destination. It is usually the case that this delta is more than the amount of memory that needs to be synchronized. Therefore, synchronizing persistent storage typically can start at an earlier point in time since this is a lengthier process.

Typically, office workloads do not sustain a large disk write throughput [77] so they are not likely to slowdown the migration process considerably. Moreover, if a guest virtual machine dirties too many pages during the last stages of the live migration process, it is *stunned* [28] and limited to a small number of writes in order to make migration possible. On the other hand, it may seem that each block of the virtual block device has to be transferred over the same network at least once, and some blocks more than once. For large drives, this might take a very long time, much more than the usual commute. The remaining time can be estimated assuming a stable network connection will exist between the two endpoints for the duration of the transfer. However, this is unlikely to happen for all network connections and guarantees regarding the success of the transfer cannot be made.

Other approaches have been used in similar projects such as using a portable hard drive with the most up to date snapshot of the persistent storage state, before the user started the commute. Another possibility is to assume a certain degree of block-level similarity

between guest virtual machines which is especially true considering most virtual appliances in fact started from standard templates that the user extends on her own. Thus, most disk blocks and memory pages containing operating system software, applications and shared libraries will already be available from other users with similar guest capsule installations. This is an advantage if these locations are *closer*, in terms of the network speed, to the destination.

I leverage existing approaches to achieve a similar result and concentrate on the problem of achieving the smallest downtime possible during the process of live migration. This is essential to ensure a good user experience. Another goal is to ensure that the network connections survive the live migration process even when the migration happens across subnets.

There are three quality related aspects that define live migration: downtime due to the last step of the live migration, impact of the memory and persistent storage copy phase on the throughput of existing network connections and the responsiveness of user programs during migration.

The downtime due to the last step of the migration can be carefully controlled in the case when user processes are write intensive. This can be done by throttling the guest VM to a manageable page dirtying rate so that the guest can safely resume at the destination. The amount of throttling is easily configurable. The question still remains, however, if it is possible to perform live migration without impacting user experience for all types of broadband links, and all user workloads.

3.3.2 Reducing the Impact of the Migration on Network Traffic

Live migration implies large network transfers which will impact the response time and throughput of existing network connections from the guest virtual machine. It is likely that these long duration transfers will exhibit a long term impact on the user experience. One way to optimize this aspect is to employ a form of prefetching that allows disk and memory blocks that are unlikely to change very soon to be pro-actively sent to the destination.

For instance, a user might be using three locations to work on a regular basis such as two places at work and one at home. The work done while she is at home is likely to change

some disk blocks and memory pages. If these are not overwritten within a very short time, they are likely to remain unchanged until next time the user will commute to work. Prefetching these blocks to one of the work locations and then synchronizing the two work locations is a good optimization. However, this is likely to create unnecessary traffic if the blocks will eventually be changed at the home location. In order to minimize the impact of prefetching, live migration of user environments can use various background transfer services such as TCP-NICE [83], TCP-LP [52] or BaTS [50].

These low priority transfer services effectively modify the priority of a TCP connection with respect to normal TCP connections. TCP-Nice and TCP-LP run on the sender side of a TCP connection and use delay as an indication of competition. They modify TCP's congestion control algorithm and decrease the congestion window in the presence of competition. This allows these low priority services to stay in a state where they only take up the residual capacity of the bottleneck link, minimizing the impact on the throughput of the normal TCP flows. BaTS is a receiver side alternative to TCP-Nice and TCP-LP that allows the receiver to detect competition and to adapt the TCP receiver window to only take up the residual capacity of the bottleneck link.

The advantage of using one of these techniques is that it automatically relinquishes bandwidth to normal priority TCP connections from within the guest virtual machine. There is no need to manually set traffic shaping rules in the kernel or to employ a fixed application level throttling rate. It only uses leftover capacity from the user connections to pro-actively fetch blocks that are out of sync with the remote replicas. Due to the low interference with normal traffic, this connection can be running at all times, bringing huge benefits in terms of number of blocks that need to be fetched when the migration process is triggered.

3.3.3 Migrating Network Connections across Subnets

Migrating an operating system as a virtual machine capsule means that the state of the network stack and associated state will be migrated as well. This is not a problem if the capsule resumes in the same subnetwork because it can resume the network connections easily and keep using the same IP address. A gratuitous ARP broadcast is sufficient to update the intermediate layer two devices such as network switches about the new location of the MAC address.

However, the capsule cannot use the same IP address when migrating across subnets. Previous research has focused on using IP tunnels to allow the capsule to keep the same IP address during and after migration. However, tunnels are sensitive to network outages and may need to be restarted manually.

The solution for live migration across subnets must allow hosts to perform network handover and to maintain the same network identifier even though migration occurred to another subnet. Such a solution has to have no backward dependencies to the home network. Thus, IP tunnels were not used in order to improve the ability of the proposed solution to cope better with mobility. Instead a mobility protocol that can support the migration to another network without tunneling the traffic to the other home network was required. Among the protocols that can be used are the Host Identity Protocol (HIP) and Mobile IPv6. The Host Identity Protocol (HIP) allows the user to migrate across subnets and maintain connectivity at all times. Note that even though the architecture is built around HIP, some other mobility protocol could have been chosen as well. There is no particular reason this thesis chose HIP as a protocol other than ease of implementation. Using HIP, an identity is associated with the guest virtual machine that is bound to the IP specific to the current subnet. During the last step of the migration HIP is used to perform the handover process and resume connectivity at the destination. HIP does not need to be modified in any way to deal with virtual machine migration and is only aware that the virtual machine moved to another subnet.

In order to make HIP work seamlessly for the scenario proposed, a local IP needs to be provided to the migrated guest capsule. HIP binds the local IP with the host identifier and thus resumes network connectivity. Please note that given that HIP is positioned between the network layer and the IP layer, applications use the HIT instead of the current IP address. Therefore, migration is transparent to the application and even though the IP changes, the HIP protocol deals with this change. Applications use the same HIT even though the user capsule is migrated to a different subnetwork.

When the guest is ready to resume at the destination, it needs to obtain another IP address. There are several issues concerning how to obtain a local IP address. This is because the guest capsule is not aware of the migration and should not be configured manually with an IP address. The IP address would depend on the host's network configuration. Given the mobility requirement, the guest is likely to be configured to obtain an IP address automatically via DHCP. This assumption is most likely to be correct for most desktop configurations.

The approach used in this solution is to cause the guest capsule to re-acquire a new IP address via DHCP as soon as it starts resuming on the destination host. The back-end network driver running on the host automatically triggers resume and hotplug events to the guest network interface card driver. Thus, the guest network interface card is caused to send a DHCP request and re-acquire a new IP address. The HIP protocol intercepts this change of the underlying IP address and initiates the network handover process. One can envision that this approach can be further optimized by allowing the host to obtain or reserve an IP address via DHCP, on behalf of the migrating guest capsule before the migration process is actually finished and bind this address to the guest once this is ready to resume. This would prevent delays caused by the DHCP protocol, lost DHCP packets or network congestion. However, this is normally not necessary for the most frequent user scenario. Commonly, in a virtual machine environment, the host (the privileged domain) acts as a DHCP server for the guest capsules and the DHCP request is likely to be served locally to the virtual network interface, without incurring further delays.

3.3.4 Discussion

This chapter described the proposed architecture for live migration of user environments including persistent storage and network connections over wide area links. For this, user environments have to be packed as virtual machines which means that users can access the same environment on different hardware machines.

The architecture provides a way to perform the live migration of persistent storage by synchronizing the backing virtual resources that provide the storage and integrating it with the existing methods to perform the live migration of the memory and CPU context. Moreover, it provides a solution to perform the live migration of network connections across subnets by allowing the user environment to reconnect the underlying virtual networking resources and to make them available to the Host Identity Protocol stack which can then resume network connectivity with a small downtime and without having to restart any of the network connections. The main disadvantage of the proposed architecture is that it relies on the network to transfer the memory and persistent state. The downtime exhibited by the user environment as well as the user experience are directly dependent on the network speed and the rate at which users perform updates on the memory and persistent state of their environment.

It is unclear if the amount of network traffic generated by the migration could be

minimized if other choices would have been made, such as using a high level virtual machine approach. The amount of network traffic can be reduced if the user is migrating to a previous location and if a small number of bytes have to be synchronized but that may not always be true. However, the main motivation for this design choice was due to the rather futuristic scenario (which is similar to the one used in previous work) that assumes pervasively available hardware and high throughput network connections.

The choice for performing live migration of network connections using mobility protocols might have several problems such as dealing with firewalls and the lack of adoption for the particular mobility protocol used in the design. However, in the worst case scenario, the network handover fails and network connections have to be restarted which is the same as previous work.

Both the live migration of persistent storage and the migration of network connections are contributions of this thesis and have been implemented in a prototype described in the next chapter.

Chapter 4

Implementation

This chapter describes the initial proof of concept implementation for the architecture described in Section 3.3 of the previous chapter. The implementation is made out of several components that are plugged in at different levels within the architecture. In the implementation, the user capsule is a fully functional operating system that runs as a guest virtual machine. The implementation makes use of the already available live migration of the memory and CPU state implemented in Xen [28].

This chapter is structured as follows. Section 4.1 describes the current approaches to migrate persistent storage inside Xen. It also shows the performance and disadvantages of such an approach and motivates the decision taken in the actual implementation that is described in Section 4.2. Section 4.3 describes the implementation of network connection migration and describes how the HIP protocol was integrated. Finally, Section 4.4 shows the way the evaluation is done, describes the limitations related to testing the prototype and describes the implementation of a specific benchmark specifically designed for testing the implementation.

The purpose of this chapter is to show that a real system prototype was implemented, to highlight how the challenges involved were addressed and to show a real implementation of the architecture described in the previous chapter.

4.1 Persistent Storage Migration

Currently Xen supports physical partitions and file-backed virtual block devices and all the Storage Area Network (SAN) solutions such as iSCSI [57], ATA Over Ethernet (AoE) [27] and Fiber Channel [47]. However, block device live migration is not supported by Xen yet. Therefore, currently, in Xen live migration uses a SAN solution (Figure 4.1).

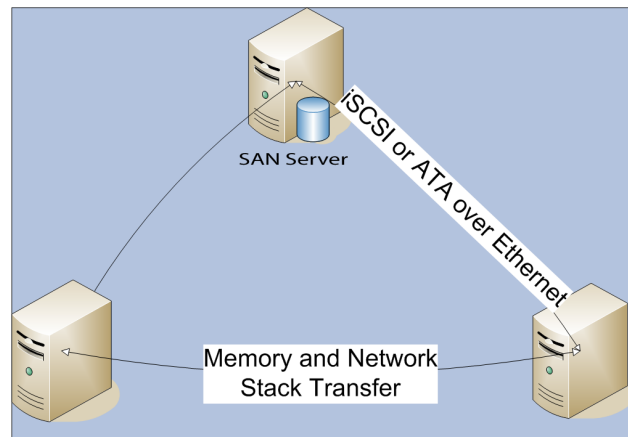


Figure 4.1: Currently, live migration of virtual block devices can be done using SAN.

SAN storage is easy to migrate because Xen already migrates network connections with an almost unnoticeable downtime in the same LAN. This approach would only work well in a cluster and is usually a very expensive option. It relies on remote storage and thus the disk performance is dependent on the network characteristics. For good performance, Gigabit links are required and migration is virtually impossible outside the subnet. Even with the use of tunnels, a solution using remote disks over WAN will suffer severe performance degradation.

Experiments in [28] analyze the performance of live migration of a heavily loaded virtual machine across two nodes in a data center using an iSCSI SAN. The nodes were interconnected with Gigabit Ethernet links. I have reproduced similar experiments (Chapter 5) at a smaller scale, using regular PCs under moderate load, interconnected via a 100Mbps link. The goal of this experiment was to test how Xen's live migration works for a lightly loaded environment using much slower links. An expected increase of up to a few seconds in the perceived network downtime was noticed compared to the original experiments in [28].

However, for the purpose of live migrating VMs across subnets, SAN is not a viable solution because it creates a backward dependency to the network where the storage is

located and is therefore susceptible to network outages all the time, not only during the live migration. Therefore, the alternative is to use local drives backed by either LVM [6] partitions or files in userspace. They provide a higher level of abstraction over low level storage block devices and allow easier administration. LVM allows partitions to be created, resized and deleted without actually writing to the disk's partition table. LVM easily supports the use of many physical disks drives, adding and removing disk drives in a way that is transparent to the user.

LVM is therefore a very good solution to manage many virtual disks. However, it still requires the intervention of the system administrator whenever operations like creating and resizing a user's disk are necessary. File-backed VBDs are a better way of managing users' disks. They can be easily moved, resized and transferred across the network. This choice does not affect the migration strategy in any way, the two approaches differ only from the point of view of the administration. The implementation in this thesis uses LVM to manage persistent storage. The next section details how LVM is used in implementing the proof of concept for the thesis.

4.2 Live Migration of Virtual Block Devices

I have developed an initial proof of concept solution to support live migration of virtual block devices based on Xen, DRBD (described in Section 2.4.1) and LVM. DRBD was used on top of LVM, that is a DRBD block device was created on top of an LVM partition.

DRBD was used just for synchronizing the disk blocks on the source and the destination. For performing correctly live migration of block devices, DRBD had to be configured in a specific way. Each DRBD device can have two attributes: Primary or Secondary. In Primary mode the device is readable and writable and all the changes are automatically mirrored to the peer device. A write to a Primary device will not show up as completed to the upper layers until the write is mirrored on the peer's side. This ensures consistent replication. In previous versions of DRBD, when one device is in Primary mode, the other one has to be in Secondary mode. This means the secondary device cannot be accessed, mounted or read.

This made live migration difficult, because the device had to be accessed by the hotplug

scripts on the destination node before migration was complete. Thus, I chose to use the latest version of DRBD 8.0 which allows both devices to be configured in Primary mode.

Using both devices in Primary state works when the underlying file system is a cluster file system such as OCFS2 or GFS. Using file systems such as ReiserFS or Ext3 and performing writes from more than one host leads to corrupting the file system.

However, any normal file system such as ReiserFS can be used because the Xen live migration mechanism ensures that the migrating machine is checkpointed on the source before being resumed on the destination, so no writes will occur at the same time on the source and the destination.

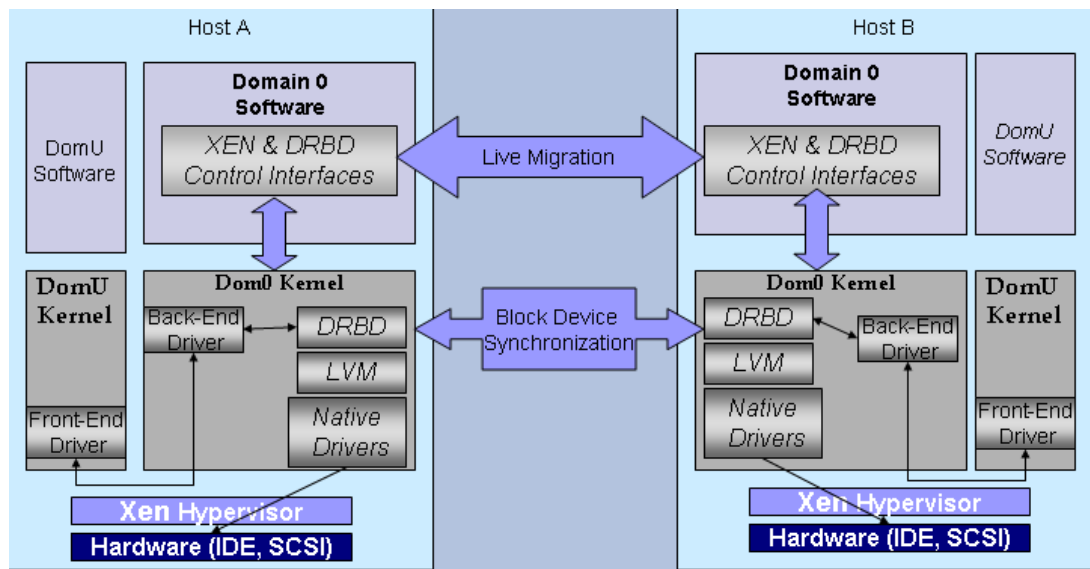


Figure 4.2: Live migration of virtual block devices.

The implementation is structured in the following way. Xen's privileged domain controls the live migration process through a control interface. It also runs a migration daemon that does an iterative copy of the memory to the destination ([28]). The migration daemon is the first component of the system and is already a part of Xen. The second component of the architecture of the system is the module that synchronizes the data between the DRBD based VBDs. I extended Xen's control interface to also give commands to the DRBD module which takes care of synchronizing the VBDs.

VBD migration was implemented in Python by extending Xen's external devices migration mechanism (Figure 4.2). The external device migration mechanism is already used for the Virtual Trusted Platform Module (vTPM) device. It can potentially be used for migrating other external devices as well, but in my current implementation there is

support just for migrating block devices.

The source host communicates with the destination using XML-RPC calls to prepare the destination host for the migration and give the appropriate commands to the DRBD module on the destination. All this logic is implemented in a daemon that is running independently from the migration daemon. This daemon orchestrates the migration process of both memory and persistent storage. The basic commands are to start and stop the synchronization between the two VBDs. The XML-RPC daemon also considers the possible error cases and can stop the migration process if there was an error while synchronizing VBDs.

Thus, using this mechanism, the source host can signal the destination to attach the DRBD device and to start the process of re-synchronization. Prior to this call, the drives are in disconnected mode and the primary device is marking the active regions where writes occur so that only dirty regions of the disk are resynchronized. During migration, disk reads are done locally but disk writes are done synchronously on both block devices.

During the experiments, it was noticed that even for an idle machine, in case there is a temporary communication error between the source and the destination, DRBD considers that a large amount of data is out of sync. For instance, in a test with an idle machine, after a short network outage between the source and the destination, 19MB of data out of a 1GB disk needed to be synchronized. The size of this data can be considered large for an otherwise idle machine. In order to support live migration over WAN, the focus is not on disk access speed but rather on minimizing the network bandwidth used by migration. Therefore, a more fine grained policy for marking the dirty disk chunks must be developed at the expense of a more complex synchronization algorithm.

Just after migration is initiated by the user, one more check is done to see if the block devices are already attached and syncing. If not, they are caused to do so. The next step is to start the iterative memory pre-copy phase that Xen performs during migration (Section 2.5).

After the last memory iteration, a check is done to see if the devices have completed syncing. If so, the checkpointed VM state is transferred to the destination host and is resumed there. All this logic is done by exchanging XML-RPC messages between the source and the destination and by inspecting the current state of the migration from Xen's control interface.

After migration is complete, the two drives can remain connected and maintain consistent replicas of each other or they can be optionally manually detached. This choice belongs to the user. This means that connections to the subnet where migration was originated are no longer required, thus breaking the residual dependencies to the originating site.

4.3 Migration of Network Connections

In a single switched LAN, a migrating VM keeps its IP and all the network stack. Migrating inside a single switched LAN, which is typical for a cluster, is done by issuing an unsolicited Address Resolution Protocol (ARP) reply from the migrating guest [28]. This broadcast contains the Media Access Control (MAC) address of the migrated virtual machine and updates the *layer two* network devices about the new location of the virtual machine.

Across subnets, live migration aims at maintaining all network connections without tunneling through the original host. The original host may be shut down or network connectivity with the host may be disrupted after migration. Moreover, handling a chain of migrations to distinct hosts would prove very complicated and inefficient to achieve using tunnels.

An open problem with migrating VMs outside the same subnet is maintaining network connectivity while the IP address changes. Solutions for creating virtual networks to be used with virtual machine migration have been developed [79] [43]. However, their approaches rely on tunnels to the original subnet and are not appropriate for scenarios such as migrating a user's VM from work to home, although they may prove sufficient for migration between computers within the same organization.

In order to solve this problem, the implementation described in this thesis uses the HIP protocol. Each guest capsule uses HIP to support mobility across subnets. After the migration of the persistent state and of the memory finished and the guest is suspended at the origin, the guest starts reconnecting the devices at the destination host. The implementation, shown in Figure 4.3, plugs into the reconnection procedure between the network interface back-end and the front end device drives in Xen.

Udev allows userspace to manage the */dev* tree in a dynamic way. It uses information

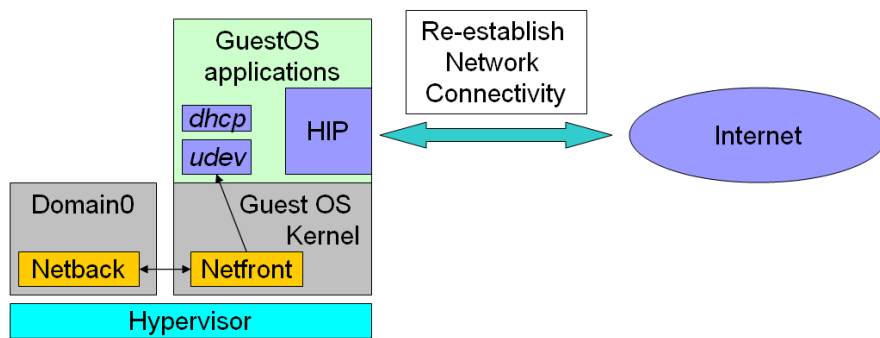


Figure 4.3: Migration of network connections using HIP.

exported from *sysfs* about the devices and receives information from */sbin/hotplug* when a device is added or removed. Thus, on successful reconnection between the back-end and the front-end, a hotplug event is sent to userspace causing the *udev* scripts to re-acquire a new local IP address via DHCP. This address will then be used by HIP to perform the handover and to resume all network activity. This approach is faster than causing the guest's network services to be restarted and effectively optimizes the handover by decreasing the network downtime caused by migration.

The implementation uses UDP encapsulation to bypass Network Address Translation (NAT) boxes. This is already part of the OpenHIP implementation for Linux. Special keep-alive messages are periodically sent to maintain NAT port bindings. Thus, a guest capsule with a private IP address can be live migrated from a network domain that uses a NAT to gain access to the Internet to another subnet where a NAT is also used, and network connections to the outside world are not interrupted.

During testing it was observed that live migration of network connections did not perform well even within a LAN. There was a reproducible downtime of 1-2 seconds until the network was able to send packets again. I have noticed that the ARP broadcast was sent by the guest netfront device but was dropped by *Domain 0* and never reached the network. Moreover, there was an additional delay before the network interface card was successfully restarted and allowed to send packets on the network.

This was in fact a software bug in Xen 3.0.3. After migration, the back-end and front-end need to be reconnected. The reconnection of the network front-end was not propagated fast enough along the network path to the bridge code in the privileged domain. This bug was reported to the Xen developers and was fixed by the Xen team by reintroducing a fake network carrier flag that enabled proper reconnection between the network back-end

and front-end instead of the *netif_carrier* flag which was causing delays. The ARP broadcast was then sent as soon as netfront and netback reconnected.

This bug also triggered an interesting discussion about the best solution to migrate network connections on a LAN. Under many conditions, ARP broadcasts can be lost. Moreover, the code in netfront was not checking if the ARP was dropped or not by lower layers. Therefore, a more reliable solution such as sending updates to all the entries in the ARP cache, in addition to the ARP broadcast might be required.

4.4 Testing

A method for testing the implementation's performance is needed. Replaying traces is a good method for benchmarking and stress testing. It has the advantage of being able to reproduce the exact set of operations defining a specific workload. Trace capture and replay tools can operate at either network packet, disk drive, system call or network file system levels. As they operate at a higher level, they lose their ability to reproduce the exact workload characteristics. Therefore, the most desirable method is to replay traces at file system level. Systems such as *Replayfs* [46] are capable of replaying traces from various file systems at VFS [72] level. ReplayFS would be a good choice to collect traces from a long period of time from users using various file systems and to replay them inside guest capsules. However, such traces would only simulate file system activity and will not capture memory access traces and network traces. Therefore, a system able to capture and replay all these traces would have to be built.

Alternatively, synthetic benchmarks can be used to stress test and to measure I/O performance during migration. However, no suitable benchmark that accurately simulates regular office work behavior was found, besides SYSmark [12], since most of the benchmarking tools aim at reproducing I/O-intensive server workloads. The SYSmark benchmark can provide good approximations of user workloads. However, it only runs on Windows and currently, the live migration prototype only supports Linux.

Thus, in order to study the performance of the live migration initial proof of concept, a simple benchmark was created to simulate real user workloads. The benchmark used in this thesis is a synthetic user workload simulator written in Java based on some patterns observed by several studies about file system workloads. It is multi-threaded and can read

and write random patterns from a set of files at random offsets. Some small set of the files are read and written to more than the rest. This choice is based on existing work that studies various user workload patterns and is explained in the rest of this section.

However, the benchmark is limited to generating random patterns. Real users are unlikely to be realistically modeled by random patterns but in the absence of real traces, the random patterns approach was considered instead. Due to these features, the results expected from running this workload generator are expected to be more likely to mimic the behavior of real users, compared to the existing synthetic benchmarks that simulate server workloads. The workload generator models an intensive read and write user workload, so it is expected that the results from testing with this workload generator will be a worst case scenario for most users. However, it is likely that some very intensive user workloads will not be simulated well enough.

In order to make it easier to compare the results of my work, I intend to study the performance of the prototype using the standard Iometer [3] benchmark as future work. One of the main reasons for using the workload generator instead of Iometer is that it provides an easy to extend platform for simulating complex user workloads that don't only perform I/O, but also perform computation between I/O bursts, perform memory intensive tasks, replay user activity traces, etc. In its current form, the benchmark is very similar to Iometer but Chapter 6 describes some future work about extending the benchmark.

As discussed in the background chapter (Section 2.4), an important aspect of storage replication is decreasing the amount of data that is transferred over the network and this can be minimized with an asynchronous replication protocol. In order to decrease the number of disk writes, the file system can delay flushing the data to the disk. Delaying block writes maximizes the chance that some of them will be overwritten multiple times. Thus, the blocks that are deleted do not need to be written to the disk at all. The study in [70] shows that the default 30 seconds time used to flush the buffers to the disk is too small. Most blocks live longer than this period and this period depends highly on the workload characteristics and the operating system and file system structure. For instance, in UNIX systems, most blocks die within an hour while in Windows NT many blocks live up to 1 day. However, most overwrites have a high degree of locality and could be accommodated by a relatively small write buffer. Moreover, the same study shows that most files have either write-mostly or read-mostly access patterns.

Block lifetime is known to be dependent on the type of workload. Some desktop

workloads have bi-modal behavior, some blocks live for less than 1 second and others live longer than a day. Most of the blocks that live longer come from large software installations. Other workloads have a gradual decrease for block lifetime. A large percentage of the blocks that die are in fact overwritten and there is a high degree of locality in the overwrites. Most of the overwrites are within less than 2 – 3% of all the files for all the workloads studied in [70].

This shows an important result for replicating user workloads. Firstly, block-level writes do not need to be replicated as soon as they are committed to the primary storage device. Delaying them would only decrease the amount of network traffic required for synchronization. Secondly, given the locality of disk writes, it is straightforward to identify blocks that are likely to be overwritten. If the blocks are from files that are highly overwritten, they are very likely to be overwritten again.

Therefore, the following policy for selecting blocks for replication is likely to decrease the network traffic overhead. Blocks are replicated over the network based on priority. If blocks map to the same file that has been overwritten often in the last time frame, they are assigned a lower priority. Thus, blocks that are likely to have a small lifetime will be at the back of the priority queue. This simple policy is likely to decrease the redundancy of mirrored blocks within the network traffic.

Another aspect relevant to tuning the Java benchmark used in this thesis is to replicate the typical read and write ratio. The proportion of read and write traffic in the various workloads analyzed in [70] shows that most of the requests are reads. However, some traces exhibit more write requests than others even within the same workload set.

One of the reasons it is hard to create a realistic benchmark that emulates user behavior is because of the increased use of memory mapped files. Several workloads studied in [70] show that memory mapped files are an increasingly popular way to access files. More processes access files via memory mapping than through file reads and writes as a result of process calls to *mmap*, *munmap*, *fork* and *exit*. This is especially the case with shared libraries. Though there are not too many files that are memory mapped, there are more processes accessing them through the specific read and writes interfaces. Memory mapped file access patterns are harder to monitor. Due to this aspect, it is harder to trace a realistic workload and thus produce a benchmark that accurately matches the behavior of real users.

4.5 Discussion

This chapter presented the implementation details of the architecture for live migration of user environments across wide area networks. The implementation deals with the two main problems of migrating persistent storage and migration of the network connections when the user capsule moves to a different subnet.

VBD migration leverages DRBD to perform block device synchronization and integrates seamlessly with Xen. This migration is transparent to the user capsule. While this was an easy to implement approach, it might not be the best choice, mainly because of two limitations of DRBD. The first limitation is that in order to use DRBD, a synchronous replication protocol had to be used and as discussed in Section 2.4 this can slow down write bursts. The second limitation is that DRBD assumes a LAN environment where network disconnections are rare. In the wide area, communication errors are more likely. After recovering from a network error, DRBD conservatively resynchronizes a larger part of the disk than needed.. This limitation could be addressed by implementing a specific block device synchronization optimized for the scenario specific to this thesis, but is left for future work.

Network connection migration uses a userspace version of the HIP protocol which will not have the same performance as a kernel space one. This could be a future work optimization. It is likely that other optimizations can be performed to improve the handover time.

Chapter 5

Performance Evaluation

This chapter describes the performance evaluation of the proof of concept implementation described in the previous chapter. In order to test the performance, two major types of tests are performed, each measuring the various impacts that live migration has on the network and disk read and write performance. The focus is to measure the downtime during migration, if network connections survive the migration and if the slowdown due to disk performance is acceptable. These results should clarify if the proposed architecture for live migration of user environments is a feasible way to deal with user mobility.

Section 5.1 describes the experimental setup and the software used for testing. Section 5.2 describes the network performance tests and results and Section 5.3 describes the disk I/O performance.

5.1 Experimental Setup

Figure 5.1 describes the experimental testbed. A guest capsule is migrated between two hosts in the same 100Mbps Ethernet LAN. The guest capsule maintains active connections to various other computers within the same LAN.

In order to simulate three separate subnetworks within the same LAN, the gateway server was set up with two real network interface cards and aliases on one of the interface cards

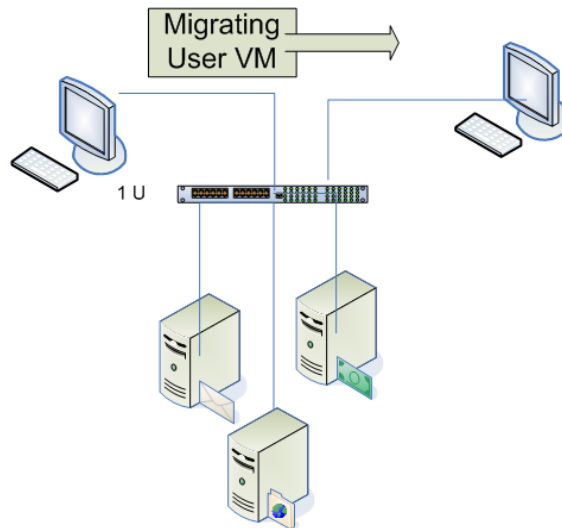


Figure 5.1: Performance Evaluation Setup

to emulate an additional network card. All the computers in the test were connected using the same network switch. Therefore, broadcasts arrive at all the network interfaces of the gateway. This brought up a complication related to identifying the interface on which DHCP broadcasts were sent and was solved by specially configuring the DHCP server. However, specific configuration of the DHCP server is only needed in the test environment used and would not be required in a real world scenario.

Xen 3.0.3 and the latest DRBD 8.0 were configured on two identical Pentium 4 2.8 Ghz machines with 1 GB of RAM connected using 100 Mbps Ethernet. The two machines and the VM were running Debian Linux. The migrating machine was configured with 256 MB of RAM and a DRBD block device on top of a LVM device.

5.2 Network Performance

Netem, a network emulator is used to test the storage migration solutions over various link types. Netem is used to emulate wide area networks properties such as variable delay, loss, duplication and reordering. It is part of the Linux 2.6 kernel and together with the Linux traffic shaper *tc* it can be used to emulate various bandwidths too. The following script is an example of how to emulate a network with a bandwidth of 8 Mbps and a delay normally distributed between 20ms and 40ms. The filter works on the outbound traffic only and in this case it only applies for the IP address *130.209.253.109*.

```
#tc qdisc add dev eth0 root handle 1: prio
#tc qdisc add dev eth0 parent 1:3 handle 30:\
    netem delay 40ms 20ms distribution normal
#tc qdisc change dev eth0 parent 30:1 tbf\
    rate 8mbit latency 150ms burst 40k
#tc filter add dev eth0 protocol ip parent\
    1:0 prio 3 u32 match ip dst\
    130.209.253.109/32 flowid 10:3
```

The experiments in this chapter use a 100Mbps Ethernet LAN setup which is significantly slower than Gigabit Ethernet links that were used in the experiments in [28]. Therefore, this network setup is closer to the performance of a wide area network link. The wide area network characteristics can be better modeled by emulating network characteristics using *netem* as in the above example. Live migration is also successful in cases where *netem* is used, however, the results are not described in this chapter and a test in different network conditions is subject to future work.

Iperf is used for measuring the downtime of the migrating machine. Iperf shows the downtime as seen by the outside world during the live migration. Iperf is in fact a bandwidth measurement tool that was used in these experiments to simulate a file download. Iperf is used because it is capable of computing fine grained bandwidth measurements and therefore measure the downtime. Opposed to [28], it is assumed that the migrating machine is mostly acting as a network client, not as a server. Therefore Iperf is used in client mode on the migrating machine with the option of computing statistics every 0.5 seconds.

Figure 5.2 shows the throughput of a single TCP connection to another server on the LAN during live migration using the unmodified Xen live migration. The throughput is measured by Iperf while running in client mode inside the guest capsule. In this experiment, the migration command starts at second 3. The additional traffic generated by Xen's live migration memory copy causes a drop in the test connection's throughput. Around second 60 there is an almost 3 seconds downtime caused by the final memory copy phase and the time to resume the user capsule on the destination. After second 62, the test network connection's throughput is restored to normal values.

Even though Iperf is not necessarily representative for a typical user workload, such a situation can represent a real user workload when a user is downloading a file from a

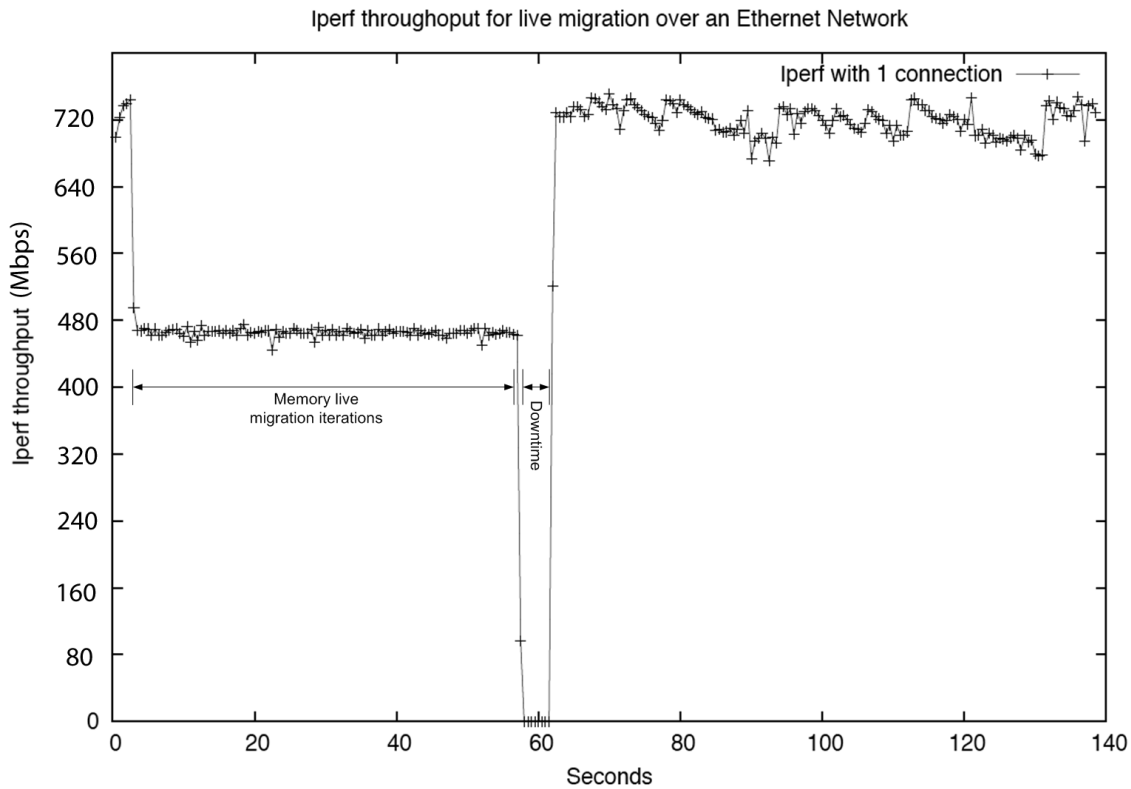


Figure 5.2: Live Migration across 100 Mbps Ethernet of a VM running an Iperf client using the unmodified Xen live migration (no storage migration was performed). The storage was mounted from a remote server using the ATA over Ethernet protocol. Migration occurs in the same subnetwork. No additional network delays were introduced.

webserver and this traffic saturates all the user's bandwidth. Moreover, a client scenario was chosen because users are typically behaving as network clients, as opposed to the scenario of migrating servers in [28]. Iperf however is unlikely to represent a typical user workload. The reverse situation when the migrating user capsule is a server is also supported but the performance was not measured for this case.

In this experiment the storage was mounted from a remote server using the ATA over Ethernet protocol. Several interactive *ssh* sessions between the guest capsule and other machines were maintained during the migration. Besides the network load, the guest machine was otherwise idle, no additional I/O load was performed.

HIP was not used for this experiment because migration took place within the same subnet. Therefore, the guest VM keeps the same IP address and only needs to update layer 2 network devices about the new location using an ARP broadcast.

The Iperf TCP connection incurred a downtime of approximately 3 seconds. Measurements were done from outside of the virtual machine. This may prove to be unacceptable to certain delay sensitive applications such as audio streaming and is not at all justifiable for the case of a 100Mbps Ethernet link.

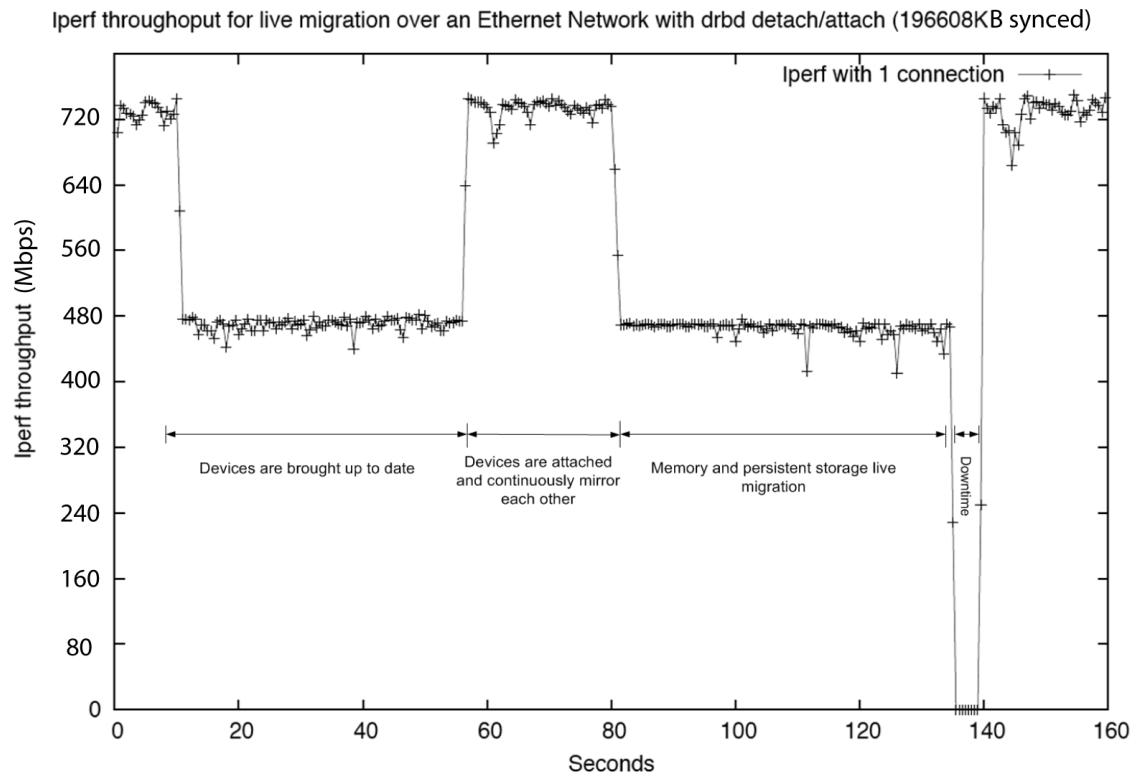


Figure 5.3: Live Migration across 100 Mbps Ethernet of a VM running Iperf in client mode. The experiment is done using the VBD migration prototype. No additional network delays were introduced.

The experiment depicted in Figure 5.3 integrates our live migration virtual block devices prototype in the same experimental settings. In this experiment, around second 9, the migration command was issued for the virtual block devices. The virtual block device replica at the destination was not up to date and had to be synchronized. For this purpose, *196608 KBytes* were updated. In this experiment, VBD migration ends at second 57, before the memory migration begins. The memory migration starts after second 80. Next, at second 135, the last phase of the migration starts and it ends after 3 seconds when the user capsule resumes on the destination. The source of this unexpected downtime was found to be a bug in Xen described in Section 4.3. The same downtime occurs with and without VBD migration therefore the results are comparable to the previous experiment, except that more data had to be sent over the network in order to synchronize persistent storage. The experiment was repeated (not shown here) after the bug was fixed and showed a downtime of 0.9 seconds instead. An even smaller downtime can be seen for

instance in the experiment discussed below, when network connection migration using HIP is enabled (Figure 5.4). This downtime should allow survivability of most user connections.

A substantial decrease in the throughput measured by Iperf during migration can be noticed. As expected, network activity is impacted during migration. The network connections used for migration compete fairly with the user connections for the capacity of the bottleneck link. However, this experiment shows the worst case scenario in which the user is making use of the whole available bandwidth. It is expected that most network activity of a regular user will be characterized by small bursts of traffic and mostly idle connections [33]. Therefore, the spare network capacity can be used by migration without a significant degradation of user experience.

Experiments were performed to measure the downtime and impact on network connections when migration is done across two different subnetworks. Block device migration was also performed. The guest capsule is migrated from behind a NAT to another using HIP. It obtains private addresses using DHCP and uses UDP encapsulation for a TCP connection to an outside HIP enabled server. The throughput measured by Iperf on the server is shown in Figure 5.4.

Migration begins at second 30 and the final copy phase occurs at second 58. The throughput was sampled by Iperf every second and it can be seen that it only exhibits a small drop during this phase but quickly recovers to the initial value. However, there is no actual downtime when no data is transmitted. This indeed is the goal of the implementation, which is to have an unnoticeable network downtime, therefore this result is very promising.

The same figure shows an interesting behavior of the OpenHIP implementation. TCP throughput is slowly degrading over time. The degrading performance occurs even without performing migration, therefore it is most likely a consequence of a performance problem with OpenHIP. The overhead might be coming from the fact that OpenHIP is implemented in userspace instead of kernel space. This is clearly a limitation of the tools used in implementation, not of the solution proposed, and are expected to disappear after integrating newer versions of OpenHIP. I reported this performance bug and did extensive work for several weeks to debug it with the help of the OpenHIP developers but I was unsuccessful. Given this performance problem, it is not possible to draw strong conclusions about the feasibility of the approach until fixing the bug. It is expected that

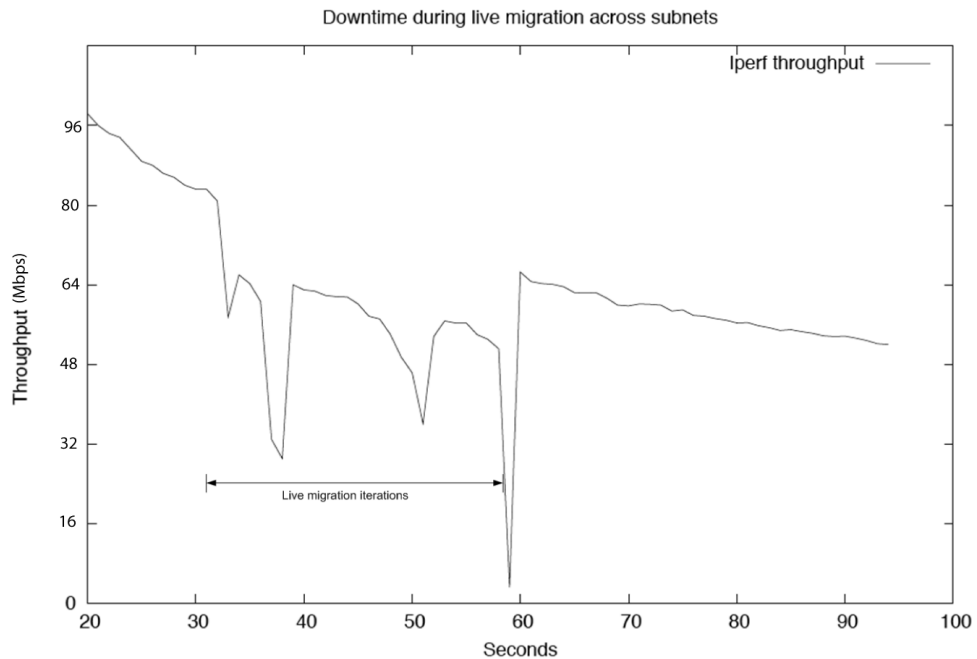


Figure 5.4: TCP throughput measured on the server during migration across subnetworks using HIP. VBD migration is also performed. No additional network delays were introduced.

this is just an implementation problem and will disappear with newer versions of OpenHIP, however, this is left for future work.

Figure 5.5 shows the throughput measured by Iperf inside the migrating guest capsule during the same experiment as above. The throughput shows a 4 seconds downtime which seems to disagree with the lack of downtime measured on the server side. A possible cause is buffering inside the virtual machine. Analyzing this discrepancy is the subject of future work. This bug is very subtle and due to lack of time, I was not able to discover the cause of this discrepancy and present it in this thesis. I did extensive work to debug this bug without finding out the cause. This is due to the many layers that needed to be profiled to obtain the exact cause. Even though this bug is not likely to impact highly user experience (the downtime is still in the order of a few seconds), it also prevents this thesis from drawing strong conclusions about the downtime of live migration. Previous studies have measured the downtime by looking at lower level traces on the host, using *tcpdump*¹. This work is more concerned with the impact of live migration on the network activity as seen in userspace, inside the guest capsule,

¹<http://tcpdump.org>

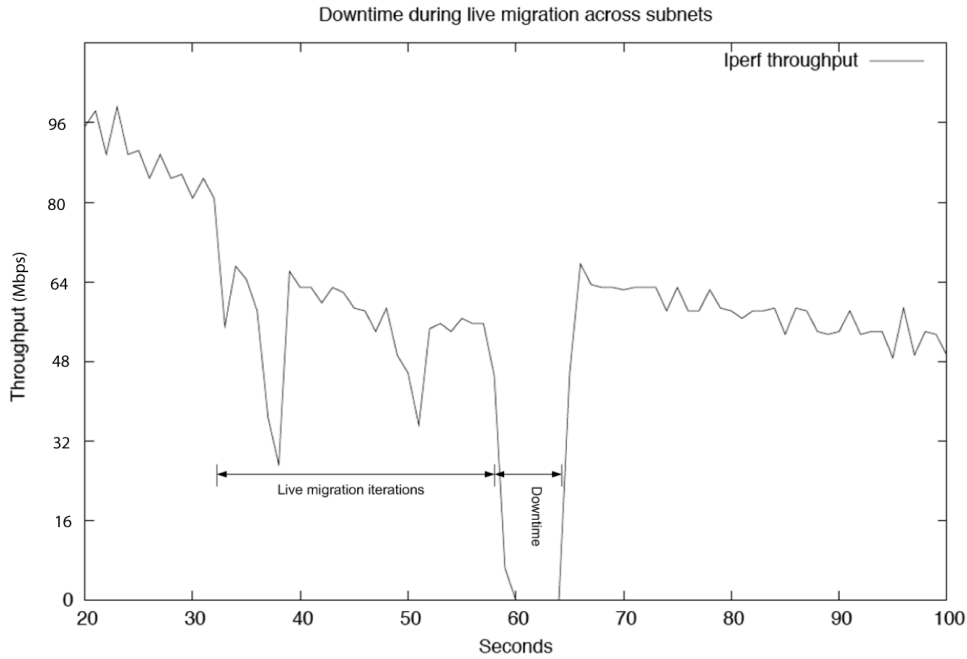


Figure 5.5: TCP throughput measured on the client during migration across subnetworks using HIP. VBD migration is also performed. No additional network delays were introduced.

therefore, this downtime needs to be minimized by future improvements to the prototype.

By using *netem*, several network setups with lower bandwidth and higher delay were emulated. In all cases, the downtime incurred by network handover was the order of a few seconds and allowed all network connections to resume seamlessly, however the results are not presented here. A more thorough performance evaluation of network handover using different network setups is the subject of future work as outlined in Chapter 6.

The experiments in this section were performed to simulate the impact of the live migration of user environments on a network benchmark that measured the throughput and perceived downtime of one TCP connection. This was meant to simulate a worst case scenario, and assumes that users would not typically use such high throughput connections. Therefore, the migration process will have less influence on the network performance for those users. The experiments show that even for such a high throughput connection, the impact of live migration could be acceptable for most users. The same experiment, repeated while other low throughput connections were generated from the user capsule showed that the downtime is the same for all connections. This result is

expected because the downtime is generated by the time required to reconnect the network interface after migration and does not depend on the number of TCP connections.

5.3 Disk I/O Performance

The experiments in Figures 5.6 and 5.7 show the disk *I/O* performance during migration. The measurements were done using the Bonnie++ [1] benchmark. Bonnie++ is a benchmark for servers and it was used here as well because of the lack of benchmarks that simulate user workloads as described in the previous chapter, in Section 4.4. This section contains performance tests with the user workload benchmark developed in this thesis.

Bonnie++ is designed to perform a number of simple tests of hard drive and file system performance. It measures the throughput when reading, writing and rewriting a single char, a whole block, as well as the percentage of the CPU used. By default it does not bypass the write buffer, that is it does not *fsync()* after each write operation.

Bonnie is in fact a synthetic benchmark, the first phase simulates a database application and the second phase simulates creating, reading, writing and deleting many files, similar to a proxy server. The benchmark is therefore more suited for emulating a server workload so this represents a worst case scenario for the type of workload addressed.

Three situations have been analyzed separately: *Standalone*, *Connected* and *During Migration*. In the *Standalone* configuration, the system performs no replication, therefore this is the baseline and this test shows the maximum performance achievable since there is no additional overhead or write throttling involved. In the *Connected* test, write requests are done synchronously over the network. A write request does not report as completed on the machine where it was issued until it is completely replicated and acknowledged by the remote node. Finally, in the *During Migration* test a whole system live migration is performed. Block device write requests are mirrored synchronously and the memory is iteratively copied over the same link.

Figure 5.6 shows the performance of sequential input at char and block level. It can be noticed that the overhead of migration is small but not zero. This is partially because an additional software layer has been inserted between the virtual block device and the

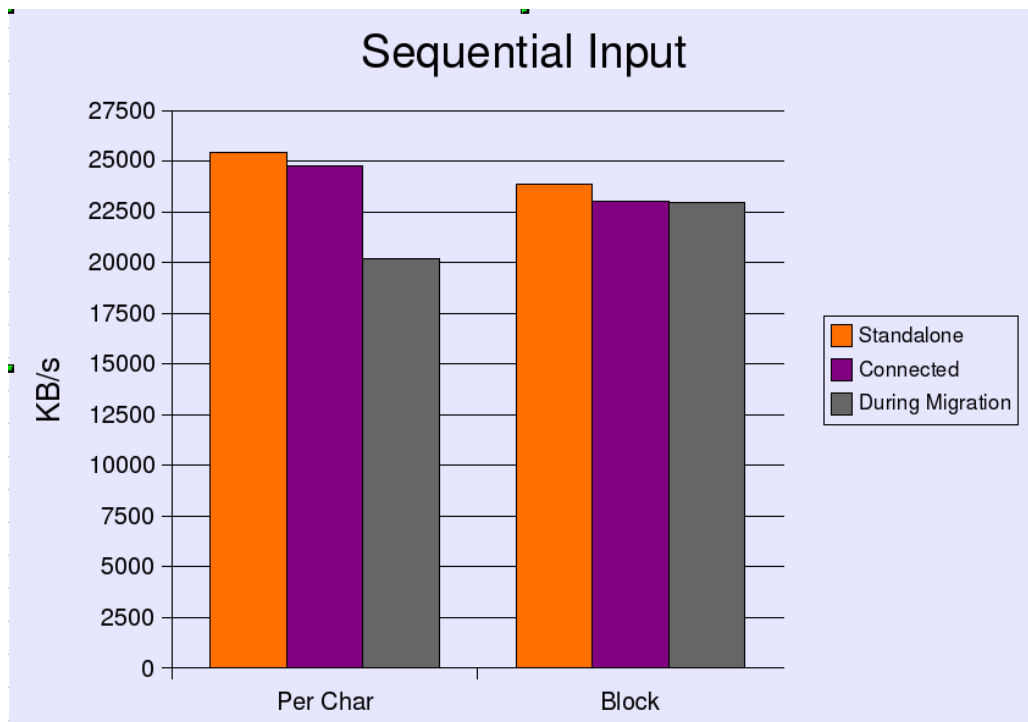


Figure 5.6: I/O performance of sequential input during virtual block device migration over a 100Mbps link measured with the Bonnie++ benchmark.

actual physical device and this layer is on a performance critical path. The *During Migration* test exhibits slightly less read throughput than the *Connected* one. This is attributed to the additional complexity of migrating the memory as well as to the small downtime incurred by migration. Since there is no replication for reads, none of this overhead is due to the disk replication.

Write access performance (Figure 5.7) during migration depends on the network bandwidth available. This is motivated by two factors. Firstly, write throttling during migration limits the amount of completed write requests. Secondly, in the *Connected* and *During Migration* tests, write requests are mirrored synchronously and are therefore limited by the network bandwidth available. In the 100 Mbps setup, a block write throughput of more than 12500 KBytes/s was measured for the *Connected* test and a block write throughput of over 5000 KBytes/s was measured for the *During Migration* test. This shows that in this setting, an adequate write throughput can be obtained during migration and that the solution makes adequate use of the available network bandwidth. The results are expected to vary greatly with network bandwidth and latency, however, currently the performance was measured only for this scenario.

I/O performance during live migration was also measured while running the benchmark I

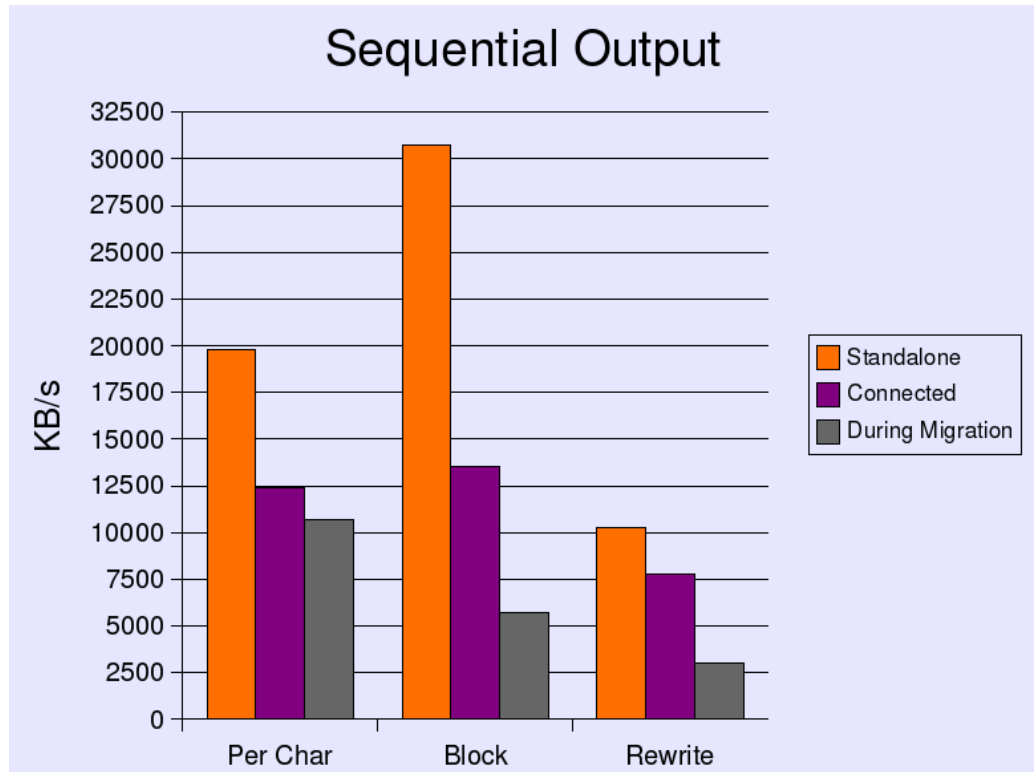


Figure 5.7: I/O performance of sequential output during virtual block device migration over a 100Mbps link measured with the Bonnie++ benchmark.

wrote in Java and presented in Section 4.4 of the previous chapter. Approximately 260 MBytes were written during this test. Two versions of the test were done. The first performs more intensive write operations interleaved by an equal number of read operations on the same files. The other test introduces a 50 ms sleep time between writes, to simulate a less data intensive environment, more likely to fit with normal user activity. Even so, this benchmark still can be regarded as a worst case scenario for the case when users are performing write intensive operations.

The benchmark and the motivation behind writing a new benchmark are described in Section 4.4. It is written in Java and uses a configurable number of threads and input/output files and a configurable I/O stress level. It can also be executed as a single thread application. The statistics are measured by timing the whole duration of the test. The amount of data to write and read is configurable and each thread picks a random file and a random pattern from a set of patterns generated at runtime. This pattern is written to the file at a random offset. The threads that perform read operations simply copy the data into an array in memory.

The results of running two experiments with this benchmark are summarized in Table 5.1.

Configuration	Total Running Time (seconds)
No pause between writes	
Standalone	14
Connected	21
Migration	55
Pause between writes	
Standalone	16
Connected	27
Migration	50

Table 5.1: Benchmark completion time for the configuration where there are no pauses between writes and for the configuration where there are pauses between writes. 260 MB were written during each run of the benchmark.

As expected, the I/O performance decreases when replication is employed. Migration of the memory uses half of the network bandwidth, leaving only half of the bandwidth for persistent storage replication. Therefore, the completion time for the I/O test is almost double during the migration test.

The second test in Table 5.1, where the threads pause briefly between read and write tasks does not show a significantly different behavior. The only difference is that the migration test takes less than twice the value obtained in the *Connected* test. This is a confirmation that the live migration solution works better for workloads that are close to user behavior (oscillations between idle time and write bursts) than for a workload that continuously writes to the disk.

The tests confirm the expected behavior of synchronous replication. Synchronous replication slows down the tests with many write bursts. It is expected that user workloads generate infrequent write bursts [33] and an asynchronous replication protocol will considerably improve the I/O completion times, provided a big enough write buffer is available. Asynchronous storage replication is the subject of future work. This was an important lesson learned when I evaluated the performance of the system.

Similar to the network performance experiments, the experiments in this section simulated a user that is generating many disk reads and writes, which is a worst case scenario for the migration given that typically users perform less intensive tasks. The experiments show that provided the network connection is stable, the migration doesn't fail. Even during migration, the user environment is able to sustain an I/O rate of 5000 KBytes/s for the Bonnie++ benchmark in the network setup used for testing.

5.4 Wrap-up

This chapter presented the performance evaluation of the implemented proof of concept prototype. It shows that live migration is possible with an almost unperceivable downtime as seen by a server and a 4 seconds downtime as seen by the client. Moreover, write performance was decreased for certain workloads up to 3 times compared to the case when the guest is not migrated. Since this is just a proof of concept prototype, it is expected that these results can be improved by further optimizing the implementation. The tests did not represent an idle user capsule, on the contrary, they were designed to test for cases when the user capsule is running under a significant network and disk stress. Therefore, performance degradation was expected. However, the measurements in this chapter show an acceptable performance level which users could be happy to accept in view of the benefit of being able to live migrate their environments.

Chapter 6

Future Work

The architecture and proof of concept presented in this work demonstrate that live migration of user environments can be achieved with a small downtime. A number of issues are the object of future work and are detailed in this chapter.

There are certainly other approaches suited for the proposed scenario. An interesting research area is to investigate and compare approaches to use other types of virtualization layers instead of system virtual machines. A higher level virtual machine allows migration between different hardware platforms but limits the degree of customization the user can apply to her virtual environment and the software availability for the respective virtual machine.

The first two directions of future work refer to obtaining realistic user traces and building asynchronous replication into the implementation of the system. These are well defined technical challenges and can most likely be completed in a reasonable amount of time. The remaining two future work directions are related to estimating the impact of live migration on user experience and the use of hypervisors for mobile devices to allow migration of user environments from a computer to a mobile device and vice-versa. These two directions are very complex research challenges that are not likely to be solved easily.

6.1 User Activity Traces

One interesting aspect is evaluating the impact of live migration on the performance the user is able to obtain from her environment while live migration is performed. A Java based synthetic benchmark that emulates user behavior has already been presented in this work. However, it can be improved by the insight obtained from studying the I/O patterns obtained from large sets of real user data.

Accurately simulating real-user workloads can be done by either using the available SYSmark [12] benchmark, or by profiling workloads from existing users over a period of time and emulating them for the actual experiments. Network activity traces as well as the update patterns and average update rates to the memory have to be collected. These traces can either be replayed or emulated by a synthetic benchmark.

6.2 Asynchronous Replication

The current prototype implementation can be improved to support asynchronous replication with write coalescing which is an essential component required for decreasing the overhead of migration in the presence of low bandwidth links with a large delay. Other techniques such as compression and exploiting redundancy between peers and the user's own backup replicas using content-based indexing can be employed to decrease the network overhead associated with live migration. This item of future work will evaluate the benefits of asynchronous replication in various network bandwidth setups such as cable modem and DSL typical speeds offered by today's providers.

6.3 Estimating the Impact of Migration on User Experience

Anticipating the next destination where a user will use the computer is an open research area. One option is to use movement tracking to speculate on the user's next location. This should dramatically decrease the amount of time a user needs to wait for her machine to be resumed at a new location in case she absentmindedly forgot to configure

it at her last location.

One may argue that live migration does not give any guarantees about the moment when the user can resume work. I intend to address this by writing an estimator for the remaining duration of live migration. This implies estimating the average write throughput as well as how many of the pages will be overwritten within a single iteration.

Insight from previous work shows that in many scenarios, live migration and non-live migration cannot be performed in time to allow the user to resume work after a short commute. Portable storage devices can help alleviate this problem but their use has only been studied in the context of non live relocation. The circumstances change when using live relocation because the memory is continuously updated during live relocation and a portable device can only contain, at best, a snapshot of a previous state of the environment. Using a mobile device such as a PDA could be a better alternative to a portable hard drive and is discussed in Section 6.4.

If migration is not likely to be possible within the required time-frame, then this is either because the write throughput is too high or the user is migrating to a destination that contains a very old replica. Being able to estimate when live migration is possible is thus a key feature. In a very bandwidth constrained scenario, falling back to non-live migration might heavily decrease the estimated transfer time. If this estimate is likely to exceed the time required for the commute, the user will be required to use a portable storage device or a mobile computing device as discussed previously.

6.4 Live Migration across Hypervisors for Mobile Devices

Mobile devices such as phones and PDAs are already ubiquitous and benefit from increasingly more computing power. They are likely to be used more often for simple tasks such as office work, reading email or basic entertainment. This work has only been concerned with migration from computer to computer. A very interesting future work topic is to facilitate live migration of user capsules from a computer to a mobile device such as a mobile phone and vice versa.

This task requires using a VM that runs on a mobile processor (one option is Xen which

has been ported by Samsung to the ARM processor). There are also other efforts to create hypervisors for mobile platforms such as the TRANGO hypervisor [13]. However, live migration of system virtual machines across different architectures is currently not supported by any of the virtual machine technologies. Migration between different ISAs may be achieved using binary rewriting techniques which can be slow. There is also encouraging ongoing work on direct OS live migration between physical machines [51]. This work does not require a virtualization layer and considers migration between machines with different hardware.

Besides the ISA incompatibilities, migrating a guest virtual machine between two very different hardware platforms poses several problems. Firstly, the mobile device is likely to use a WiFi device while the desktop or laptop x86 machine will have a wired network card. This implies creating a network driver to allow seamless migration from a wired network interface card to a wireless network interface card. There is also a mismatch between the computing capabilities of the desktop and the mobile host. This means that the guest virtual machine has to be configured with software that can run at reasonable performance and at a low energy cost on a such a low powered device. Storage is going to be heavily limited as well. On the other hand, the user will be able to run any PC software on her mobile device. This encourages the user to build several virtual appliances and to split or replicate some of the software and functionality among them. The difference in storage capabilities between mobile devices and desktop PCs is at the moment at least one order of magnitude (a typical laptop has approximately a 250GB hard drive and the iPhone has 16GB of storage) therefore, migration will not be possible unless the guest capsule is a stripped version of a typical operating system.

There are several advantages of this approach: running the desktop software stack, replacing the synchronization between replicas via the network with a fast local synchronization via USB and being able to work during a commute.

Chapter 7

Summary and Conclusions

In this work I propose a user mobility model based on the idea of live migrating entire user environments. User environments are *live migrated* during user commutes on locally available hardware where the user can take advantage of the high performance of software running on local hardware. Moreover, the user environment is never interrupted and network connectivity is not disrupted at any time. I propose some user scenarios that motivate why this approach is desirable and define the requirements and limitations that characterize the proposed solution.

An overview of related work is presented and similar approaches are compared to the solution I propose. This work defines a general architecture for live migration of user environments based on a few building blocks: a virtualization layer, persistent and transient state replication as well as network mobility protocols. A specific architecture based on system virtual machines is defined and discussed in detail.

I have implemented an initial proof of concept prototype of the architecture proposed using the Xen virtual machine which addresses the challenges of live migration of persistent storage and network connections across subnets. The system's performance is evaluated in a real testbed using various benchmarks and shows that live migration of user environments is a promising user mobility solution. A number of future work improvements and open problems are discussed in detail, including optimizing the downtime caused by live migration.

The initial conclusions that emerge from the solution I propose are that live migration of

user environments is a viable and promising solution for user mobility. It provides uninterrupted computing in the sense that the user capsule is never restarted and network connectivity is maintained. It also shows that live migration of user environments across subnets is possible without having to restart network connections. This work also discusses the network overhead caused by the actual migration process and suggests ways to substantially minimize this overhead as part of future work. Based on the initial results obtained in this thesis, I estimate that this approach to user mobility has the potential to provide a practical user mobility model in the future.

One of the lessons learned is that the live migration of the persistent state can considerably slowdown the tasks running in the migrated user capsule. This is a disadvantage of using this approach if performance, and not mobility is the user's priority. On the other hand, once migration is completed, the approach described in the thesis allows the user to run her user capsule on available hardware, at high speed which is a significant advantage over other approaches towards user mobility.

The specific architecture and implementation presented in this thesis are a first step in providing live migration to full user environments. This work addresses some of the fundamental problems of live migration but it is not a complete solution. There are limitations such as using live migration in heterogeneous environments. For instance, if a CDROM drive or sound card are not present at the migration destination, these devices will be disconnected after migration. Of course, the user can be notified that these devices are not available at the destination and the user can choose not to migrate.

This thesis has shown that live migration of user environments across wide area networks is a highly desirable mobility solution by presenting some compelling scenarios. It also proposed an architecture, implemented and evaluated a proof of concept prototype that shows that live migration of user environments has the potential to improve user mobility in a pervasive computing environment. Moreover, it provided some directions for future work and suggested improvements that can further enhance the performance of the system and make it an effective and desired solution to user mobility.

Bibliography

- [1] The Bonnie++ benchmark. <http://www.coker.com.au/bonnie++/> accessed on December 2007.
- [2] Csync2. <http://oss.linbit.com/csync2/> accessed on December 2007.
- [3] Iometer benchmark. <http://www.iometer.org/>.
- [4] Kvm - Kernel-based Virtualization Machine. Qumranet's KVM Whitepaper.
- [5] Linux Vserver. <http://linux-vserver.org/>.
- [6] Logical volume manager. <http://sourceware.org/lvm2/> accessed on December 2007.
- [7] Microsoft Virtual PC. <http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx>.
- [8] Openvz. <http://wiki.openvz.org>.
- [9] Qemu. <http://bellard.org/qemu/>.
- [10] Software RAID. <http://tldp.org/HOWTO/Software-RAID-HOWTO.html> accessed on December 2007.
- [11] Ssh. <http://www.openssh.com/>.
- [12] SYSmark benchmark. <http://www.bapco.com/>.
- [13] Trango Systems. <http://www.trango-systems.com> accessed on December 2007.
- [14] Trusted platform module (TPM) specifications.
<https://www.trustedcomputinggroup.org/specs/TPM/>.
- [15] Virtual Network Computing. <http://www.cl.cam.ac.uk/research/dtg/attarchive/vnc/xvnc.html>, access on December 2007.

- [16] Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent, 6397242, Oct. 1998.
- [17] Wget. <http://www.gnu.org/software/wget/>.
- [18] Address allocation for private Internets, RFC 1918.
- [19] Security architecture for the Internet Protocol, RFC 2401.
- [20] NAT and firewall traversal for HIP, Internet Draft.
- [21] BARATTO, R. A., POTTER, S., SU, G., AND NIEH, J. Mobidesk: mobile virtual desktop computing. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking* (New York, NY, USA, 2004), ACM Press, pp. 1–15.
- [22] BERGER, S., CÁCERES, R., GOLDMAN, K. A., PEREZ, R., SAILER, R., AND VAN DOORN, L. vTPM: virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2006), USENIX Association.
- [23] BITTMAN, T. Predicts 2004: Server virtualization evolves rapidly.
- [24] BOLOSKY, W. J., DOUCEUR, J. R., ELY, D., AND THEIMER, M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on measurement and modeling of computer systems* (New York, NY, USA, 2000), ACM, pp. 34–43.
- [25] BRADFORD, R., KOTSOVINOS, E., FELDMANN, A., AND SCHIÖBERG, H. Live wide-area migration of virtual machines including local persistent state. In *VEE '07: Proceedings of the 3rd international conference on virtual execution environments* (New York, NY, USA, 2007), ACM, pp. 169–179.
- [26] CÁCERES, R., CARTER, C., NARAYANASWAMI, C., AND RAGHUNATH, M. Reincarnating PCs with portable SoulPads. In *MobiSys '05: Proceedings of the 3rd international conference on mobile systems, applications, and services* (New York, NY, USA, 2005), ACM, pp. 65–78.
- [27] CASHIN, E. L. Kernel korner: ATA over Ethernet: putting hard drives on the LAN. *Linux J.* 2005, 134 (2005), 10.

- [28] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *Proc. of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (Boston, MA, May 2005), pp. 273–286.
- [29] COX, L. P., MURRAY, C. D., AND NOBLE, B. D. Pastiche: making backup cheap and easy. *SIGOPS Oper. Syst. Rev.* 36, SI (2002), 285–298.
- [30] CULLY, B., AND WARFIELD, A. Secondsite: disaster protection for the common server. In *HOTDEP'06: Proceedings of the 2nd conference on Hot Topics in System Dependability* (Berkeley, CA, USA, 2006), USENIX Association, pp. 12–12.
- [31] DEERING, S., AND HINDEN, R. Internet Protocol, version 6 specification, RFC 2460.
- [32] DEFANTI, T., DE LAAT, C., MAMBRETTI, J., NEGGERS, K., AND ARNAUD, B. S. Translight: a global-scale lambdagrid for e-science. *Commun. ACM* 46, 11 (2003), 34–41.
- [33] DETHE, C., AND WAKDE, D. User based network (WAN) traffic analysis. *Communications, 2003. APCC 2003. The 9th Asia-Pacific Conference on* 3 (Sept. 2003), 1152–1156 Vol.3.
- [34] DRAGOVIC, B., KOTSOVINOS, E., HAND, S., AND PIETZUCH, P. R. Xenotrust: Event-based distributed trust management. In *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications* (Washington, DC, USA, 2003), IEEE Computer Society, p. 410.
- [35] EGGERT, L. TCP abort timeout option Internet Draft.
- [36] EGGERT, L. TCP extensions for immediate retransmissions, Internet Draft.
- [37] FLAUTNER, K., UHLIG, R., REINHARDT, S., AND MUDGE, T. Thread-level parallelism and interactive performance of desktop applications. *SIGPLAN Not.* 35, 11 (2000), 129–138.
- [38] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on operating systems principles* (New York, NY, USA, 2003), ACM, pp. 193–206.

- [39] HANSEN, J. G., AND JUL, E. Self-migration of operating systems. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop: beyond the PC* (New York, NY, USA, 2004), ACM Press, p. 23.
- [40] HITZ, D., LAU, J., AND MALCOLM, M. File system design for an NFS file server appliance. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference* (Berkeley, CA, USA, 1994), USENIX Association, pp. 19–19.
- [41] HOWARD. An overview of the Andrew File System. In *USENIX Winter Technical Conference* (February 1988).
- [42] JI, M. Instant snapshots in a federated array of bricks. *Technical Report HPL-2005-15, HP Laboratories* (2005).
- [43] JIANG, X., AND XU, D. Violin: Virtual internetworking on overlay infrastructure. In *Proc. of the 2nd Intl. Symp. on Parallel and Distributed Processing and Applications* (2003).
- [44] JOHNSON, D., PERKINS, C., AND ARKKO, J. Mobility support in IPv6, RFC 3755.
- [45] JOKELA, P., RINTA-AHO, T., JOKIKYYNY, T., WALL, J., KUPARINEN, M., MAHKONEN, H., MELEN, J., KAUPPINEN, T., AND KORHONEN, J. Handover performance with HIP and MIPv6. In *1st International Symposium on Wireless Communication Systems* (2004).
- [46] JOUKOV, N., WONG, T., AND ZADOK, E. Accurate and efficient replaying of file system traces. In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2005), USENIX Association, pp. 25–25.
- [47] JURGENS, C. Fibre channel: A connection to the future. *Computer* 28, 8 (1995), 88–90.
- [48] KAMP, P.-H., AND WATSON, R. N. M. Jails: Confining the omnipotent root. In *In Proc. 2nd Intl. SANE Conference* (2000).
- [49] KEETON, K., SANTOS, C., BEYER, D., CHASE, J., AND WILKES, J. Designing for disasters. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2004), USENIX Association, pp. 59–62.
- [50] KEY, P., MASSOULIÉ, L., AND WANG, B. Emulating low-priority transport at the application layer: a background transfer service. In *SIGMETRICS '04/Performance*

'04: *Proceedings of the joint international conference on measurement and modeling of computer systems* (New York, NY, USA, 2004), ACM, pp. 118–129.

- [51] KOZUCH, M. A., KAMINSKY, M., AND RYAN, M. P. Migration without virtualization. In *HOTOS '09: Proceedings of the 12th Workshop on Hot Topics in Operating Systems* (2009).
- [52] KUZMANOVIC, A., AND KNIGHTLY, E. W. TCP-LP: low-priority service via endpoint congestion control. *IEEE/ACM Trans. Netw.* 14, 4 (2006), 739–752.
- [53] LAGAR-CAVILLA, A., TOLIA, N., BALAN, R., DE LARA, E., SATYANARAYANAN, M., AND O'HALLARON, D. R. Dimorphic computing. *Tech. Report CMU-CS-06-123*.
- [54] LEE, E. K., AND THEKKATH, C. A. Petal: distributed virtual disks. In *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems* (New York, NY, USA, 1996), ACM Press, pp. 84–92.
- [55] LINDHOLM, T., AND YELLIN, F. *The Java Virtual Machine Specification*. Addison-Wesley, 1999.
- [56] MANBER, U. Finding similar files in a large file system. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference* (Berkeley, CA, USA, 1994), USENIX Association, pp. 2–2.
- [57] METH, K. Z., AND SATRAN, J. Design of the iSCSI protocol. In *MSS '03: Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 116.
- [58] MOSKOWITZ, R., AND NIKANDER, P. Host identity protocol draft, draft-moskowitz-hip-06, work in progress.
- [59] MUTHITACHAROEN, A., CHEN, B., AND MAZIÈRES, D. A low-bandwidth network file system. In *SOSP '01: Proceedings of the eighteenth ACM symposium on operating systems principles* (New York, NY, USA, 2001), ACM, pp. 174–187.
- [60] NATH, P., KOZUCH, M., O'HALLARON, D., HARKES, J., SATYANARAYANAN, M., TOLIA, N., AND TOUPS, M. Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines. In *Proceedings of the 2006 USENIX Annual Technical Conference (USENIX '06)* (Boston, MA, June 2006).

- [61] PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. A case for redundant arrays of inexpensive disks (RAID). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 1988), ACM, pp. 109–116.
- [62] PATTERSON, R. H., MANLEY, S., FEDERWISCH, M., HITZ, D., KLEIMAN, S., AND OWARA, S. Snapmirror: File-system-based asynchronous mirroring for disaster recovery. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2002), USENIX Association, p. 9.
- [63] PAWLOWSKI, B., JUSZCZAK, C., STAUBACH, P., SMITH, C., LEBEL, D., AND HITZ, D. NFS Version 3: Design and Implementation. In *USENIX Summer* (1994), pp. 137–152.
- [64] PERKINS, C. IP mobility support for mobile IPv4, RFC 3344.
- [65] POSTEL, J. Transmission control protocol RFC 793.
- [66] QUINLAN, S. A cached worm file system. *Softw. Pract. Exper.* 21, 12 (1991), 1289–1299.
- [67] QUINLAN, S., AND DORWARD, S. Venti: A new approach to archival data storage. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2002), USENIX Association, p. 7.
- [68] RAPIER, C., AND STEVENS, M. High performance SSH/SCP - HPN-SSH. <http://www.psc.edu/networking/projects/hpn-ssh/>.
- [69] REISNER, P. DRBD v8 - replicated storage with shared disk semantics. In *Proc. of the 12th International Linux System Technology Conf* (2005).
- [70] ROSELLI, D., LORCH, J. R., AND ANDERSON, T. E. A comparison of file system workloads. In *ATEC'00: Proceedings of the Annual Technical Conference on 2000 USENIX Annual Technical Conference* (Berkeley, CA, USA, 2000), USENIX Association, pp. 4–4.
- [71] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* (November 2001), pp. 329–350.
- [72] RUBINI, A. Kernel korner: The "virtual file system" in Linux. *Linux J.* 1997, 37es (1997), 21.

- [73] SAITO, Y., FROLUND, S., VEITCH, A., MERCHANT, A., AND SPENCE, S. FAB: building distributed enterprise disk arrays from commodity components. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems* (New York, NY, USA, 2004), ACM, pp. 48–58.
- [74] SAPUNTZAKIS, C., AND LAM, M. S. Virtual appliances in the collective: a road to hassle-free computing. In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems* (Berkeley, CA, USA, 2003), USENIX Association, pp. 10–10.
- [75] SAPUNTZAKIS, C. P., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Optimizing the migration of virtual computers. *SIGOPS Oper. Syst. Rev.* 36, SI (2002), 377–390.
- [76] SATYANARAYANAN, M., KISTLER, J. J., KUMAR, P., OKASAKI, M. E., SIEGEL, E. H., AND STEERE, D. C. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers* 39, 4 (1990), 447–459.
- [77] SATYANARAYANAN, M., KOZUCH, M., HELFRICH, C., AND O'HALLARON, D. R. Towards seamless mobility on pervasive hardware. In *Pervasive and Mobile Computing 1* (2005) 157–189.
- [78] STEVENS, W. *TCP/IP Illustrated Volume 1: The Protocols*. Addison-Wesley, 1994.
- [79] SUNDARARAJ, A. I., AND DINDA, P. A. Towards virtual networks for virtual machine grid computing. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium* (Berkeley, CA, USA, 2004), USENIX Association, pp. 14–14.
- [80] TOLIA, N., ANDERSEN, D. G., AND SATYANARAYANAN, M. Quantifying interactive user experience on thin clients. *Computer* 39, 3 (2006), 46–52.
- [81] TRAVOSTINO, F., DASPIT, P., GOMMANS, L., JOG, C., DE LAAT, C., MAMBRETTI, J., MONGA, I., VAN OUDENAARDE, B., RAGHUNATH, S., AND WANG, P. Y. Seamless live migration of virtual machines over the MAN/WAN. *Future Gener. Comput. Syst.* 22, 8 (2006), 901–907.
- [82] TRIDGELL, A., AND MACKERRAS, P. The Rsync algorithm. Tech. rep., Department of Computer Science Australian National University, 1996.
- [83] VENKATARAMANI, A., KOKKU, R., AND DAHLIN, M. TCP Nice: a mechanism for background transfers. *SIGOPS Oper. Syst. Rev.* 36, SI (2002), 329–343.

- [84] WARFIELD, A., HAND, S., FRASER, K., AND DEEGAN, T. Facilitating the development of soft devices. In *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), USENIX Association, pp. 22–22.
- [85] WARFIELD, A., ROSS, R., FRASER, K., LIMPACH, C., AND HAND, S. Parallax: Managing storage for a million machines. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Operating Systems (HotOS X)* (Santa Fe, NM, June 2005).
- [86] WHITAKER, A., SHAW, M., AND GRIBBLE, S. D. Scale and performance in the Denali isolation kernel. In *OSDI '02: Proceedings of the 5th symposium on operating systems design and implementation* (New York, NY, USA, 2002), ACM, pp. 195–209.