N° d'ordre : 4444 ANNÉE 2011





THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Pierre Riteau

préparée à l'unité de recherche n° 6074 - IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires
ISTIC

Dynamic Execution Platforms over Federated Clouds

Thèse soutenue à Rennes
le 2 décembre 2011

devant le jury composé de :

Frédéric Desprez / Rapporteur
Directeur de Recherche, Inria Grenoble - Rhône-Alpes

Pierre Sens / Rapporteur
Professeur, Université Paris 6

Gabriel Antoniu / Examinateur
Chargé de recherche, HDR, Inria Rennes - Bretagne Atlantique

Noël De Palma / Examinateur
Professeur, Université de Grenoble

Kate Keahey / Examinateur
Scientist, Argonne National Laboratory

Guillaume Pierre / Examinateur
Associate professor, Vrije Universiteit Amsterdam

Thierry Priol / Directeur de thèse
Directeur de Recherche, Inria Rennes - Bretagne Atlantique

Christine Morin / Co-directeur de thèse
Directrice de Recherche, Inria Rennes - Bretagne Atlantique

Acknowledgments

I would like to start these acknowledgments by thanking all the members of my thesis defense committee. Thank you to Frédéric Desprez and Pierre Sens for making me the honor of reading and evaluating my thesis manuscript, even though you were on very tight deadlines because of me! Thank you to Noël De Palma for acting as president of the committee. Thank you to Gabriel Antoniu, Guillaume Pierre, and Kate Keahey for participating in the committee and evaluating my work.

Little did I know, when I first met Kate at SC08 and started using Nimbus a few weeks later, that it would lead to such a productive collaboration. It is clear that this thesis would have been very different had Kate not been involved.

Over these three years, I was advised by Thierry and Christine. Thank you to both of you. Although it was sometimes difficult to meet due to busy schedules (mine included!), you were always there to guide and advise me. I am extremely grateful for the freedom you offered me in my work. In all our interactions I always felt like a colleague rather than a student.

Very big thank you to all members of the PARIS and Myriads teams throughout the years. It has been a joy to work and spend time with all of you. Thank you to the members of the KerData team with whom I often interacted. Thank you to my colleagues from my teaching duties, in particular Laurent and Achour. And thank you to all the great colleagues from the laboratory that make it such a nice workplace. Thank you Maryse for helping with all the administrative tasks. Thank you Yvon: I learned a lot from you, and you were an excellent travel buddy. Thank you Amine, Julien, Landry, Alexandra, Roberto, Ghislain, Stefania, Loïc, Djawida, Maxence, Eugen, Héctor, Jérôme, Rémy, Guillaume, Piyush, Anca, Sylvain, Matt, André, Box, Adrien, Raúl, Sinziana, Diana, Izabela, Pascal, Bogdan, Marko, Cyril, Thomas, Amélie, Trung... and all others whom I forget, for the many excellent times we had together. Big thanks to Ghislain and his accomplices for the hilarious thesis poster.

E206, E224, E207. I occupied three different offices, but my office buddy remained the same. Thanks a lot Jérôme. It was always delightful and fun to spend my working days with you, even when rain was pouring down on our computers ;-) I will remember your kindness and sunny mood, even in the difficult times of your own thesis writing. And thank you to Sylvie for her delicious homemade cookies!

Adrien, thank you for introducing me to the world of research during my Master internship. You warned me numerous times about the difficulties of doing a PhD, and you were right. But you didn't scare me enough to flee ;-)

University of Florida became my home for a visit of three months at the ACIS laboratory in summer 2010. Thank you to José Fortes and Maurício Tsugawa for hosting me and making it such a fruitful collaboration. Thank you to all members of ACIS, I had such a great time! My summer in Florida was also made very enjoyable by meeting two great people, Matt and Kate. Thank you for being such good friends. I hope to see you soon!

Reading further into this manuscript, you will notice that I made extensive use of the Grid'5000 platform. Thus, I want to express my gratitude to everyone who contributes to making Grid'5000 possible, in particular Pascal, Cyril, Ghislain, David, Sébastien, Emmanuel, Lucas, and Tina.

Even though we don't meet as often as we used to since you all stayed South of

Brittany, I would like to thank my friends from earlier times: Bertrand, Éric C., Éric D., Étienne, Hugo, Thomas, Yvan.

Like I said before, I became quickly involved with the Nimbus project. Thank you to the previous and current members of the Nimbus team: Kate, Tim, David, John, Patrick. I am glad to be part of the team now and to work with you daily.

I would like to thank Djawida and Anca who agreed to work under my supervision for several months. Our exchanges have always been very interesting, and I hope you enjoyed working with me as much as I enjoyed working with you. I wish you the best for your future journeys.

Enfin, je souhaite remercier ma famille pour leur support, leurs encouragements et leur aide continuelle toutes ces années.

Contents

1 Introduction **11**
- 1.1 Preamble . . . 12
- 1.2 Objectives . . . 13
- 1.3 Contributions . . . 13
- 1.4 Publications . . . 14
- 1.5 Organization of the Dissertation . . . 16

I Background **19**

2 Background **21**
- 2.1 Virtualization Technologies . . . 22
  - 2.1.1 Concept . . . 22
  - 2.1.2 Types of Virtualization Technologies . . . 22
  - 2.1.3 Live Virtual Machine Migration . . . 31
- 2.2 Parallel and Distributed Computing . . . 35
  - 2.2.1 Types of Applications . . . 36
  - 2.2.2 Architectures . . . 39
- 2.3 Cloud Computing . . . 40
  - 2.3.1 Abstraction Layers . . . 41
  - 2.3.2 Deployment Types . . . 42
  - 2.3.3 Infrastructure as a Service Clouds . . . 42
  - 2.3.4 Open Problems in IaaS Clouds . . . 46
- 2.4 Federation of Cloud Computing Infrastructures . . . 46
  - 2.4.1 Sky Computing . . . 48
  - 2.4.2 Other Cloud Federation Efforts . . . 49
- 2.5 Conclusion . . . 49

II Elastic Execution Platforms over Federated Clouds **53**

3 Large-Scale and Elastic Execution Platforms over Federated Clouds **55**
- 3.1 Introduction . . . 57
- 3.2 Scientific Computing on Clouds . . . 57
- 3.3 Sky Computing . . . 58
  - 3.3.1 ViNe . . . 58
  - 3.3.2 Nimbus Context Broker . . . 58
- 3.4 Large-scale Sky Computing Experiments . . . 59

3.4.1 Elasticity . . . 59
3.4.2 Experimental Infrastructures . . . 59
3.4.3 Deployment of Nimbus Clouds . . . 61
3.5 Scalability of Cloud Infrastructures . . . 62
3.5.1 Propagation with a Broadcast Chain . . . 64
3.5.2 Propagation using Copy-on-write Volumes . . . 64
3.5.3 Performance Evaluation . . . 65
3.5.4 Combination of the two mechanisms . . . 66
3.5.5 Related work . . . 67
3.6 Scalability of Sky Computing Platforms . . . 67
3.7 Extending Elastic Sky Computing Platforms . . . 68
3.8 Related Work . . . 69
3.9 Conclusion . . . 70

4 Resilin: Elastic MapReduce for Private and Community Clouds 71
4.1 Introduction . . . 72
4.2 Amazon Elastic MapReduce . . . 72
4.3 Architecture and Implementation . . . 73
4.3.1 Multi-Cloud Job Flows . . . 76
4.4 Evaluation . . . 77
4.4.1 Deployment Time . . . 78
4.4.2 Execution Time . . . 79
4.4.3 Cloud Storage Performance . . . 81
4.4.4 Financial Cost . . . 82
4.5 Related Work . . . 83
4.6 Conclusion . . . 84

III Dynamic Execution Platforms with Inter-Cloud Live Migration 87

5 Network-level Support for Inter-Cloud Live Migration 89
5.1 Introduction . . . 90
5.2 Problem Statement . . . 90
5.3 Related Work . . . 92
5.3.1 Mobile IP . . . 93
5.3.2 Overlay Networks . . . 93
5.3.3 Virtual Private Networks (VPN) . . . 93
5.3.4 ARP and Tunneling . . . 93
5.4 Architecture of ViNe . . . 94
5.5 Live Migration Support in ViNe . . . 96
5.5.1 Automatic Detection of Live Migration . . . 96
5.5.2 Virtual Network Reconfiguration . . . 96
5.5.3 ARP and tunneling mechanisms . . . 97
5.5.4 Routing Optimizations . . . 99
5.6 Implementation . . . 99
5.7 Conclusion . . . 100

6 Shrinker: Efficient Live Migration of Virtual Clusters over WANs 101
6.1 Introduction . . . 102
6.2 Related Work . . . 102
6.3 Architecture of Shrinker . . . 103
6.3.1 Architecture Overview . . . 103
6.3.2 Security Considerations . . . 105
6.4 Implementation . . . 106
6.4.1 Prototype Limitations . . . 108
6.5 Evaluation . . . 108
6.5.1 Evaluation Methodology . . . 108
6.5.2 Evaluation with Different Workloads . . . 109
6.5.3 Impact of Available Network Bandwidth . . . 113
6.5.4 Impact of Deduplication Granularity . . . 113
6.6 Conclusion . . . 116

IV Conclusion 117

7 Conclusions and Perspectives 119
7.1 Contributions . . . 120
7.2 Perspectives . . . 122
7.2.1 Short Term Perspectives . . . 122
7.2.2 Long Term Research Perspectives . . . 124

Bibliography 125

A Résumé en français 141
A.1 Préambule . . . 141
A.2 Objectifs . . . 142
A.3 Contributions . . . 143
A.3.1 Mécanismes pour la création efficace de plates-formes d'exécution élastiques à large échelle, au-dessus de nuages informatiques fédérés en utilisant l'approche *sky computing* . . . 143
A.3.2 Resilin, un système pour facilement créer des plates-formes d'exécution de calculs MapReduce au-dessus de ressources de nuages informatiques distribués . . . 145
A.3.3 Des mécanismes permettant aux machines virtuelles de réaliser des communications réseaux efficaces en présence de migrations à chaud entre nuages informatiques . . . 146
A.3.4 Shrinker, un protocole de migration à chaud optimisé pour la migration à chaud efficace des grappes de machines virtuelles entre nuages informatiques connectés par des réseaux étendus . . . 147
A.4 Publications . . . 149
A.5 Organisation du manuscrit . . . 150

List of Figures

2.1 Simplified view of hardware virtualization 23
2.2 Type 1 hypervisor with a privileged and two guest operating systems . . 25
2.3 Type 2 hypervisor running two guest operating systems 25
2.4 Example of operating system-level virtualization 29
2.5 Example of process-level virtualization 30
2.6 Generic architecture of an IaaS cloud 44
2.7 Generic architecture of a virtual machine monitor node in an IaaS cloud . 46

3.1 Map of the Grid'5000 testbed 60
3.2 Map of the FutureGrid testbed 62
3.3 Standard SCP-based propagation in Nimbus Infrastructure 63
3.4 Improved propagation based on a broadcast chain 65
3.5 Improved propagation based on pre-propagated copy-on-write volumes . 65
3.6 Propagation performance 66

4.1 Overview of Resilin and its interactions with other cloud components . . 75
4.2 Deployment time of Hadoop clusters on Amazon Elastic MapReduce and Resilin with different number of instances 79
4.3 Execution time of Word Count on Amazon Elastic MapReduce and Resilin with different number of instances 80
4.4 Execution time of CloudBurst on Amazon Elastic MapReduce and Resilin with different number of instances 81
4.5 Comparison of cloud storage performance between Amazon Elastic MapReduce and Resilin (with the Cumulus cloud storage repository) 82

5.1 Example of two separate cloud IP networks 91
5.2 Example deployment of ViNe 95
5.3 ViNe deployment after live migration 98
5.4 Architecture of our mechanisms implemented in ViNe 100

6.1 Distributed data deduplication algorithm 104
6.2 Distributed content-based addressing algorithm 105
6.3 Data structures used to index local memory pages 108
6.4 Shrinker performance for idle virtual machines 110
6.5 Shrinker performance for the NBP cg.C memory-intensive workload . . . 112
6.6 Shrinker performance for the DBENCH I/O-intensive workload 114
6.7 Average total migration time of 16 idle virtual machines 115

6.8 Comparison of the amount of memory fully transferred versus amount of memory deduplicated when migrating 16 idle virtual machines 115

Chapter 1

Introduction

Contents

1.1 Preamble **12**
1.2 Objectives **13**
1.3 Contributions **13**
1.4 Publications **14**
1.5 Organization of the Dissertation **16**

This chapter introduces this PhD thesis. The background of our work is centered around parallel and distributed systems used for executing applications with large computational power requirements, virtualization technologies, and cloud computing infrastructures. We present our objectives for this thesis, introduce our contributions, and describe the organization of this dissertation.

1.1 Preamble

Since the emergence of computer science, an uninterrupted stream of technological improvements has been making computers more accessible and more capable. This tremendous progress in computer technology has allowed heterogeneous computers to spread to companies, homes, purses and pockets around the world. But, at the same time, this increasing computing capability has led to new problems to solve with larger requirements. The software industry has grown rapidly and creates increasingly larger applications to meet the demands of many fields of science and industry. Large quantities of computing power are required to simulate complex problems such as natural phenomenons or fluid dynamics, to analyze genome sequences, to model financial markets, to make computer-generated imagery, etc.

This high demand for computing power, driven by the adoption of computing in all fields of science, industry, and entertainment, cannot be satisfied only by the capabilities of stand-alone computers. This has lead computer scientists to build systems consisting of many processors collaborating to solve problems in a parallel fashion. These systems are either large, special-purpose computers with hundred of thousands of processors (*supercomputers*), or interconnections of large numbers of general-purpose machines (*clusters*). Bringing this idea further, grid systems federate multiple geographically distributed systems together, creating highly distributed infrastructures belonging to multiple organizations. Acquiring and managing such systems is costly, and only academic and industrial organizations with the available financial and human resources can buy and operate such infrastructures. Furthermore, these systems need to be highly utilized to offer a good return on investment.

Another development in computer science has been the popularization of virtualization technologies. Virtualization decouples software from the underlying physical platform, offering more flexibility and stronger isolation. While it was popular in the 1970s, virtualization remained mostly restricted to mainframe computers and did not propagate to the rest of the industry for several decades. This is explained by the availability of efficient time-sharing systems for managing servers, and by the omnipresence of personal workstations. However, in the 2000s, virtualization technologies rose to popularity again, thanks to the availability of implementations with little performance overhead. In particular, virtualization has been increasingly used to efficiently manage distributed computing infrastructures.

These advances in virtualization technologies, combined with the general availability of high speed Internet connections, led to the emergence of cloud computing. Cloud computing offers computing resources of different abstraction levels, provided as on-demand services over the Internet. It builds on ideas that have been circulating in the computer science community for decades. In 1961, John McCarthy, recipient of the Turing award and artificial intelligence pioneer, declared: *"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."* Cloud computing gives the potential to anyone to provision computing resources for a time and in a size exactly matching their requirements, rather than making long term investments by acquiring expensive computing hardware.

In this new context, several problems emerge. How can we easily and efficiently

execute computations requiring large amounts of computing power on cloud infrastructures? How do we take advantage of the new possibilities offered by virtualization technologies in cloud infrastructures?

1.2 Objectives

Given the increasing popularity and availability of cloud computing infrastructures, and distributed virtualized systems in general, this thesis argues that *it is possible to easily and efficiently build large-scale elastic execution platforms over federations of multiple cloud infrastructures*, and that *these execution platforms can be efficiently migrated between multiple cloud infrastructures, making them more dynamic than traditional infrastructures*. Users who need to solve computationally intensive problems, whether they are from the areas of scientific simulation, finance, data mining, etc., now have access to multiple cloud infrastructures of different types. These infrastructures put a lot of control in the hands of users, who may not have the necessary knowledge to make the best use of these resources. These multiple cloud infrastructures offer the potential to be used simultaneously, following an approach known as *sky computing* [130]. Furthermore, these infrastructures provide elasticity, offering methods to create and destroy resources to respond to changing requirements from users and applications. *The first objective of this thesis is to easily and efficiently create large-scale elastic execution platforms on top of federated cloud resources, potentially belonging to different providers and institutions, using the sky computing approach.*

A major shift between traditional computing infrastructures and cloud environments is the use of virtualization as a fundamental mechanism for creating and managing resources. By decoupling software from the underlying physical infrastructure, virtualized environments are not tied to the hardware used for their execution. This lack of dependence allows execution platforms to be migrated between remote machines over network connections, like regular data transfers. This property, implemented in a transparent manner by a mechanism called *live migration*, offers the potential to dynamically relocate execution platforms between cloud infrastructures, in order to take advantage of changes in the environment and optimize the execution of the computations. *In this context, the second objective of this thesis is to make cloud-based execution platforms more dynamic by allowing efficient inter-cloud live migration.*

1.3 Contributions

The contributions presented in this thesis are the following:

Mechanisms to efficiently create large-scale elastic execution platforms over federated clouds using the sky computing approach. We performed a series of real-world experiments creating large-scale, intercontinental sky computing platforms using Nimbus [27, 128] clouds deployed on the Grid'5000 [79] testbed in France and the FutureGrid [15] testbed in the USA. We made these sky computing platforms elastic by extending them to speed up computations. Lessons learned during these experiments led us to design and implement two new mechanisms for efficiently propagating virtual machine images in the Nimbus cloud toolkit. These mechanisms, based on a broadcast chain

and copy-on-write volumes, allow to decrease the time required to create and elastically extend large-scale cloud-based execution platforms.

Resilin, a system to easily create execution platforms over distributed cloud resources for executing MapReduce [96, 97] computations. In order to allow users to easily take advantage of distributed cloud resources, we built Resilin, a system for creating execution platforms for running MapReduce computations. Resilin implements the Amazon Elastic MapReduce web service API and uses resources from private and community clouds. Resilin takes care of provisioning, configuring and managing cloud-based Hadoop [1] execution platforms, potentially using multiple clouds. In this dissertation, we describe the design and implementation of Resilin. We also present the results of a comparative evaluation with the Amazon Elastic MapReduce service.

Mechanisms to allow virtualized resources to perform efficient communications in the presence of inter-cloud live migration. When migrating virtual machines between different clouds, the network infrastructure needs to be adapted to allow virtual machines to continue communicating. We propose mechanisms to automatically detect live migrations and reconfigure virtual networks to keep communications uninterrupted. Our approach optimizes communication performance when virtual machines are located on the same physical network. We implemented a prototype of our mechanisms for the ViNe virtual network and validated our approach experimentally.

Shrinker, a live migration protocol optimized to efficiently migrate virtual clusters between data centers connected by wide area networks. Live migration allows execution platforms to be dynamically relocated between different infrastructures. Live migration across multiple clouds offers a lot of advantages, both for users and administrators. However, the large amount of traffic generated when transferring large virtual clusters makes live migration inefficient over wide area networks. We propose Shrinker, a live migration protocol eliminating identical data among virtual machines to reduce the traffic generated by inter-cloud live migration. Shrinker uses distributed data deduplication and distributed content-addressing to achieve its goal. We implemented a prototype in the KVM [132] hypervisor. Evaluations on the Grid'5000 testbed show that Shrinker reduces both traffic generated by live migration and total migration time.

The work presented in this PhD thesis was carried out in the Myriads research team (previously PARIS), part of the IRISA laboratory and the Inria Rennes - Bretagne Atlantique research center, with collaborations with the Nimbus project team at Argonne National Laboratory / University of Chicago Computation Institute and with the ACIS laboratory at University of Florida.

1.4 Publications

The contributions presented in this document were published in several peer-reviewed books, journals, conferences, workshops, and magazines.

Book Chapters

- Andréa Matsunaga, Pierre Riteau, Maurício Tsugawa, and José Fortes. Crosscloud Computing. In *High Performance Computing: From Grids and Clouds to Exascale*, volume 20 of *Advances in Parallel Computing*, pages 94–108. IOS Press, 2011.

International Journals

- Christine Morin, Yvon Jégou, Jérôme Gallard, and Pierre Riteau. Clouds: a New Playground for the XtreemOS Grid Operating System. *Parallel Processing Letters*, 19(3):435–449, September 2009.

International Conferences

- Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker: Improving Live Migration of Virtual Clusters over WANs with Distributed Data Deduplication and Content-Based Addressing. In *17th International Euro-Par Conference on Parallel Processing*, pages 431–442, 2011.

International Workshops

- Maurício Tsugawa, Pierre Riteau, Andréa Matsunaga, and José Fortes. User-level Virtual Networking Mechanisms to Support Virtual Machine Migration Over Multiple Clouds. In *2nd IEEE International Workshop on Management of Emerging Networks and Services*, pages 568–572, 2010.

Posters in International Conferences & Workshops

- Pierre Riteau. Building Dynamic Computing Infrastructures over Distributed Clouds. In *2011 IEEE International Symposium on Parallel and Distributed Processing: Workshops and PhD Forum*, pages 2097–2100, 2011. Best Poster Award.

- Pierre Riteau, Kate Keahey, and Christine Morin. Bringing Elastic MapReduce to Scientific Clouds. In *3rd Annual Workshop on Cloud Computing and Its Applications: Poster Session*, 2011.

- Pierre Riteau, Maurício Tsugawa, Andréa Matsunaga, José Fortes, Tim Freeman, David LaBissoniere, and Kate Keahey. Sky Computing on FutureGrid and Grid'5000. In *5th Annual TeraGrid Conference: Poster Session*, 2010.

International Magazines

- Pierre Riteau, Maurício Tsugawa, Andréa Matsunaga, José Fortes, and Kate Keahey. Large-Scale Cloud Computing Research: Sky Computing on FutureGrid and Grid'5000. *ERCIM News*, (83):41–42, October 2010.

Research Reports

- Pierre Riteau, Ancuţa Iordache, and Christine Morin. Resilin: Elastic MapReduce for Private and Community Clouds. Research report RR-7767, *Institut national de recherche en informatique et en automatique (Inria)*, October 2011.

- Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker: Efficient Wide-Area Live Virtual Machine Migration using Distributed Content-Based Addressing. Research report RR-7198, *Institut national de recherche en informatique et en automatique (Inria)*, February 2010.

- Christine Morin, Yvon Jégou, Jérôme Gallard, and Pierre Riteau. Clouds: a New Playground for the XtreemOS Grid Operating System. Research report RR-6824, *Institut national de recherche en informatique et en automatique (Inria)*, February 2009.

Technical Reports

- Eliana-Dina Tîrşa, Pierre Riteau, Jérome Gallard, Christine Morin, and Yvon Jégou. Towards XtreemOS in the Clouds – Automatic Deployment of XtreemOS Resources in a Nimbus Cloud. Technical report RT-0395, *Institut national de recherche en informatique et en automatique (Inria)*, October 2010.

1.5 Organization of the Dissertation

The rest of this dissertation is organized in four parts. The first part, consisting of Chapter 2, presents the state of the art in the context of our work. We present an overview of virtualization technologies, which are used in our contributions. We describe parallel and distributed computing, and the different types of applications and architectures used to solve problems requiring large amounts of computational power. Finally, we introduce the concept of cloud computing, bringing together virtualization and distributed systems, and the ongoing efforts towards federating cloud computing infrastructures.

The second part presents our contributions for the creation of large-scale elastic execution platforms from federated cloud resources, using the sky computing approach. Chapter 3 presents our experiments with large-scale sky computing platforms performed on Grid'5000 and FutureGrid. From the lessons learned during these experiments, we propose two improved virtual machine image propagation mechanisms. They allow to create execution platforms in clouds more efficiently. We also make sky computing platforms elastic by extending them to speed up computations. Chapter 4 presents Resilin, a system to easily perform MapReduce computations on distributed cloud resources. We present the design and implementation of Resilin, and a comparative evaluation with Amazon Elastic MapReduce.

The third part describes our contributions to make cloud-based execution platforms more dynamic using efficient inter-cloud live migration of virtual machines. Chapter 5 describes mechanisms to allow virtual machines to continue communicating efficiently in the presence of inter-cloud migration, and their implementation in the ViNe virtual network. Chapter 6 presents Shrinker, a system optimizing live migration of virtual clusters between clouds connected by wide area networks. We describe the design of

Shrinker, its implementation in the KVM hypervisor, and the results of its evaluation on the Grid'5000 testbed.

The fourth part terminates this dissertation with Chapter 7, summarizing our contributions and presenting future work directions.

Part I

Background

Chapter 2

Background

Contents

2.1 Virtualization Technologies 22
2.1.1 Concept 22
2.1.2 Types of Virtualization Technologies 22
2.1.3 Live Virtual Machine Migration 31
2.2 Parallel and Distributed Computing 35
2.2.1 Types of Applications 36
2.2.2 Architectures 39
2.3 Cloud Computing 40
2.3.1 Abstraction Layers 41
2.3.2 Deployment Types 42
2.3.3 Infrastructure as a Service Clouds 42
2.3.4 Open Problems in IaaS Clouds 46
2.4 Federation of Cloud Computing Infrastructures 46
2.4.1 Sky Computing 48
2.4.2 Other Cloud Federation Efforts 49
2.5 Conclusion 49

This chapter describes the scientific and technological background of the contributions presented in this PhD thesis. First, we present virtualization technologies, which provide mechanisms that we leverage to create the contributions of this thesis. Then, we present parallel and distributed applications, which require large quantities of computing power. We also describe the computing infrastructures that are traditionally used to execute these types of applications: clusters and grids. Next, we present cloud computing, which brings together virtualization technologies and distributed systems, allowing to get on-demand access to computing resources. Finally, we present the ongoing efforts towards federating cloud computing infrastructures, which allow to create large scale execution platforms from resources belonging to multiple cloud providers and institutions.

2.1 Virtualization Technologies

In this section, we present the concept of virtualization and its different types of implementations. Then, we describe live virtual machine migration, a mechanism enabled by virtualization allowing to transparently relocate execution environments between different systems. We use live migration as a basic tool for building dynamic execution platforms, as presented in Part III of this dissertation.

2.1.1 Concept

Virtualization [196] refers to the concept of hiding a physical resource behind a virtual resource. Instead of directly accessing the interface of a physical resource, a virtualized system accesses a virtual interface that represents a virtual resource. A virtualization layer translates operations on the virtual interface into operations on the physical resource.

The concept of virtualization is similar to the concept of *abstraction*, but they do not refer to the exact same thing. Abstraction offers access to an underlying system through a simplified interface. For example, a file system provides a simplified interface to a block storage system. Users of a file system do not need to know how the block storage system is implemented. They access it through a higher level abstraction, the file system interface.

All computer systems are implemented as sets of layers. Each layer interacts with the underlying one through a well-defined interface. By using such *abstraction layers*, it is possible to limit the increase in complexity in computing systems. These abstractions made possible the creation of the large number of advanced computing systems we know and use today.

Virtualization differs from abstraction because it offers *different* (virtual) interfaces instead of simpler interfaces. The virtual interface can be identical to the underlying one, or can even be more complicated. For instance, storage virtualization can give access to a file through a hard disk drive interface.

Another concept related to virtualization is *emulation*. Emulation refers to the act of imitating the behavior of an existing hardware or software system on a different platform. For instance, Virtual PC was a popular emulator for Macintosh computers based on PowerPC processors, which emulated an Intel-based computer, allowing to execute a Microsoft Windows operating system. Virtualization can be implemented using emulation. However, virtualization generally refers to a system where most virtual instructions are executed directly on physical hardware, while in emulation, all virtual instructions are transformed through software to be executed on the physical hardware. Virtualization systems can also employ emulation for specific features, such as emulating input/output interfaces.

2.1.2 Types of Virtualization Technologies

Virtualization involves creating an entire virtual execution platform. Even though the focus is usually put on virtualization of CPU resources, a complete execution platform also includes means of communicating with the outside world (for instance via a network) and means of storing data (in memory or on persistent storage). An execution platform can be virtualized by presenting different interfaces, which leads us to classify

virtualization in three different types: hardware virtualization, operating system-level virtualization, and process virtualization.

2.1.2.1 Hardware Virtualization

In hardware virtualization (also known as system-level virtualization), the virtual resource represents a full computer. This virtual computer contains the same hardware components as those found in a real computer: one or several CPUs, some memory, storage devices, network interface controllers, and devices for user interaction such as a keyboard, a mouse, and a screen. However, these components are all virtual. Operations on virtual components are translated into operations on real components of the physical machine: CPU instructions are executed by the physical CPU(s), data is stored in physical memory and on physical storage devices, etc. An instance of such a virtual computer is known as a *hardware virtual machine* or *system virtual machine*.

Since this type of virtualization creates a virtual computer, its usage requires the same software as a real computer. Hardware virtual machines execute operating systems compatible with their virtual hardware: Microsoft Windows or Linux for Intel x86 virtual machines, Solaris for SPARC virtual machines, etc. These systems are known as *guest operating systems*. Figure 2.1 illustrates a simplified view of hardware virtualization. Multiple operating systems are executed on the same physical hardware, in separate virtual machines: Microsoft Windows, Red Hat, and Debian Linux operating systems.

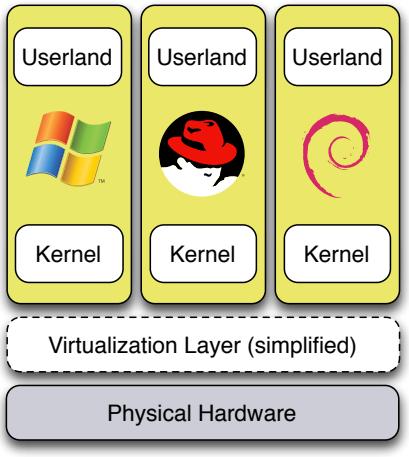


Figure 2.1: Simplified view of hardware virtualization

Hardware virtualization offers a number of advantages:

Binary compatibility Legacy applications and operating systems that do not run on new hardware can be executed on compatible virtualized platforms without modification of their binary code.

Server consolidation Multiplexing resources by executing multiple virtual machines on the same physical machine can reduce the number of servers used in a service hosting environment. Often, each service (DNS, DHCP, etc.) is hosted on a dedicated

physical machine. Instead, multiple services can be *consolidated* onto a reduced number of physical nodes, which leads to a reduction of maintenance and energy cost. Consolidation also allows multitenancy, in which several clients share the same resources.

Isolation Each virtual machine is isolated from the others, which reduces the amount of damage possible from each operating system. If a virtual machine is compromised by an attacker, its ability to perform attacks on other virtual machines is limited compared to a single multitasking operating system where multiple services share the same environment[1].

Migration The complete encapsulation of operating systems in their virtualized execution environment allows to fully capture the state of a virtualized system: the state of the virtual hardware (CPUs, memory, storage, etc.) completely represents the state of the operating system and its applications. This state can be transferred on the network to another physical machine to be resumed on the new hardware. Additionally, live migration performs most of the transfer while the virtual machine is executing, minimizing virtual machine downtime. We describe live migration in more detail in Section 2.1.3.

Replication for fault tolerance With a technique similar to migration, the state of a virtual machine can be replicated and kept synchronized on another physical node. In case of a failure of the primary node, the secondary node can use the replicated state to continue execution of the virtualized system, providing high availability [93].

Snapshots By saving the encapsulated state of a virtual machine to disk, snapshots can be created. They represent the state of a virtual machine at a specific point in time, and can be used to restore a virtual machine to a previous state. By saving only differences between two states (using a technique called copy-on-write), several snapshots can be stored efficiently.

The simplified view of Figure 2.1 does not fully describe the virtualization layer. This software layer, in charge of mapping virtual operations to physical operations, is called a *hypervisor* or *virtual machine monitor (VMM)*. Hypervisors can be implemented in two different ways, leading to two different types: type 1 and type 2 hypervisors.

A type 1 hypervisor runs directly on physical hardware, and is in complete control of it. On top of this hypervisor, several guest operating system can be executed. Often, one guest operating system is running with more privileges in order to be used for the administration of the machine and for controlling the other guests, by providing a user interface to the hypervisor. Figure 2.2 shows an example of a type 1 hypervisor. This hypervisor runs directly on bare hardware. A privileged operating system and two unprivileged guest operating systems are executed on top of a type 1 hypervisor. Examples of type 1 hypervisor are Xen[2] [70] and VMware ESX [212].

[1]However, like in most computer systems, this isolation is not 100 % secure. Implementation bugs do exist, and attackers can exploit them to escape virtual machine isolation.

[2]Xen relies on a specialized guest operating system known as dom0 (a normal guests is known as a domU). This guest is actually in charge of most of the hardware interaction, allowing Xen to remain relatively small compared to an operating system kernel like Linux.

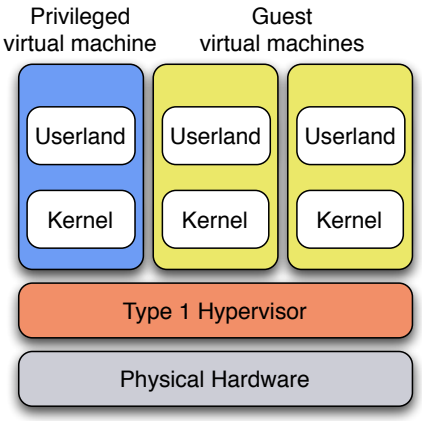


Figure 2.2: Type 1 hypervisor with a privileged and two guest operating systems

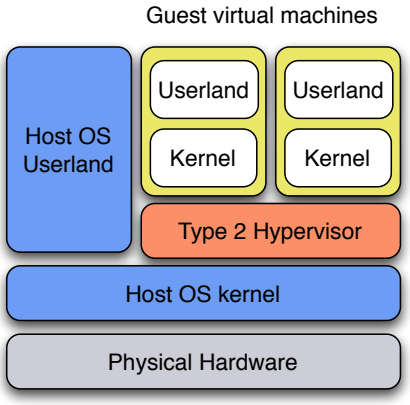


Figure 2.3: Type 2 hypervisor running two guest operating systems

A type 2 hypervisor does not execute directly on bare hardware, but instead on top of an operating system. This operating system, having the role of controlling the hardware, is called the *host operating system*. A type 2 hypervisor does not interact directly with the hardware – instead, it uses the interfaces offered by the host operating system, for example the POSIX interface. Figure 2.3 shows an example of a type 2 hypervisor running on top of a host operating system and executing two guest virtual machines. Examples of type 2 hypervisors are KVM [132], VirtualBox [171], and VMware Workstation [214] [3].

Hardware virtualization is realized using different mechanisms. They can be classified in two main approaches: full virtualization and paravirtualization.

[3]However, since these hypervisors need support from the host operating system to offer good virtualization performance, it can be argued that they are not truly type 2 hypervisors [137]. For instance, KVM relies on modules loaded in the Linux kernel that can interact directly with hardware.

Full Virtualization The first approach, full virtualization, creates virtual machines that are faithful representation of an existing physical platform. An accurate representation of a specific physical platform (e.g. a fully virtualized x86 machine) can execute an unmodified operating system designed for the same platform. Since no modification of the guest operating system is required, this approach allows to execute proprietary operating systems, such as Microsoft Windows.

To use the full virtualization approach, the architecture of the physical computer requires a set of conditions to be met. These requirements were formalized in 1974 by Popek and Goldberg [176], who proposed sufficient conditions for an architecture to be virtualizable. These conditions are based on the behavior of instructions executed on physical hardware. To present these requirements, we define two properties of instructions:

Privileged instruction Privileged instructions are instructions that trap (i.e. trigger a signal that can be caught by the system) when executed in unprivileged mode, and do not trap in privileged mode.

Sensitive instruction Sensitive instructions are divided into two kinds. Control sensitive instructions modify the system state of the machine. Behavior sensitive instructions change their behavior depending on the system state of the machine.

The first theorem of the Popek and Goldberg virtualization requirements states that:

Theorem 1 *For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.*

What this theorem essentially means is that instructions which are related to the system state of the machine need to be privileged. When executed in virtual machines, these instructions can be trapped by hypervisors. Instead, hypervisors execute code that hides the real state of the physical machine and mimics the behavior of the instruction for the virtual machine. This method, a standard way of performing virtualization, is called *trap-and-emulate*.

Popek and Goldberg formalized their virtualization requirements in the 1970s, using a simplified model of computers from *third generation architectures*, such as the IBM 360/67 computer running the CP-67 virtual machine system [107]. Nevertheless, their requirements are still relevant for modern architectures. In 2000, Robin and Irvine studied the architecture of the Intel Pentium[4] to determine whether a secure virtual machine monitor could be implemented on it. They determined that 17 instructions did not meet the virtualization requirements, because they were both sensitive and unprivileged. These *problematic instructions* prevent the x86 architecture from being virtualized using the traditional trap-and-emulate method.

[4]In the rest of this thesis, we will refer to the IA-32 architecture, which the Intel Pentium implements, under the term *x86 architecture*. Technically, x86 refers to the architecture of the entire Intel 8086 family, starting with the 16-bit Intel 8086 microprocessor released in 1978. However, nowadays, x86 is commonly used to refer to more recent extensions of the popular instruction set architecture. When a distinction between the 32-bit instruction set and the more recent 64-bit instruction set is required, we will use the terms *i386* and *x86-64*.

However, it is possible to virtualize the x86 platform by using more advanced techniques than trap-and-emulate. In 1999, VMware released the first version of their x86 hypervisor. Using binary translation, the VMware hypervisor rewrites problematic instructions into instructions that can be virtualized [52]. Since most of the guest code still runs unmodified, the VMware hypervisor showed good performance, and its success helped to renew the popularity of virtualization in the 2000s.

In the mid-2000s, Intel and AMD, the two largest vendors of x86 processors, introduced extensions in order to support virtualization: VT-x from Intel and AMD-V from AMD. These two extensions, developed independently, allow modern x86 systems to be fully virtualized using trap-and-emulate, resulting in simpler hypervisor implementations [210].

Paravirtualization An alternative approach to full virtualization is paravirtualization. In the paravirtualization approach, the guest operating system is modified, making it aware that it is running inside a virtual machine.

This approach brings a number of advantages. First, it is a way of virtualizing a non-virtualizable architecture. By replacing problematic instructions in guest operating systems with instructions that interact with the hypervisor, a paravirtualized operating system can run on a non-virtualizable architecture, such as the x86 without virtualization extensions. Second, this approach allows better performance, for two reasons:

- Interactions between guest operating systems and the hypervisor are usually less expensive than the traditional trap-and-emulate method.

- Because guest operating systems know they are running in virtual machines, it is possible to avoid emulating real I/O devices (e.g. a Realtek 8139 network card), and offer instead a more abstract and more efficient interface.

Xen [70], started as a research project at the University of Cambridge, is a hypervisor that uses the paravirtualization approach[5] to run guest operating systems in virtual machines, which are called *domains*. Guest operating systems are modified to interact with the hypervisor through *hypercalls*. Xen was widely adopted thanks to its low performance overhead and its availability as open source software. However, for many years, running Linux under the Xen hypervisor required a heavily patched version of the kernel, which complicated the integration of Xen in Linux distributions. In July 2011, Linux finally included Xen support. A few other operating systems are supported by Xen in paravirtualized mode, such as NetBSD.

Hybrid Approach: Paravirtualized Drivers An hybrid approach of full virtualization and paravirtualization also exists. It consists in:

- Using full virtualization as the core virtualization mechanism, allowing compatibility with unmodified guest operating systems and simple hypervisor implementations.

[5]Since Xen 3.0, it is possible to run unmodified guest operating systems by taking advantage of the virtualization extensions of Intel and AMD processors. Nonetheless, paravirtualization in Xen has remained popular because of its higher performance.

- Extending the guest operating systems with paravirtualized drivers which are aware that they are running in a virtual machine. Contrarily to normal paravirtualization, this approach does not require modifying the core of the guest operating system. As a consequence, developing paravirtualized drivers is possible on most operating systems, including proprietary ones like Microsoft Windows.

This hybrid approach allows better I/O performance than full virtualization emulating real I/O devices, while maintaining compatibility with proprietary operating systems.

Paravirtualized drivers can also improve usability of virtual machines. Guest operating systems can be made aware that a host operating system is available, and allow features such as copy and paste between host and guests.

This approach is used by many different hypervisors. VMware makes available a set of utilities called VMware Tools. It includes high performance I/O drivers, and tools to improve usability (copy and paste, drag and drop, etc.). The KVM hypervisor features *virtio* [187], a set of paravirtualized interfaces providing high performance network and storage I/O drivers, a balloon driver[6] for managing guest memory allocation, etc.

Hardware Virtualization Performance The performance of hardware virtualization has been a topic of research ever since virtualization regained momentum. Performance comparisons [167, 223, 225] have focused on two aspects:

Comparison of hardware virtualization versus native execution Many comparisons of hardware virtualization platforms versus direct execution on physical hardware have been performed. The general conclusion is that virtualization has a low overhead for CPU-intensive tasks, and a more significant overhead for I/O intensive tasks. This performance loss is decreasing with new advances in virtualization technologies. For some specific use cases, such as high performance computing (HPC), the overhead created by virtualization is still too significant. Furthermore, HPC relies on specific hardware not yet well supported by virtual machines (e.g. high speed interconnects like InfiniBand and Myrinet). However, in general, the distributed computing community has been advocating the use of virtualization technologies since the early 2000s [103].

Comparison of different hardware virtualization technologies With the multiple available virtualization technologies, performance evaluation of different technologies is critical in order to help deciding which technology to choose. The conclusion of these comparisons is that paravirtualization or paravirtualized drivers generally offers the best performance.

2.1.2.2 Operating System-Level Virtualization

An operating system is generally divided in two levels. The lower level is the *kernel*, which manages hardware and shares resources between processes. The upper level is

[6] A balloon driver is a component in charge of balancing memory between the hypervisor and a guest virtual machine. The balloon driver runs in the guest virtual machine. To decrease memory allocation of the virtual machine, the hypervisor requests the balloon driver to allocate memory, which is given back to the hypervisor. This memory can then be used by the host system, or allocated to another virtual machine.

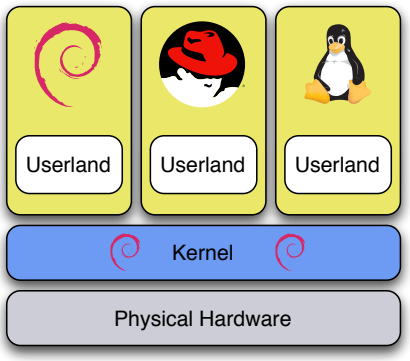


Figure 2.4: Example of operating system-level virtualization

the *user-space* or *userland*, where processes are executed. Operating system-level virtualization [198] refers to the virtualization of the kernel interface of an operating system. Multiple user-space domains can run in parallel on the same system and share the same kernel. However, each domain is isolated from the others, each one believing it is the only userland running on the machine. This type of virtualization is referred to with different terms: containers for OpenVZ [36], jails for FreeBSD [14], zones for Solaris [177], etc.

The ancestor of operating system-level virtualization is the *chroot* [200] mechanism. *chroot* allows a set of processes to be bound to a subdirectory of the file system. These processes see this subdirectory (usually containing a complete operating system installation) as their root file system. However, *chroot* only provides virtualization of the file system. Processes in different *chroot* share the same process space, use the same network interfaces (hence the same IP addresses), etc.

Contrarily to *chroot*, operating system-level virtualization mechanisms isolate user-space domains for all resources offered by the operating system kernel: file system, memory, CPU, network.

Figure 2.4 shows an example of operating system-level virtualization. The physical node is running a Linux kernel of the Debian distribution. Three user-space domains share this kernel for their execution: a Debian userland, a Red Hat userland, and a generic Linux distribution.

In this kind of virtualization, the user-space domains needs to be compatible with the underlying kernel. For instance, it is not possible to execute a Microsoft Windows userland on top of a Linux kernel.

However, this approach offers some advantages over hardware virtualization. Since the kernel is already running when a new user-space domain is started, there is no lengthy boot phase: only some new resources need to be allocated for the new domain, and a new init process is started[7]. This results in a short start procedure, in the order of a few seconds rather than a minute or more for a hardware virtual machine. Resource consumption is also reduced compared to hardware virtualization. Since the kernel is shared, there is no need to have a full kernel loaded in each virtual machine. Several

[7] A new user-space domain can also be started with only a shell instead of using a regular init process. This makes the start process even faster, but no service daemon is available in the domain.

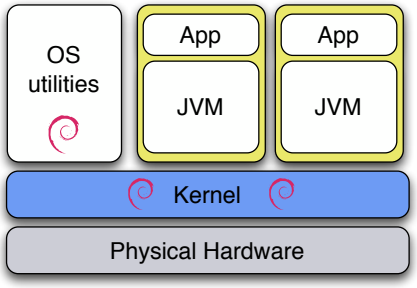


Figure 2.5: Example of process-level virtualization

user-space domains can share the same file system (while writing modified data in a copy-on-write fashion), and share common data in the same buffer cache. It is also easier to change memory allocation between domains without relying on tools like a ballooning driver (see page 28). Finally, there is no performance overhead, since the userland applications and the kernel are running directly on physical hardware.

2.1.2.3 Process-Level Virtualization

In contrast to hardware virtualization, process-level virtualization creates execution platforms that directly run applications instead of guest operating systems. These applications, built specifically for the interface provided by a specific process-level virtual machine technology, run as single processes in instances of these virtual machines.

The most popular example of a process-level virtual machine is the Java Virtual Machine (JVM). The JVM executes applications consisting of Java bytecode, instead of executables with machine code targeting the instruction set of a particular processor architecture. This bytecode is portable, in the sense that it can be executed on any JVM implementation. JVMs are available on many platforms, from computers running different operating systems to embedded systems such as cellphones. Before executing bytecode, the JVM first verifies it. For example, the JVM verifies the validity of branch instructions targets, helping to prevent attackers from executing arbitrary code. This makes a JVM application more secure than a regular application compiled to machine code (e.g. a C application).

In order to be executed on a physical platform, the Java bytecode needs to be transformed into native instructions. The execution can be performed by interpreting the bytecode, but this approach can offer poor performance. Currently, the prevalent approach is to use just-in-time compilation (JIT). Using a JIT compiler, the JVM can turn bytecode into native machine code dynamically, right before its execution. When doing repeated executions of the same code, a JIT approach, which will compile to machine code once per execution, is more efficient than interpreting the same code over and over.

Figure 2.5 shows an example of process-level virtualization. The physical node is running a Linux kernel of the Debian distribution, and its accompanying OS utilities and services. Two Java Virtual Machines are running in userland, each executing an application consisting of Java bytecode.

2.1.3 Live Virtual Machine Migration

In this section, we present live migration of hardware virtual machines, which is a major mechanism allowing execution platforms to be more dynamic. We describe the techniques used to perform live migration. We also discuss approaches that make live migration possible in wide area networks.

Virtual machine migration is the act of moving a virtual machine from a physical node to another one. It offers numerous advantages for managing computing infrastructures, including:

Load balancing Virtual machines can be dynamically migrated depending on their workload, to offer more efficient usage of computing resources.

Reduced energy consumption Virtual machines with low workloads can be consolidated to fewer physical machines, making it possible to power off nodes to reduce energy consumption. For instance, Entropy [117] is a resource manager that uses live migration to dynamically consolidate virtual machines executed on a homogeneous cluster. It uses constraint programming to optimize the placement of virtual machines while respecting performance constraints and reducing the number of running physical nodes.

Transparent infrastructure maintenance Before physical machines are shut down for maintenance, administrators can relocate virtual machines to other nodes to avoid interrupting users.

Improved fault tolerance By monitoring physical hardware for signs of imminent failures, live migration can be pro-actively triggered to relocate virtual machines on healthy nodes [100, 157].

2.1.3.1 Origins

Before virtualization regained popularity in the 2000s, migration of processes (instances of computer programs being executed) had been a hot research topic. Starting from the late 1970s, process migration implementations have been proposed for a large variety of operating systems. However, mainstream operating systems never adopted process migration. Several reasons explain the lack of popularity of process migration [153]. First, since processes are tightly coupled with their underlying operating system, internal resources of the operating system kernel need to be available on the destination system after migration: sockets, pipes, memory segments, open files, etc. To solve this problem, some implementations forward operations to the source node to access these resources [69]. This requires the source node to remain in operation for migrated processes to continue running. This reliance on the availability of the source system is called having *residual dependencies*. Second, process migration was not adopted by major operating systems vendors because there was no strong commercial incentive for it.

Contrarily to processes that do not encapsulate all their state, system-level virtual machines include userland processes together with their underlying operating system. As such, all system resources used by processes are included in a virtual machine. This property makes it possible to migrate a virtual machine from one node to another without the residual dependencies of process migration.

Originally, migration of virtual machines was done with a simple *stop-and-copy* mechanism, which works as follows. First, the virtual machine is suspended: its non-persistent state (CPU registers, hardware device registers and memory content) is written on disk. Then, this suspended state is transferred to another node via the network (this is usually done manually by the administrator of the machines). Finally, the virtual machine is restarted from its suspended state on the destination node. The transfer of the virtual machine state can include the persistent storage of the virtual machine, unless it is stored on a shared file system accessible by the source and destination nodes.

This method of migrating virtual machines has a major drawback. The time to migrate a virtual machine is proportional to its size, and inverse proportional to the available network bandwidth. Thus, migration in wide area networks can take a long time because of the low available network bandwidth. During this transfer, the virtual machine cannot be used, since it is in a suspended state.

Even with this downside, migration is interesting in some contexts. The Internet Suspend/Resume project [133] uses migration to allow mobile computing without transporting physical hardware. For instance, the desktop environment of a user would be transferred between her home and her work place during her commute.

However, long downtime kept migration from entering data center infrastructures, which are generally hosting user services that cannot tolerate long interruptions.

2.1.3.2 Concept

Live virtual machine migration [87, 163] refers to a more advanced type of migration than stop-and-copy. In a live migration, the virtual machine is kept running in parallel with its state transfer. Live migration has first been implemented with a pre-copy [203] algorithm (also used in process migration [153]). Virtual machine memory is copied from the memory of the source host to the memory of the destination host, while the virtual machine is still executing. Live migration continues sending modified memory pages until it enters a last phase where the virtual machine is paused. Then, remaining modified pages are copied, along with the state of the CPU registers and hardware device registers. Finally, the virtual machine is resumed on the destination host. This last phase is responsible for the downtime experienced during live migration, which should be kept minimal.

This downtime can range from a few milliseconds to seconds or minutes, depending on page dirtying rate and network bandwidth [54]. The condition to enter this last phase depends on the hypervisor. Xen switches to this last phase when the number of remaining memory pages goes under a threshold, when too many iterations have been executed, or when too much data has been transferred. KVM computes an expected downtime based on the number of remaining pages and available network bandwidth, and finishes migration only if the expected downtime is less than the maximum downtime set by users or virtual machine management frameworks. As a consequence, a live migration in KVM can continue indefinitely if the amount of memory modified at each iteration is too large to be sent without violating the maximum downtime. In this case, users or virtual machine management frameworks are expected to increase the maximum downtime to complete live migration. The persistent state of the virtual machine (its storage devices) is assumed to be accessible by the source and destination hosts through a shared file system (e.g. NFS).

2.1.3.3 Alternative Live Migration Algorithms

In the previous section, we presented live migration and its main implementation using the pre-copy algorithm. In this section, we describe two alternative algorithms that have been proposed to improve the efficiency of live migration compared to pre-copy.

Post-Copy An alternative to pre-copy is live migration based on post-copy [118, 119], which transfers only CPU state before resuming a virtual machine on the destination host. Memory pages are fetched from the source host on demand, in addition to a background copying process to decrease the total migration time and quickly remove the residual dependency on the source host. Similarly, SnowFlock [134, 135] uses demand-paging and multicast distribution of data to quickly instantiate virtual machine clones on multiple hosts.

Trace and Replay Another live migration algorithm was proposed by Liu et al. [139], based on checkpointing/recovery and trace/replay. Initially, the virtual machine is checkpointed (i.e. snapshotted) to a file. This checkpoint file is transferred to the destination node while the virtual machine continues its execution. When the destination node finishes receiving the checkpoint file, it is in possession of the complete state of the virtual machine (CPU state, memory and disk content, etc.). However, this state is outdated, since the virtual machine has continued its execution on the source node in the meantime. To synchronize the state on the destination node with the state on the source node, this algorithm logs all non-deterministic events (time-related events and external input) occurring on the virtual machine in a log file. This log file is transferred to the destination node and is replayed in the virtual machine created from the checkpoint. Typically, a log file is much smaller than the equivalent state changes (modified memory pages and disk blocks), and can generally be replayed faster than on the source host by skipping idle time. Several rounds of trace/replay are performed, until the remaining log is small enough to be quickly replayed. When this happens, the virtual machine is stopped on the source node, the last log is transferred, and it is replayed on the destination node. By reducing the size of the data to send at each iteration, this algorithm improves live migration efficiency (reduced downtime, total migration time, and total data transferred). However, this approach is not adapted to migrate multiprocessor virtual machines, as a large number of memory races (which are non-deterministic events) between processes need to be logged and replayed.

2.1.3.4 Wide Area Live Migration of Virtual Machines

Previous sections focused on live migration in the context of local area networks, where live migration mechanisms rely on two requirements to operate:

Availability of shared storage Virtual machine storage needs to be provided through a shared file system, in order to be accessible from both source and destination nodes. This allows live migration to operate without transferring virtual storage state, which can be several orders of magnitude larger than memory and require a long transfer time.

Migration within the same local area network When a virtual machine is migrated, all its network state is transferred. This includes the status of the TCP stack and IP addresses assigned to the virtual machine. For communications to remain uninterrupted after migration, live migration mechanisms generally require that virtual machines migrate within the same local area network. This allows network traffic to continue to be routed to the correct subnetwork. Network traffic sent to the virtual machine is redirected to the destination node by sending unsolicited ARP or RARP replies from the destination hypervisor. This reconfigures the switch of the local area network to send network packets to the destination machine.

In the context of wide area networks, these two requirements are usually not obtainable. First, most virtualized infrastructures connected through wide area networks belong to different administration domains. Thus, they often cannot share the same file system, for technical and/or administrative reasons. Furthermore, most shared file system implementations are designed for being operated in cluster environments and do not tolerate the high latencies present on wide area networks. Second, migrating through a wide area network will cause a change of local area network, and prevent traffic from being correctly sent to the virtual machine after migration. Additionally, the migrated virtual machine would be using an invalid IP address in the destination local network.

Wide area live migration could bring further benefits to the management of virtualized infrastructures. For example, a maintenance on a whole data center could be made transparent by migrating virtual machines to a different data center. To make wide area live migration possible, solutions have been proposed to overcome these two issues: requirement for shared storage and restriction to local area networks.

Solving the requirement for shared storage There are two approaches to solve the requirement for shared storage. The first approach is to make shared storage available beyond the boundaries of a local area network, so that the virtual machine storage is accessible from source and destination nodes belonging to different networks. This can be done using grid file systems like XtreemFS [122]. However, this approach requires coordination between the administrative entities of the different virtualized infrastructures, which may not always be possible.

Alternatively, storage can be migrated directly from the source node to the destination node with algorithms similar to live migration of the memory state [74, 120, 121, 144, 221]. Storage state can be iteratively transferred to the destination node (like in the pre-copy algorithm) or copied on-demand (like in the post-copy algorithm). These research contributions are either using full implementations or leveraging existing storage synchronization systems like DRBD [10]. Additionally to these systems, implementations of storage migration are available in the KVM hypervisor [158] (since version 0.12) and in VMware ESX [148]. KVM uses a pre-copy algorithm similar to memory migration. Multiple algorithms have been used in different versions of VMware ESX. VMware ESX 3.5 uses a storage migration method based on snapshots. VMware ESX 4.0 uses an iterative copy with a Dirty Block Tracking (DBT) mechanism. Finally, VMware ESX 5.0 leverages synchronous I/O mirroring.

Solving the restriction to local area networks Several approaches exist to achieve the goal of keeping communications uninterrupted after a live migration across two different local area networks. A first approach establishes a tunnel between the source node and destination node to forward network traffic to the migrated virtual machine [74]. It assumes that the virtual machine changes IP address after being migrated, and uses dynamic DNS to make the virtual machine host name point to the new IP address. The tunnel can be destroyed once all connections have stopped using the old IP address. Another approach based on tunnels creates them between nodes which want to communicate with virtual machines (client nodes) and hypervisors [206]. After migration, tunnel endpoints are reconfigured to contact the destination node. A downside of this approach is that it requires collaboration on the client side to work, instead of being fully managed by the network infrastructure.

Another approach establishes layer-2 VPN between different local area networks. This allows them to be treated as a single network. VPN reconfiguration can establish these VPN connections dynamically when the first wide area live migration occurs [221].

Lastly, Mobile IP [173, 174] is a standard solution to provide machine mobility, which can be applied to virtual machine migration [115]. Connections to the virtual machine are first routed through an entity of the source network called the *home agent* which forwards them to the current address of the virtual machine (known as the *care-of address*). The home agent is required to stay reachable in order to continue forwarding traffic to the migrated virtual machine.

Since we study this particular problem in Chapter 5, we further describe existing solutions in Section 5.3 of this chapter.

2.2 Parallel and Distributed Computing

In this section, we present two approaches, *parallel computing* and *distributed computing*, that enable to solve the very large problems of scientific computing, data mining, etc. Instead of using the limited capabilities of a single CPU to solve a problem, these approaches assign many CPUs to the same problem. They divide the work so that each CPU solves only a fraction of it, which speeds up the computation. The speedup depends on the fraction of parallel versus sequential parts of the program, and is modeled by Amdahl's law [64].

A large number of systems have been designed to solve problems using parallel or distributed computing approaches. The simplest architecture consists of a single machine with several CPUs connected to the same memory (symmetric multiprocessing). Nowadays, the trend is towards multi-core architectures, where a single physical processor contains many CPU units sharing some elements of the processor, for instance the L3 cache. Since the number of processors that can share the same memory is limited, parallel computers linked through high performance interconnects have been developed. These computers are *massively parallel processors* (MPPs). Examples of MPPs include the supercomputers implementing the Blue Gene architecture [53].

An alternative to the MMP approach is clusters. Clusters are created from regular, off-the-shelf machines, connected together through a network. Although it is possible to build clusters with TCP/IP over Ethernet (the same technology than personal computers of the consumer market), high speed interconnects such as InfiniBand or Myrinet offer

better performance because of their lower latency and higher bandwidth. Clusters now form the majority of the 500 most powerful computers in the world [43].

Finally, geographically distributed resources belonging to different organizations can be federated together in grids, in order to bring massive amounts of computing power to users.

The distinction between parallel computing and distributed computing is not straightforward. Parallel programs cover a wide range of problems. Some of these problems, such as computational fluid dynamics, generally require highly coupled execution threads with frequent data exchange. This type of programs can only achieve good performance on architectures with low latency and high communication throughput between processors: SMPs, MPPs and clusters (possibly with high speed interconnects). Contrarily, *embarrassingly parallel* problems can be divided into partially or completely independent subproblems, which do not have requirements on communication performance. In this case, distributed computing can be used to execute computations on resources distributed all over the world in different sites and belonging to different organizations or individuals.

2.2.1 Types of Applications

Parallel and distributed applications can be differentiated and classified through many properties. In this section, we distinguish applications by two properties:

Dynamic capabilities Whether the application can be adapted during runtime to change (increase or decrease) the number of resources assigned to a computation.

Programming models How the different parts of the application communicate and synchronize with each other: message passing, distributed shared memory, or data oriented models like MapReduce.

2.2.1.1 Dynamic Capabilities

Static applications Static applications run on a fixed set of resources, which is allocated at the beginning of the computation and cannot change over time. Typically, the problem is divided in as many tasks as there are resources involved. We distinguish two types of static applications: *rigid* and *moldable* [101]. Rigid applications execute on a number of resources that is hardcoded or selected by the user, while moldable applications run on a number of resources chosen by the resource manager to optimize the usage of the infrastructure.

Dynamic Applications Dynamic applications run on a set of resources that can change during the execution. We distinguish two types of dynamic applications: *malleable* and *evolving* [101]. Malleable applications can grow or shrink depending on resource availability. When more resources become available, a malleable application may grow to use these resources and finish its execution faster. If resources become unavailable, a malleable application can shrink (to some extent) to continue running in the decreased set of resources. Evolving applications present variable resource requirements during their execution. This can be caused by computations changing in complexity, by containing

multiple phases with different resource requirements, or by user actions modifying application parameters. Some applications can be both malleable and evolving. Malleable applications include Bag of Tasks [86] (BoT) and Parameter Sweep Applications [50] (PSA). Bag of Tasks solve problems where many independent tasks have to be computed. These independent tasks correspond to computations on different data, which can be stored in different input files. Parameter sweep applications analyze the same data with different parameters.

2.2.1.2 Programming Models

To take advantage of multiple computing resources, programs have to be designed to work in a parallel or distributed fashion. Several approaches can be used to parallelize computations. In this section, we describe different approaches: shared memory, message passing, remote procedure calls, and the MapReduce programming model.

Shared Memory Using a shared memory approach, multiple threads of execution can access the same memory region and perform read and write operations on it in parallel. This allows different threads to operate on the same data without relying on communication procedures. This type of parallel programming can be very efficient as threads access data directly, without going through several layers of code to handle inter-process communications. However, execution safety can be compromised: since memory is shared between all threads, a bug in one thread can corrupt memory for all others. Furthermore, synchronization between threads is generally performed using locks, which can lead to livelocks or deadlocks if programmed incorrectly.

The multiple threads involved in a shared memory parallel program can be executed in the same process. In this case, memory sharing is automatically available. Creation of these threads can be performed manually by the programmer, for instance using POSIX threads [38] (the *Pthreads* library), or with higher level paradigms such as OpenMP [32] or Intel Threading Building Blocks [123] (Intel TBB). It is also possible to access shared memory from threads executing in different processes. In this case, threads interact with the operating system kernel to set up shared memory segments (e.g. System V shared memory on Linux). The operating system kernel configures virtual memory to make the same memory region transparently available to all processes.

Distributed shared memory systems [136] implement shared memory across multiple computers. These systems offer the illusion that all computers are sharing the same large memory space, while in reality each node only has direct access to its own local memory. When an operation needs data residing on a remote node, this data is migrated over the network to perform the operation locally. Since caching mechanisms are used to improve efficiency of these systems, a coherence protocol is used to maintain consistency in the presence of concurrent read and write operations.

Distributed shared memory allows programs with large data sets to be executed in parallel on many machines, with few or no modification to their code. The performance of these mechanisms depend on the programmer's ability to efficiently use the caching mechanisms provided by the system.

Message Passing In the message passing model, processes do not share the same memory space. Instead, the programmer explicitly describes how the multiple processes

of her application send and receive messages. This can be performed through a library, for instance using an implementation of the Message Passing Interface (MPI) [197] such as MPICH2 [26] or Open MPI [30]. This model is generally used to exchange data in parallel systems.

Hybrid systems combining shared memory (for sharing data between threads on the same machine) and message passing (for sharing data between threads on different machines) can also be used, for instance by combining OpenMP and MPI [80].

Remote procedure calls Remote procedure calls [72] (RPC) allow to transfer the execution to another process by invoking functions of a remote process. This model is implemented in many programming languages: RMI [22] in Java, DRb [11] for Ruby, etc. Some implementations are portable among different languages by relying on a standardized format (e.g. XML-RPC [47]). This model is generally used in distributed systems.

MapReduce MapReduce [96, 97] is both a programming model and a framework for distributed processing of large amounts of data. The design of MapReduce was introduced by Google in 2004, after having been used internally for five years. It is inspired by the *map* and *reduce* functions found in functional programming languages.

Users provide a map and a reduce function. Input data (a set of key/value pairs) is split into chunks, and each chunk is fed to the map function (executed on many nodes in parallel), which produces intermediate key/value pairs. The intermediate key/value pairs are merged and sorted, and fed to the reduce function (also executed in parallel) which produces the result of the computation. This programming model became popular because it is simple yet expressive enough to perform a large variety of data-intensive computing tasks. This programming model is backed by a proprietary framework developed by Google that takes care of scheduling tasks to workers, sharing data through a distributed file system, handling faults, etc.

Although Google's implementation of MapReduce is proprietary, its design inspired the creation of the Hadoop project [1], an open source framework for distributed processing of large data sets. Hadoop is used extensively by companies such as Yahoo!, Facebook and eBay to perform thousands of computations per day over petabytes of data [194, 204]. The two main components of Hadoop are the Hadoop Distributed File System (HDFS) and Hadoop MapReduce.

HDFS [193] is a distributed file system created by the Hadoop project. The design of HDFS is similar to the architecture of the Google File System [106]. It is designed to be highly-reliable (using replication) and to provide high throughput for large data sets used by applications. For performance reasons, it relaxes some POSIX semantics: files can only be written once, in order to simplify coherency between nodes. It is designed to scale to thousands of nodes to create multi-petabyte file systems. Another feature of HDFS is rack awareness. Using its knowledge of what nodes are stored in what rack, HDFS is able to place replicas of data so that the failure of a rack does not put all replicas offline. The architecture of HDFS contains two types of services. A centralized NameNode daemon acts as a metadata server, tracking presence of physical nodes and the positions of data blocks inside the cluster. Each node participating in the HDFS cluster runs a DataNode daemon, storing blocks and managing read and write operations on files.

The MapReduce implementation of the Hadoop project is designed to work together with HDFS. Tasks are preferably scheduled on nodes storing the data they require, moving the computation to the data (which prevents data transfer from impairing performance). This is done using the locality information kept in HDFS. The architecture of Hadoop MapReduce consists of a centralized service, the JobTracker. The JobTracker receives job submissions, divides each job in a number of map and reduce tasks, and schedules these tasks on compute nodes. Each compute node runs a TaskTracker daemon, which interacts with the JobTracker to receive tasks to be executed.

2.2.2 Architectures

In this section, we present in more details the major architectures that have been used to execute parallel and distributed applications and led to the creation of cloud computing: clusters and grids.

2.2.2.1 Clusters

A cluster is a set of physical nodes connected through a local area network. The local area network is used for communication between the processes involved in a parallel or distributed computation. This network can be a typical Gigabit Ethernet network, or a high speed interconnect such as InfiniBand or Myrinet.

Clusters are controlled by resource managers (e.g. TORQUE [44]), software that allocate jobs to compute nodes, track their status, and share the infrastructure between cluster users. Resource managers contain queues to store jobs waiting for execution, and schedule jobs on available resources according to constrains set by users and/or administrators.

Another approach is to build resource management directly in the operating system. Single system image [143] (SSI) gives the illusion of a cluster being a very large SMP machine. This allows to execute and manage jobs exactly like on a single computer, using standard operating system tools. Examples of SSI are openMosix [31], OpenSSI [34] and Kerrighed [156].

2.2.2.2 Computing Grids

Grid computing, proposed by Ian Foster and Carl Kesselman [104] in the 1990s, thrives to provide access to computing resources in a transparent way, similarly to how power is delivered through the electricity grid. To achieve this goal, architectures known as computing grids are created by federating computing resources from several different organizations. Grids can be made of different types of resources. Typically, clusters of compute nodes are connected to the Internet and used to create grids. Desktop grids are another type of grids, which take advantage of users workstations. These desktop resources can belong to the organizations creating the grid, or to volunteers connected to the Internet who choose to help scientists solve problems (e.g. by participating in projects of the BOINC platform [65]).

Virtual organizations are a central notion in the management of grid computing infrastructures. Virtual organizations are groups of individuals or organizations sharing resources to achieve common goals. They are used to regulate resource sharing in a grid.

Computing grids are managed by software providing services for discovering resources, submitting and monitoring jobs, storing job input and output data, etc. Over time, several of these services were standardized by the Open Grid Forum [29] (OGF).

Some very large scale grids have been created. For instance, the Worldwide LHC Computing Grid [46], created to process the extensive amounts of data generated by Large Hadron Collider at CERN, consists of about 200,000 processing cores and 150 petabytes of disk space, distributed across 34 countries.

The main approach used to implement grids has been through middleware. A middleware is an extra layer of software that sits between applications and operating systems and provides services to applications[8]. The Globus Toolkit [18] and gLite [16] are two middleware implementing a large collection of grid services.

Similarly to a single system image (see Section 2.2.2.1), support for a grid architecture can be built directly into an operating system. This approach was initiated by the Legion system [110], providing the illusion of a desktop computer through which users can manipulate objects stored among million of hosts connected by wide area networks. Developed in the framework of the XtreemOS European project [48, 92], XtreemOS is a grid operating system built on Linux and supporting virtual organizations natively. Users can discover resources and interact with them using regular Unix tools. Additionally, several grid standards are supported: jobs can be described in JSDL (an OGF standard), and an implementation of the SAGA API [40] is available.

2.3 Cloud Computing

Parallel and distributed computing have allowed to tackle more and more complex problems. With architectures such as clusters and grids, users leverage large numbers of computing resources. Although the vision of a global worldwide grid – *the Grid* – delivering computing power to anyone in a totally transparent manner was not achieved, many grid systems have been created and successfully used in scientific fields. More recently, in the second part of the 2000s, another concept allowing to remotely access computing resources emerged: *cloud computing*.

Cloud computing [67, 211] is a computing paradigm in which resources are made available through a computer network (e.g. Internet) in an on-demand fashion. Cloud computing provides elastic provisioning of resources (users can modify their resource requests at any time) with a pay as you go model (users are charged only for the resources they consume). Users have low or no visibility on how and where their applications are executed: a cloud is shown as an infinite pool of computing resources.

While cloud computing shares some concepts with grid computing, it can be seen as a more practical approach. It does not try to federate resources in a transparent manner. Instead, cloud computing is made available by companies called *cloud providers*, each provider proposing its own independent service.

Cloud computing has seen a tremendous growth in the second part of the 2000s, and many types of services have been marketed as cloud services. There is a general agreement on a classification of cloud computing in three different layers [49]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service

[8] A middleware can be seen as similar to libraries, but at a much larger scale, and involving many components.

(SaaS). However, not all services can be easily restricted to one category.

2.3.1 Abstraction Layers

In this section, we describe the properties of the three different cloud computing abstraction layers recognized by the community.

2.3.1.1 Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) delivers computing resources in the form of virtual machines to users in an on-demand and elastic fashion. IaaS users can build execution platforms from these resources. For instance, a web hosting company can build its infrastructure from IaaS resources, by provisioning virtual machines with their desired operating systems, deploying software (load balancers, web servers, application servers, databases, proxies), persisting data in cloud storage, etc. In this scenario, the main advantages of IaaS are the elasticity and the pay as you go model: users can scale their infrastructure according to their needs, dynamically resize the infrastructure depending on the load, and pay only for resources that are actually used. This is in contrast to traditional systems where users have to size their infrastructure according to their peak load, leading to wasted resources and increased expenses. IaaS allows users to have a high degree of control over their infrastructure. They manage virtualized operating systems with administrative rights and can customize execution environments as required, which eases the migration of existing applications and systems to IaaS clouds. Examples of IaaS services are Amazon EC2 [57] and Rackspace Cloud [39] on the commercial side, and Science Clouds [131] on the community side. Additionally, IaaS cloud services can be created with several open source toolkits, described in Section 2.3.3.

2.3.1.2 Platform as a Service (PaaS)

Platform as a Service (PaaS) provides a higher layer of abstraction than IaaS. Instead of offering raw resources with full administrative control, PaaS gives access to environments specifically designed to execute particular types of tasks (e.g. web service hosting). Users write their applications for a specific platform, with high level tools to ease programming, deployment and monitoring. The platform is in charge of allocating resources to applications and scaling them according to user demand. PaaS does not necessarily uses virtualization, since applications are isolated directly at the platform layer. The origins of PaaS can be traced back to shared web hosting service that have been available since the emergence of the web. Examples of PaaS are Google App Engine [109] (for web applications written in Java or Python) and Heroku [20] (for web applications written in Ruby, Node.js, Clojure, Java, Python, or Scala).

2.3.1.3 Software as a Service (SaaS)

Software as a Service (SaaS) refers to end-user software provided in an on-demand manner over the Internet. Instead of acquiring software licenses for installation on a local platform, users can directly access software hosted by cloud providers. Generally, this software is provided in the form of web applications accessible with web browsers. The main advantage of this model is that no installation or maintenance is required for

SaaS users. It also allows SaaS providers to update their software much more frequently, for example multiple times per day. Examples of SaaS are Google Apps like Gmail [108], and business applications from Salesforce.com [41].

2.3.2 Deployment Types

Cloud services also differ in how their deployments are made available to users. According to this criteria, clouds are classified in three types: public clouds, private clouds, and community clouds.

Public cloud A public cloud is available to the general public, whether they are individuals or organizations. The main requirement to access a public cloud is to provide a payment method (e.g. a credit card number) to be billed for resource usage. This is the model followed by commercial cloud providers like Amazon EC2 and Rackspace.

Private cloud A private cloud is a cloud infrastructure managed and used exclusively by a single organization (e.g. a company or a laboratory). Before the emergence of cloud computing, many organizations already used virtualization in order to manage their internal computing infrastructures. These infrastructures could already be considered as private clouds. There is generally no billing involved in these systems.

Community cloud A community cloud is a middle approach between a public and a private cloud: it is restricted to a limited set of users belonging to specific organizations, or sharing a common goal. For instance, scientific clouds [131, 179] provide cloud computing resources to scientists in order to experiment with clouds. This model shows similarities with grids, in which multiple organizations share resources to reach a common goal.

2.3.3 Infrastructure as a Service Clouds

Of all the cloud abstraction levels, Infrastructure as a Service offers the most control over resources. As such, it is particularly suited to create execution platforms for solving specific problems. For instance, existing scientific applications such as embarrassingly parallel programs can be deployed in these execution platforms. Instead, a PaaS approach would require applications to be adapted to use the PaaS interface. Consequently, we focus on IaaS clouds in the rest of this dissertation.

IaaS became mainstream in the commercial world when Amazon launched its EC2 service in summer 2006. However, prior to this date, multiple research projects from institutions around the world had already started tackling the problem of providing on-demand virtualized resources to users. One example is the In-VIGO system [51] for constructing virtual grids to support execution of scientific applications. Over time, several open source software stacks for cloud computing were created. The most popular are Eucalyptus [13, 166], Nimbus [27, 128], OpenNebula [33, 199], and OpenStack [35] (originating from the NASA Nebula cloud computing platform [159] and now backed by a large number of commercial companies). In this section, we present the general architecture of an IaaS cloud, and highlight how these projects have chosen to implement specific features. Other comparisons are available in the literature [84, 168, 172, 192].

The purpose of an IaaS cloud is to allow its users to build execution platforms according to their computing needs, using virtualized resources. Users interact with an IaaS service through a web service protocol, such as REST [102], SOAP or XML-RPC. This can be done using a command line program or a web interface, or by contacting directly the service API through a library. In the case of Amazon EC2, these three choices are available: respectively, the Amazon EC2 API Tools, the AWS Management Console, and the AWS SDKs available for multiple programming languages.

The actions that can be performed on an IaaS cloud allow to control the life cycle of virtual machines:

Upload virtual machine images Although libraries of virtual machines images (containing generic operating system or environments customized for specific tasks) are commonly available in clouds, users can also upload their own virtual machine images to create customized execution platforms.

Start virtual machines Users start virtual machines by selecting a virtual machine image and a number of instances to start (which allows to start a virtual cluster with only one service interaction).

Terminate virtual machines When virtual machines are not needed anymore in the infrastructure, users can terminate them. Since users are charged for the running time of their virtual machines, terminating unused virtual machines is important to reduce the cost of operating execution platforms.

Save virtual machine states to images After customizing virtual machines running in an IaaS cloud, users are able to save the content of their virtual disks as virtual machine images. These images can then be used to provision new virtual machines with the customized environments.

Most of these interactions involve virtual machines images. When starting a new virtual machine, a virtual machine image (i.e. a file containing an operating system installation) must be made available for the virtual machine. When saving the state of a virtual machine, the content of its disk needs to be copied to a virtual machine image (by overwriting an existing image or creating a new one). To store these virtual machine images, IaaS clouds all provide image repositories. For instance, virtual machines images for Amazon EC2, called Amazon Machine Images (AMIs), are stored in Amazon S3 [59], a storage service part of the Amazon Web Services (AWS) platform.

The generic architecture of an IaaS cloud is represented on Figure 2.6. We now describe in more detail each component of this architecture.

2.3.3.1 IaaS Service

The first component of an IaaS architecture is the service endpoint. An IaaS service acts as a front-end to the cloud, with which users interact to perform IaaS requests. In addition to performing actions triggered by user requests, this front-end authenticates users, manages accounting information, and in the case of a commercial cloud, manages billing information. It also contains a scheduler that decides how to share the physical infrastructure between users.

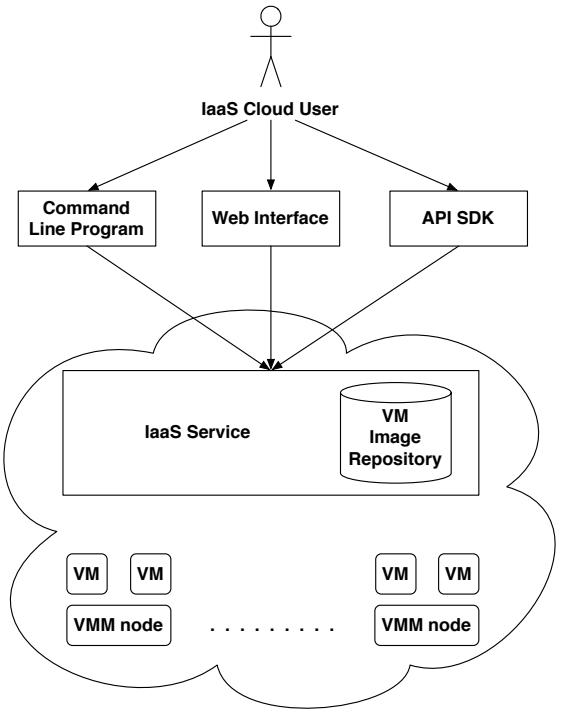


Figure 2.6: Generic architecture of an IaaS cloud

Most IaaS services provide programmable APIs. Amazon EC2 offers two different APIs: the first is based on SOAP (Simple Object Access Protocol), the second is a simpler HTTP-based API called Query[9]. Because of the large market share of Amazon EC2 in IaaS clouds, these interfaces have become de-facto standards: Eucalyptus, Nimbus, OpenNebula, and OpenStack all provide some support of EC2 APIs (however, their support is limited to the subset of EC2 features they each implement). In parallel, the Open Grid Forum has started an initiative to create a set of standards for cloud services: the Open Cloud Computing Interface [28] (OCCI). Although the project is still young, it has started to get adoption with support being implemented in OpenNebula and OpenStack.

After receiving a user request, accepting it (by verifying the user identify and its rights to perform the actions), and taking a scheduling decision if need be, the service triggers actions on the physical nodes hosting the virtual machines. For instance, if a user asks for termination of one of her virtual machines, the service will contact the physical host in order to shutdown the virtual machine and release allocated resources (CPU, memory, disk).

2.3.3.2 Virtual Machine Image Repository

The second component of an IaaS architecture is the virtual machine image repository. All virtual machines in an IaaS cloud need to be based on virtual machine images.

[9]Note that while the EC2 Query API is simpler than the SOAP interface, it does not follow a REST model.

A virtual machine image contains the operating system of a virtual machine, and is needed for the whole life time of a virtual machine. At any time when a virtual machine is running, its operating system or applications can write on disk. This requires each virtual machine to have its own private copy of its virtual machine image.

A common architectural pattern in an IaaS cloud is to include a virtual machine image repository. This repository contains virtual machine images, either provided by the cloud administrator staff or uploaded by users. These repositories usually support private and public images: a private image belongs to a user and is accessible only by her, while a public image is available for all users. Prior to starting a virtual machine, a private copy of a virtual machine image is created. This private copy can be created on a shared file system (for instance if the repository file system is accessible through NFS), or directly on the local disk of a VMM node by a file system copy or using the secure copy protocol (SCP). This phase of the provision process is called *virtual machine image propagation*.

A virtual machine image repository offers two different interfaces: one for users (to list, upload, and download virtual machine images), and one for VMMs (to propagate virtual machine images or save images of running virtual machines). The first interface generally follows a well-defined and stable protocol, in order to allow the creation of a large ecosystem of user tools. For instance, the Amazon S3 API is well established as a protocol for accessing IaaS cloud repositories. Both Eucalyptus and Nimbus provide replacement of Amazon S3, respectively called Walrus [21] and Cumulus [76]. OpenStack's repository, Swift [42], does not use S3 as its native API, but has experimental support for translating S3 requests to its native API (the Cloud Files API from Rackspace Cloud). The second interface, completely internal to an IaaS cloud, can follow many different implementations, depending on requirements of the cloud infrastructures. For instance, Nimbus has support for an advanced propagation mechanism, LANTorrent [23], designed for efficiently copying the same virtual machine image on many VMM nodes in parallel. This drastically decreases the creation time of a virtual cluster based on the same image.

In many cases, the virtual machine image repository is not dedicated to storing virtual machines images, but can also be used to store any kind of files. For instance, Amazon S3 is used to host static web objects like images, scientific data, etc. At the end of September 2011, Amazon S3 was holding more than 566 billion objects. This repository can also be used by applications executing in a cloud, to store input/output data. A standardization effort in this area is made by creating the Cloud Data Management Interface (CDMI) [6].

2.3.3.3 Virtual Machine Monitors Nodes

The third part of the architecture of an IaaS cloud is formed by virtual machine monitor nodes (or hypervisor nodes). These nodes provide the necessary infrastructure for executing the virtual machines provided by an IaaS cloud. The architecture of a virtual machine monitor node is presented in Figure 2.7.

The most important component is the hypervisor itself, in charge of executing virtual machines. Many different hypervisors can be used: Xen, KVM, VMware ESX, Microsoft Hyper-V [151], VirtualBox, etc. To control these hypervisors, a management program is invoked by the IaaS service, to perform actions like creating or terminating virtual

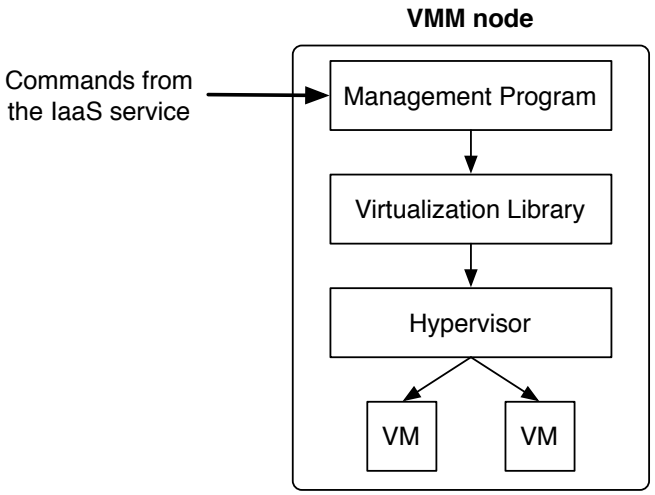


Figure 2.7: Generic architecture of a virtual machine monitor node in an IaaS cloud

machines. This management program is specific to the IaaS cloud software being used. For IaaS clouds supporting several hypervisor implementations, a virtualization library providing a common interface to different hypervisors can help simplify management programs. libvirt [24] is the de-facto library for this purpose, thanks to its support for a large number of hypervisors.

2.3.4 Open Problems in IaaS Clouds

Most problems in IaaS clouds are related to their lack of interoperability. Efforts have been made to make IaaS clouds more compatible, like the standardization of the OCCI API (see Section 2.3.3.1). However, true interoperability faces many problems. For instance, different IaaS clouds support different virtual machine image formats (usually linked to the hypervisors they rely on). Tools exist to convert images between different formats: Amazon Web Services proposes the VM Import service [62], and VMware distributes vCenter Converter [213]. Additionally, the DMTF released the Open Virtualization Format (OVF), a standard to package and distribute software to be run in virtual machines. However, there is no ubiquitous solution compatible with all IaaS clouds.

Live migration between different hypervisor implementations is also prevented by incompatible interfaces. A solution as been proposed by Liu et al. with Vagrant [142], a framework that supports live virtual machine migration between the Xen and KVM hypervisors. However, no standardization effort exists on this problem.

2.4 Federation of Cloud Computing Infrastructures

Most cloud computing infrastructures are completely independent and managed by different companies or organizations. Users who want to take advantage of multiple clouds at the same time need to interact with each cloud separately. Multi-cloud infrastructures can be required for the following reasons:

High availability Creating execution platforms hosted on multiple independent cloud infrastructures can make them more resilient to failures, ensuring high availability. For instance, on April 21st, 2011, an outage of Amazon Elastic Block Store (EBS) in the Amazon EC2 US east region brought hundreds of web sites down, including large web sites such as Quora, Reddit and Foursquare [68]. These outages are not uncommon in public clouds, and are becoming increasingly problematic as more and more Internet services rely on these infrastructures. By engineering execution platforms to be redundant across multiple cloud infrastructures (either using clouds belonging to the same provider, e.g. multiple regions of Amazon EC2, or using clouds from different providers), highly available services can be delivered.

Hybrid cloud An hybrid cloud is a combination of two different types of cloud, generally a private cloud and a public cloud. Many organizations have internal compute resources managed as private clouds. When more resources are needed than what is available in a private cloud, computations can be offloaded to public clouds (this is called *cloudbursting*). Federating private and public clouds allows to do this offloading in an automatic way.

Large or complex computing requirements For problems requiring large amounts of computing power to be solved, or problems with complex resource requirements that individual clouds cannot deliver, users may want to federate multiple clouds together (e.g. several community clouds) to create distributed execution platforms.

Federation can be performed at multiple levels of the cloud computing stack: IaaS or PaaS. Although federating different PaaS resources would be possible, PaaS clouds solve a large variety of problems with very different features and APIs. On the contrary, IaaS clouds offer a similar feature set revolving around the same goal: provisioning virtual machines to create execution platforms. Although each cloud's API can be implemented differently, the fundamental goal remains the same.

We consider two cases of IaaS cloud federation. First, resources from multiple sites of one IaaS cloud can be federated in a single platform, when requiring high availability or large numbers of resources. Cloud providers with multi-site infrastructures generally do not provide a fully transparent federation themselves, as most customers wish to control the geographical placement of their virtual machines (for example for minimizing latency with external Internet services and end users, or to address regulatory requirements). For instance, the Amazon Web Services infrastructure includes the concept of regions, which are accessed independently by customers. As such, customers are responsible for federating resources from multiple sites.

The second approach is to federate resources belonging to different providers. It can involve different APIs and virtual machine image formats. These federations can also include different deployment types, such as private and community clouds, potentially using restricted networks with private IP addressing and firewalls. As such, we consider it a more general problem than the previous approach. This multi-provider federation can be performed in several ways:

Manually by cloud users with help from software tools Several libraries exist to offer a unified API to multiple providers: Deltacloud [9], Libcloud [2], etc. A more

advanced approach, sky computing, is presented in Section 2.4.1. Sky computing creates fully configured and coherent platforms with all-to-all connectivity by transparently federating multiple clouds infrastructures.

Automatically through a cloud broker Recently, several projects have advocated for the creation of cloud brokers. These brokers would be independent entities which would create markets of cloud resources from different providers. They would allow to find resources matching user requirements at the best price. Further in this section, we present several projects that plan to build such cloud brokering infrastructures.

Automatically by cooperation between cloud providers Although currently there is little or no cooperation between cloud providers, as they compete for the market, members of the community are advocating for a cloud architecture where different cloud providers would lease resources from each other, and exchange user requests when they cannot handle them.

2.4.1 Sky Computing

Sky computing [130] is an approach for federating cloud computing infrastructures. It creates sky computing platforms by provisioning clusters of virtualized resources spanning multiple clouds. The particularity of this approach is to hide completely the boundaries of each cloud, by presenting each federation as a fully unified cluster. To achieve this goal, two components are used:

The ViNe virtual network ViNe [207, 208] is a virtual network relying on an overlay to transparently tunnel inter-cloud traffic and provide all-to-all connectivity between nodes participating in a sky computing platform. All-to-all connectivity, meaning that any node can initiate a connection to any other node, is crucial for many scientific applications. A ViNe network is composed of ViNe nodes (e.g. virtual machines running in a cloud) and of ViNe routers which act as relays for ViNe nodes. ViNe can be used to connect together networks which do not provide all-to-all connectivity, such as networks using private IP addressing, NAT, or containing firewalls.

Contextualization tools A sky computing cluster is deployed to provide a specific execution platform for users. In most cases, these platforms are composed of nodes providing different services (e.g. a front-end node, a storage node, worker nodes, etc.). As such, in order to deploy a consistent execution platform, virtual machines need to be configured to provide these roles. Some of the roles require virtual machines to know about other virtual machines involved in the cluster: for instance, nodes that need to mount a shared file system must learn the IP address of the storage server. All this configuration can be done automatically using contextualization tools, and is required in both single and multi-cloud scenarios. Many tools offer mechanisms for contextualization. OpenNebula uses ISO images to provide virtual machines with configuration parameters. Amazon provides CloudFormation [3], a service to provision a collection of AWS resources that follows a deployment and configuration template. Nimbus provides a context broker to assign roles to virtual machines and configure nodes [129].

Additionally, the Nimbus contextualization tools are able to configure machines belonging to different clouds. The Nimbus context broker can manage resources from different Nimbus clouds and from Amazon EC2. Another tool from the Nimbus project, cloudinit.d [75], provisions and configures sets of virtual machines on different clouds by following a deployment plan. cloudinit.d uses the boto [5] and Libcloud [2] libraries, making it compatible with a large number of cloud infrastructures.

2.4.2 Other Cloud Federation Efforts

The Contrail project [8] aims to create a fully integrated open source cloud stack (combining both IaaS and PaaS) with built-in support for cloud federations. Clouds using the Contrail software will be able to provide resources to customers and to other clouds when its infrastructure is not used at its maximal capacity, and will be able to get resources from other clouds in periods of peak activity. Contrail will provide a service stack providing both IaaS virtual machines and PaaS services (web application hosting, databases, and MapReduce execution platforms), with service level agreements (SLAs) and quality of service (QoS).

The mOSAIC [25] project will allow users to specify service requirements and will find best-fitting cloud resources according to their needs. This will be implemented through a multi-agent broker that can compose services from multiple cloud providers.

Buyya et al. propose InterCloud [77], an architecture for federated cloud computing environments supporting scaling of applications across multiple cloud providers. This architecture is composed of several entities. Cloud coordinators export information about the cloud infrastructure to the outside world. A cloud exchange stores activity patterns of multiple clouds and acts as a central point to enable load sharing between multiple domains. A cloud broker is responsible for matching and negotiating user requirements with cloud coordinators.

The RESERVOIR [186] project created an architecture allowing cloud providers to dynamically partner together. Their model separates the services providers from the infrastructure providers. Service providers lease resources from infrastructure providers.

Open Cirrus [78] develops a cloud computing testbed federating heterogeneous data centers. They offer physical and virtual machines, and services such as single sign-on, global monitoring, and storage services. Similarly, the BonFIRE European project [4] federates multiple experimental cloud infrastructures to create a testbed for experimentation of the Internet of Services.

Celesti et al. [83] present enhancements to cloud architectures to increase federation capabilities. Their proposal is based on a cross-cloud federation manager, a new component of cloud architectures that establishes federations through three phases: discovery (finding other available clouds), match-making (selecting other clouds with compatible requirements), and authentication (establishing a trust context with the selected clouds).

2.5 Conclusion

In this chapter, we presented the background of this thesis. We first focused on two aspects: virtualization, and parallel and distributed computing. Virtualization is a technique that decouples software from underlying physical hardware, increasing flexibility

and portability. Virtualization can be implemented using different approaches: hardware virtualization, operating system-level virtualization, and process-level virtualization. Hardware virtualization, which runs regular operating systems in virtual machines, offers a flexible approach to the management of computing infrastructures. Existing applications and operating systems are fully encapsulated in virtual machines, becoming totally independent of the underlying hardware. Although hardware virtualization did not spread to the whole industry after its initial popularity in the 1970s, hypervisors have been much improved in the recent years, which helped to reintroduce the technology among many computing fields. Live migration is one of the fundamental features offered by recent hardware virtualization implementations. Live migration relocates virtual machines from one physical node to another one, with negligible interruption for users. It offers numerous advantages, such as improved load balancing, reduced energy consumption through consolidation, transparent infrastructure maintenance, and improved fault tolerance. Numerous works have been proposed to improve the performance of live migration. However, few research works have focused on improving its efficiency in the context of wide area networks.

Modern computer science has evolved at a dramatic speed since its inception in the middle of the 20th century. Big and slow computers were continuously replaced by smaller and faster hardware. However, computing requirements have increased as well. The only way to solve these problems is to leverage the power of both parallel and distributed systems consisting of large numbers of processors by using them in a concurrent fashion. Every day, scientists use infrastructures with hundred of thousands of processors to model and solve complex problems. We presented the properties of applications used in these systems: their dynamic capabilities, and the major programming models used to create parallel and distributed applications. We covered the architectures used to execute parallel and distributed applications. Initially, dedicated supercomputers such as those designed by Seymour Cray were the main platforms used to run parallel programs. Today, some massively parallel computers are still used, such as the IBM Blue Gene. However, the field of High Performance Computing is mostly relying on clusters. Clusters are general purpose computers connected through local network interconnects. Clusters encompass a large variety of systems, ranging from small infrastructures connected with Gigabit Ethernet to large scale systems with thousands of nodes connected with high speed interconnects.

Bringing further the scale of distributed infrastructures, the concept of grid computing has been proposed in the 1990s. Grid computing federates geographically distributed resources belonging to multiple organizations sharing the same goal, aiming to deliver them in a fully transparent way. It allows scientists to solve very large problems by combining resources from different partners. However, the ultimate goal of grid computing was to provide computing power to anyone as transparently as electricity is delivered through an electric grid. Although many grid systems have been successfully used by scientists, this goal of creating *the Grid* has not been reached.

From this context, cloud computing has emerged. In this new paradigm, companies (cloud providers) deliver computing resources in an elastic and on-demand fashion. Customers pay only for resources they actually consume. Several levels of abstraction exist in cloud computing: Infrastructure as a Service, Platform as a Service, and Software as a Service. IaaS allows to provision raw virtual machines to build customized execution platforms. It has attracted a lot of interest in the scientific community, as

it provides compatibility with existing applications and the power to create execution platforms matching user requirements. Furthermore, the growing popularity of IaaS infrastructures has led to the availability of different deployment types: public clouds, private clouds, and community clouds.

With the increasing number of IaaS clouds, resources from different infrastructures can be federated together to create large scale execution platforms. However, federating cloud infrastructures brings a number of challenges. Several working groups are focusing on creating API standards for cloud services, in order to improve cloud interoperability. Additionally, many projects are working on cloud federation architectures that would optimize the provisioning of resources among multiple cloud providers, according to user and application requirements. In this context, the first focus of our work is the efficient and easy creation of large-scale elastic execution platforms on top of federated clouds.

The different deployment types of cloud infrastructures also present variable availability of resources. Private cloud infrastructures can become overused, in which case users can decide to move their computation to public clouds. Moreover, we believe that the cloud ecosystem will shift towards a more dynamic economic model, in which clouds will deliver resources at variable prices depending on availability. Amazon EC2 already includes spot instances, a type of *best-effort* resources provided at a variable price. In such context, making execution platforms more dynamic by allowing them to be live migrated between different clouds would bring important advantages for both users and administrators. As such, the second focus of our research is to propose efficient inter-cloud live migration mechanisms that would serve as a founding block for the dynamic usage of these next generation cloud computing infrastructures.

Part II

Elastic Execution Platforms over Federated Clouds

Chapter 3

Large-Scale and Elastic Execution Platforms over Federated Clouds

Contents

3.1 Introduction . . . 57
3.2 Scientific Computing on Clouds . . . 57
3.3 Sky Computing . . . 58
3.3.1 ViNe . . . 58
3.3.2 Nimbus Context Broker . . . 58
3.4 Large-scale Sky Computing Experiments . . . 59
3.4.1 Elasticity . . . 59
3.4.2 Experimental Infrastructures . . . 59
3.4.3 Deployment of Nimbus Clouds . . . 61
3.5 Scalability of Cloud Infrastructures . . . 62
3.5.1 Propagation with a Broadcast Chain . . . 64
3.5.2 Propagation using Copy-on-write Volumes . . . 64
3.5.3 Performance Evaluation . . . 65
3.5.4 Combination of the two mechanisms . . . 66
3.5.5 Related work . . . 67
3.6 Scalability of Sky Computing Platforms . . . 67
3.7 Extending Elastic Sky Computing Platforms . . . 68
3.8 Related Work . . . 69
3.9 Conclusion . . . 70

Many different cloud computing infrastructures are now available, with large numbers of public, private, and community clouds. In this context, federating resources from multiple clouds allows to build large-scale execution platforms, in order to execute applications requiring large amounts of computational power. Additionally, elastic capabilities of clouds are interesting for dynamic applications. This chapter presents our contributions for efficiently building large-scale and elastic execution platforms when federating resources from multiple clouds, using *sky computing*. This work was realized in collaboration with Kate Keahey (Argonne National Laboratory / Computation Institute, University of Chicago), Maurício Tsugawa, Andréa Matsunaga, and José Fortes

(University of Florida). It was initiated during a 3-month visit in the ACIS laboratory at University of Florida, Gainesville, USA, in summer 2010.

3.1 Introduction

In the previous part of this thesis, we presented clusters and grids, two architectures that have been used by the scientific community to execute parallel and distributed applications with important performance requirements. More recently, the emergence of cloud computing has attracted a growing interest from this community. Scientists see in cloud computing a potential to create on-demand execution platforms with minimum upfront investment. Instead of buying expensive hardware clusters, cloud computing allows to rent resources only when they are required, and using amounts of resources matching application requirements. With the increasing popularity of cloud computing, scientists are getting access to multiple clouds of different types (public, community/scientific, private), some of them being of small to medium size. In this chapter, we explore how to create large-scale and elastic execution platforms with resources from distributed clouds, using sky computing. This approach federates multiple clouds in a transparent way to execute unmodified applications.

3.2 Scientific Computing on Clouds

The field of scientific computing solves scientific problems by constructing models and analyzing them using computer programs. The types of scientific problems that are studied include simulations of physical events (earthquakes, tornadoes) or bioinformatics research (genome sequence alignment). Solving these problems requires large amounts of computing power, and scientists usually rely on infrastructures such as clusters and grids for their execution.

Cloud computing appeared in the middle of the 2000s, initially targeting web service hosting. It has been adopted by many businesses, from startups to large corporations. Since then, scientists have been studying whether cloud computing, and particularly IaaS clouds, can be used for execution of scientific applications with high performance requirements [116, 124, 217]. Most studies conclude that applications with low communication rates (e.g. embarrassingly parallel applications) can be successfully executed on IaaS clouds, while more tightly coupled applications suffer important performance degradations. This is explained by two factors. First, clouds resources lack the high speed interconnects commonly found in high performance clusters. Second, the infrastructure is generally shared, with many collocated machines on the same host. This can increase the latency of communications, which decreases performance of tightly coupled applications.

Different types of IaaS clouds are available to scientists: commercial clouds from many different providers (public clouds), clouds managed by academic institutions and dedicated to the scientific community (scientific clouds), and clouds created from private clusters (private clouds) using open source IaaS toolkits. Some of these clouds are of small or medium size, and cannot run large computations. In such an ecosystem, it is possible to build distributed execution platforms on top of multiple clouds to run scientific software requiring large amounts of computational power. This is the fundamental idea behind the approach studied in this chapter: sky computing [130].

3.3 Sky Computing

Sky computing, proposed by Keahey et al., transparently federates cloud resources from multiple IaaS clouds in order to create a single virtual execution platform on top of highly distributed resources. By putting all these resources, gathered from different clouds, in a single trusted networking environment, they can be used exactly like a local cluster. This simplifies management of the resources, and enables execution of unmodified applications designed to run on a local cluster. This transparent federation is performed using two tools: the ViNe virtual network and the Nimbus context broker.

3.3.1 ViNe

ViNe is a virtual network implementation developed at the University of Florida [207, 208]. It is based on network overlay on Internet connections, and targets high performance communications. It enables all-to-all communications between resources from different physical networks, even when some of the resources are using private IPs and are in networks behind NATs or firewalls. In a sky computing context, ViNe provides all-to-all connectivity between virtual machines of different clouds.

The architecture of ViNe is composed of two types of components. At the edge of the network, ViNe nodes (VN) communicate between each other either directly (using the available switched or routed physical network) or through the ViNe infrastructure. At the center of the virtual network, ViNe routers (VR) create an overlay network used for forwarding traffic between VNs which cannot communicate directly. Generally, a single VR is deployed on each LAN segment. Each VR is responsible for VNs located on the same subnetwork. However, it is also possible to create deployments involving more VRs, including setups in which every VN acts as a VR (in which case all communication goes through the ViNe network overlay).

3.3.2 Nimbus Context Broker

When a virtual machine is provisioned from a cloud, the operating system contained in the virtual machine image is booted. However, some operating system services and applications need to be configured to be operational. Parts of this configuration can depend on information known only at deployment time, such as IP addresses and host names. This information can correspond to the virtual machine itself, or to other virtual machines belonging to the same virtual cluster – the same *context*. For instance, a virtual machine acting as an NFS client needs to know the IP address of its NFS server in order to be able to mount its file system. Adapting a virtual machine to its deployment context is called *contextualization*. The Nimbus context broker [129] performs contextualization of resources, by distributing roles to the machines of a virtual cluster and configuring them. During the boot process, each virtual machine executes a small program called the *context agent*. The context agent queries the context broker service to learn the roles allocated to the virtual machine, along with any required configuration data, such as IP addresses and host names of other virtual machines. It then executes scripts to configure the environment according to the roles and information learned from the context broker.

3.4 Large-scale Sky Computing Experiments

Our large-scale sky computing experiments focused on the deployment of sky computing platforms and their elastic extension. We deployed sky computing platforms on top of multiple clouds, and used them to execute Basic Local Alignment Search Tool [55, 162] (BLAST), a bioinformatics application that finds regions of local similarity in biological sequences. We use a version of BLAST adapted to run on top of the Hadoop framework [150].

3.4.1 Elasticity

A major feature offered by cloud computing is elasticity. We studied how a sky computing platform running a scientific application can be dynamically extended. Two challenges present themselves in this scenario. First, extending a sky computing platform, by provisioning new virtual machines and integrating them in an existing execution platform. Second, adapting the application to use the extended pool of resources.

3.4.1.1 Execution Platform Elasticity

To provide elasticity at the level of resources composing sky computing platforms, we interact with IaaS clouds. IaaS clouds are able to provide new resources at any time, and create them in a matter of seconds to minutes. Resources are requested through cloud APIs, usually SOAP or REST web services. Accessing this type of interface can be easily scripted and automated. In our experiments, we used the Nimbus cloud client to request resources from different Nimbus clouds. After new resources have been requested, contextualization automatically integrates them into an existing sky computing platform, by making them join the ViNe virtual network and configuring their role in the platform.

3.4.1.2 Application Elasticity

The application also needs to be adapted to use the newly provisioned resources. This is not always possible: some applications are static and cannot adapt to a changing platform (see Section 2.2.1). In our work, we used the Hadoop MapReduce framework, which supports dynamic addition and removal of resources. A MapReduce job is broken in a large number of tasks. Tasks pending execution can be scheduled to nodes joining a Hadoop cluster. Moreover, addition of new nodes does not require any action on the Hadoop master node: a new Hadoop resource contacts the master node to join the cluster. Using this feature, the elastic extension of a sky computing platform is automatically leveraged by Hadoop applications.

3.4.2 Experimental Infrastructures

Studying and evaluating large-scale distributed systems can be performed through different types of methods: analytical modeling, simulation, or real life experiments.

Analytical modeling represent the behavior and performance of a system using mathematical properties. Its principal advantage is to provide direct insight on what variables are conditioning the behavior of the system, and how important each of them

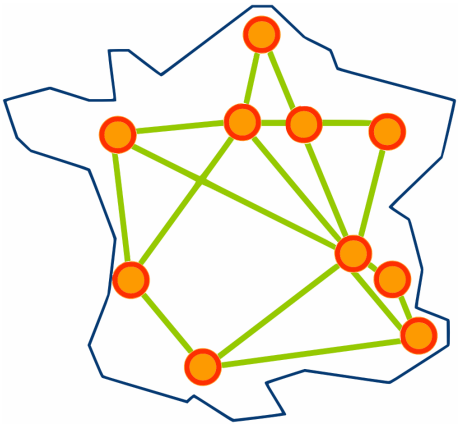


Figure 3.1: Map of the Grid'5000 testbed

is. However, the more complex the system, the more complicated the analytical model is. The behavior of a system based on a multi-cloud platform is controlled by a large number of variables, making it difficult to create a realistic model.

Simulation reproduces the behavior of a system by replicating its state in a virtual environment called a simulator. The actions performed in a simulator follow a model. This model can be close to reality by replicating the system with a high precision level, or can be at a high level of abstraction by simplifying the behavior of the system. The more precise the model is, the more compute intensive the simulation process is. SimGrid [82] is an example of a simulator targeting distributed systems. Simulation can be used to provide good insight on the behavior of a system, but a model too abstracted from reality can hide issues that would be revealed in real-life conditions.

The third method is to run actual implementations of a system on a real infrastructure to observe its behavior in a real-life environment. This allows to validate a system implementation. To enable scientists to perform experiments with large-scale distributed systems, several projects have created large-scale experimental testbeds, such as PlanetLab [175] for research on distributed network services, or GENI [17] for research on network architectures.

For experimenting with federations of cloud computing infrastructures, large scale and distributed testbeds are particularly suited. They offer access to a large number of clusters distributed over multiple sites, which resembles the multi data center scenarios found when dealing with multiple clouds. In our experiments, we use two large-scale and distributed experimental testbeds: Grid'5000 and FutureGrid. We now present in more details the specificities of both testbeds.

3.4.2.1 Grid'5000

Grid'5000 [79] is an experimental testbed for research in large-scale parallel and distributed systems. It is geographically distributed on 10 sites in France. Additionally, two sites outside of France (in Luxembourg and in Porto Alegre, Brazil) are being integrated. As of September 2011, Grid'5000 offered access to more than 1,500 nodes, aggregating more than 8,500 cores. All sites in France are connected through a 10 Gb/s backbone. Figure 3.1 presents a map of the Grid'5000 testbed and its network backbone. Grid'5000

provides a highly reconfigurable, controllable and monitorable experimental platform. It offers a tool (Kadeploy3) to deploy customized operating systems (primarily used to deploy Linux, but supporting other systems such as FreeBSD) and application stacks on bare hardware. This allows experiments in all software layers from network protocols to applications. The capabilities of Grid'5000 were available before the emergence of cloud computing. They can be seen as similar to those provided by IaaS services, except that they deliver physical resources rather than virtualized ones, which allows more control over the infrastructure.

3.4.2.2 FutureGrid

FutureGrid [15] is an experimental testbed for grid and cloud research. It is distributed over 6 sites in the United States and offers around 4,000 cores (and is planned to grow to 5,000). All sites but one are connected through a private network. Figure 3.2 presents a map of the FutureGrid testbed and its wide area connectivity. FutureGrid resources provide several types of services. Nodes are statically allocated to one of the following groups:

- HPC nodes are managed by the TORQUE [44] batch scheduler. They are not virtualized and are running a fixed operating system suitable for executing parallel programs (e.g. MPI). This setup is similar to a production cluster, and allows users to experiment with parallel and distributed applications.
- 4 Nimbus clouds are available, at Texas Advanced Computing Center (TACC), University of Chicago (UC) in Illinois, San Diego Supercomputer Center (SDSC) in California, and University of Florida (UF).
- 2 Eucalyptus clouds are available, at Indiana University and SDSC.
- A small OpenStack cloud (24 cores) is available at SDSC.

On all FutureGrid clouds, users can deploy virtual machines (from images created by the administrative staff or from their own images) to experiment with middleware or systems which would be difficult or impossible to deploy on HPC nodes without administrative privileges.

3.4.3 Deployment of Nimbus Clouds

Whereas FutureGrid offers access to several Nimbus clouds, Grid'5000 does not include any cloud infrastructure. Users access to nodes in two different modes:

Normal mode In this mode, nodes are executing a production operating system installation. Users do not have administrative privileges in this environment.

Deployment mode In this mode, customized operating systems can be deployed on nodes. Users have administrative privileges, allowing them to install, configure, and modify software.

We developed software to reserve Grid'5000 nodes in deployment mode, deploy a customized operating system, and install and configure a Nimbus cloud. It relies on the

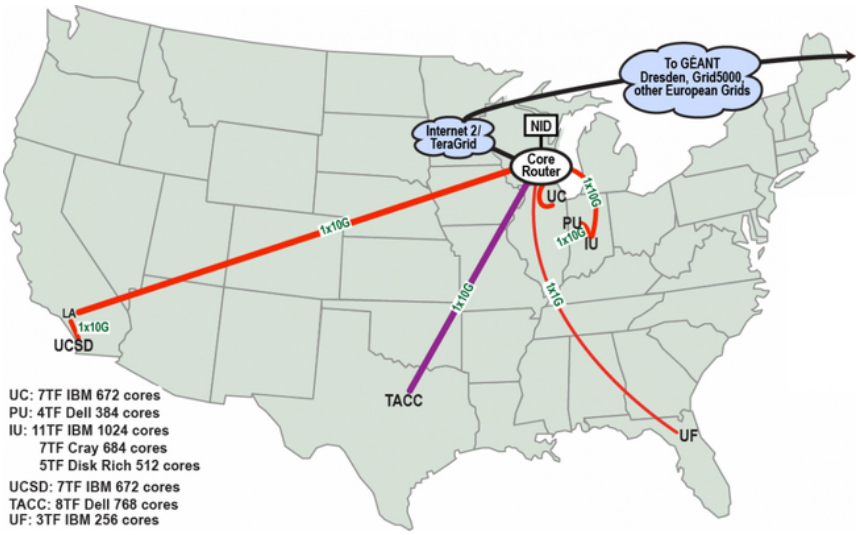


Figure 3.2: Map of the FutureGrid testbed

Chef [170] configuration management tool to manage the different components of the Nimbus cloud infrastructure and their dependencies. This utility configures nodes in a parallel fashion, and has been used to configure Nimbus clouds with hundreds of nodes in several minutes.

3.5 Scalability of Cloud Infrastructures

Provisioning a large scale sky computing platform on cloud infrastructures is performed by the following sequence of actions:

1. **Provisioning request** A request for provisioning a sky computing platform is made to the cloud infrastructures chosen to host the platform.

2. **Resource allocation** Each cloud infrastructure allocates the required virtual machines by reserving resources on their hypervisor nodes.

3. **Virtual machine image propagation** As explained in Section 2.3.3.2, each virtual machine requires its own private copy of a virtual machine image in order to write modifications to the file system. In this step, images are propagated from a cloud storage repository to the hypervisor nodes selected in step 2.

4. **Virtual machine creation** Once virtual machine images have been propagated on each hypervisor node, virtual machines are created by interacting with the hypervisor. Once started, each virtual machine boots its operating system.

5. **Execution platform configuration** When all virtual machines have booted, the contextualization takes place in order to configure the execution platform for its intended purpose. This includes configuration of the ViNe virtual network to connect together resources allocated on different clouds.

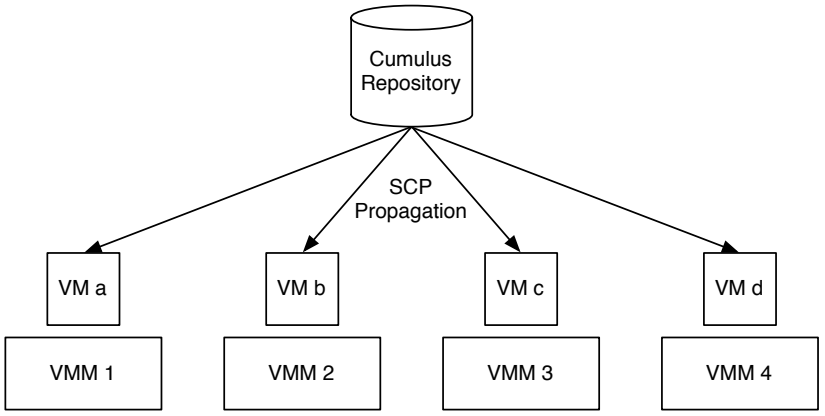


Figure 3.3: Standard SCP-based propagation in Nimbus Infrastructure

In our experience with Nimbus clouds, the most significant scalability issue is related to the propagation of virtual machine images (step 3). To provide each virtual machine with its own copy, Nimbus was performing network transfers from the repository node storing virtual machine images to each hypervisor selected to run virtual machines, using the Secure Copy Protocol (SCP). Figure 3.3 presents the SCP-based propagation mechanism used by Nimbus to transfer images from the Cumulus repository to the VMM nodes. In this example, four hypervisors were allocated, each hosting one virtual machine. Thus, four SCP copies of the virtual machine images are required.

Since all data originates from the repository node, the transfer capability of this machine (either disk performance or network bandwidth) is the limiting factor for the performance of propagation. When small clusters of a few virtual machines are provisioned, this bottleneck is not very limiting. However, when scaling up to hundreds of virtual machines, this limitation severely increases propagation time. For example, assuming a theoretical network bandwidth of 125 MB/s (the speed of Gigabit Ethernet, ignoring network protocols overhead) and a 5 GB virtual machine image, the propagation time is shown in Table 3.1 for an increasing number of virtual machines. While most users will accept to wait less than 3 minutes to deploy a 4-node virtual cluster, propagation times of one or several hours prohibit any large-scale deployment. To avoid this problem and accelerate the creation of large-scale virtual clusters, we developed two propagation mechanisms with improved performance.

A fundamental property of virtual clusters is that most or all virtual machines are started from identical virtual machine images. For example, many clusters exhibit a master-slave architecture where one or a few master nodes are started from a master virtual machine image, and many slave nodes are started from the same slave virtual machine image. In our study, we use the same image for both the Hadoop master node and the Hadoop slave nodes, since only a few configuration files need to be changed between master and slaves.

Since we are dealing with propagating identical content to many nodes, this enables us to perform optimizations which would not be possible if the virtual machines were all started from different images.

Number of VMs	Propagation time
1	41 s
2	1 min 22 s
4	2 min 44 s
8	5 min 28 s
16	10 min 55 s
32	21 min 51 s
64	43 min 41 s
128	1 h 27 min 23 s
256	2 h 54 min 46 s
512	5 h 49 min 32 s
1024	11 h 39 min 3 s

Table 3.1: Theoretical propagation time of a 5 GB image for different numbers of virtual machines

3.5.1 Propagation with a Broadcast Chain

The first optimization we developed is based on the Kastafior and TakTuk [88] programs, created at Inria. TakTuk is a tool for deploying parallel remote executions of commands to a potentially large set of remote nodes. It sets up a logical interconnection network which allows to perform I/O multiplexing and demultiplexing. Kastafior uses this logical interconnection network to create a broadcast chain between all nodes: the first node (which is the storage node) is connected to the second node, which is connected to the third one, etc.

Once the chain is created, we use it to broadcast virtual machine images from node to node: the storage node streams the image to the second node, which forwards it to the third one, etc. If n is the number of virtual machines to start and *size* is the size of the propagated virtual machine image, this reduces the amount of data sent from the storage node from $n \times size$ to *size*. This mechanism is illustrated in Figure 3.4. Like in the previous example with SCP, one virtual machine image is copied on each hypervisor node. In this case, the broadcast chain is established between all VMMs, and only one copy is made by the repository. The three other transfers are made by broadcasting the virtual machine image data from node to node.

However, this propagation mechanism is still limited by the time required to send one virtual machine image copy. In order to make virtual cluster provisioning almost instantaneous, we developed an additional optimization based on pre-propagated copy-on-write volumes.

3.5.2 Propagation using Copy-on-write Volumes

In a virtualization context, copy-on-write (COW) volumes are virtual disks storing only modified blocks compared to another virtual disk. A virtual machine image, called the backing file, contains the original virtual machine image data. When a virtual machine writes a modified block to disk, the modified data is stored in the copy-on-write volume instead of the original image. This allows multiple virtual machines to share the same backing file in read-only mode and independently write their modifications in separate

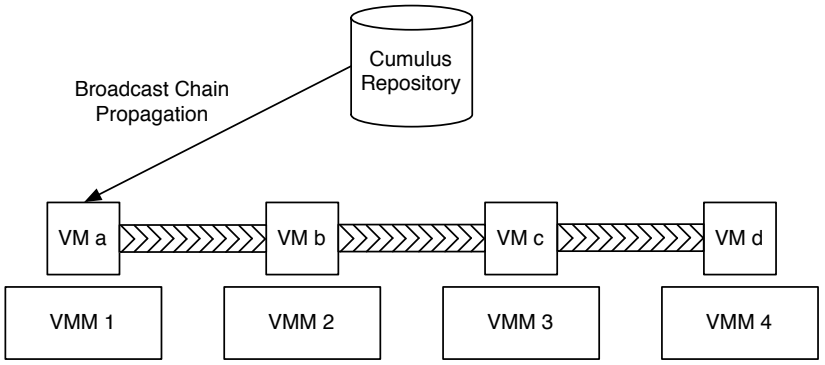


Figure 3.4: Improved propagation based on a broadcast chain

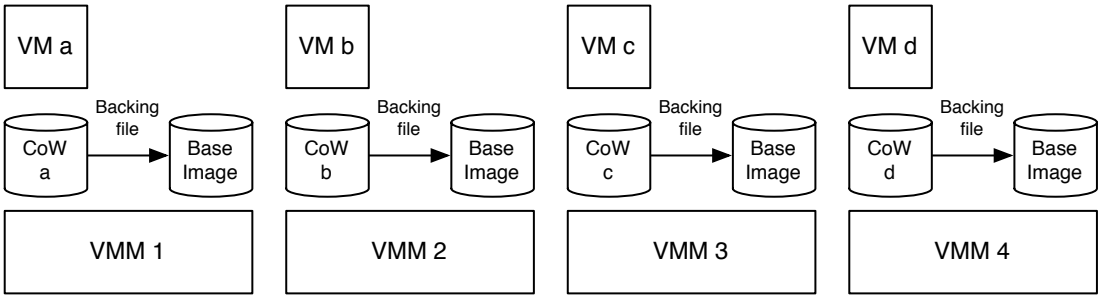


Figure 3.5: Improved propagation based on pre-propagated copy-on-write volumes

COW disks.

This technology is implemented in the Xen and KVM hypervisors, using a file format called QCOW. We created a propagation mechanism leveraging the Xen implementation so that virtual machines based on the same virtual machine image would share a unique backing file. Since creating a copy-on-write volume is an almost instantaneous operation, this offers a great speed up in propagation time. However, virtual machines still need to access the backing files. We decided to pre-propagate our backing files to all hypervisors, which is performed before requesting a cluster instantiation. This mechanism is illustrated in Figure 3.5.

Note that it would also be possible to store the backing file on a shared file system (e.g. NFS). In this case, no pre-propagation would be needed. However, a large virtual cluster accessing the same base image could generate a lot of traffic, and be slowed down by the shared file system performance.

3.5.3 Performance Evaluation

To evaluate the performance of these two mechanisms, we measured the time required to start and contextualize virtual clusters with an increasing number of virtual machines. These experiments were performed on Grid'5000 using nodes of the *gdx* cluster in Orsay. These nodes are equipped with AMD Opteron 246 (2.0 GHz) or AMD Opteron 250 (2.4

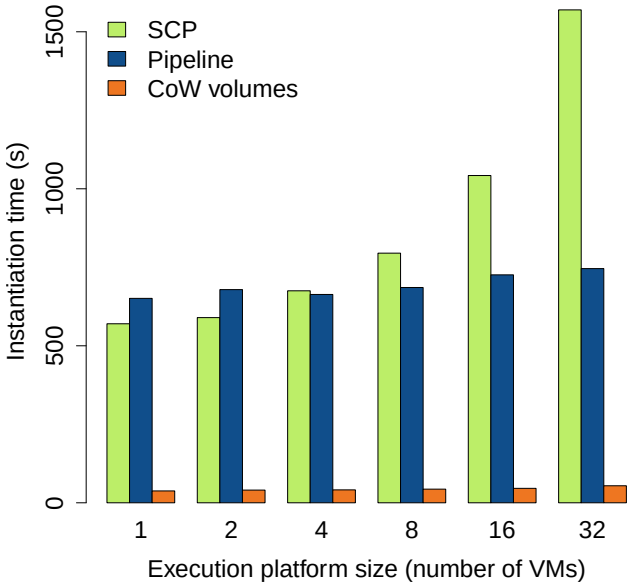


Figure 3.6: Propagation performance

GHz) processors, 2 GB of memory, Gigabit Ethernet, and 80 GB SATA drives. They were running Debian Lenny with the Xen hypervisor (version 3.2) and a 2.6.26 Linux kernel.

In the cases of SCP (the standard Nimbus propagation) and Kastafior, the image is compressed and is 2.2 GB in size. Once transferred, it is uncompressed to 12 GB. For QCOW, the image was pre-propagated and uncompressed on all nodes before requesting virtual machine instantiation. In this case, propagation consists in creating new copy-on-write volumes, followed by starting and contextualizing the virtual cluster.

Figure 3.6 presents the performance results we obtained. With SCP, when the number of virtual machines increases, the time required to start a virtual cluster greatly increases, going from less than 10 minutes for one virtual machine to almost 30 minutes when starting 32 virtual machines. When using Kastafior, augmenting the number of virtual machines has a low impact on instantiation time: the propagation time is almost constant. However, it remains higher than 10 minutes (this is because the nodes need to uncompress the 12 GB image). Finally, using pre-propagated Xen copy-on-write volumes, the instantiation time remains low, under one minute. The small increase in instantiation time is caused by an increase in contextualization time, and not in the propagation time itself.

3.5.4 Combination of the two mechanisms

The two propagation mechanisms we proposed were developed independently. Propagation with a broadcast chain allows to efficiently propagate the same virtual machine image to many nodes, avoiding the bottleneck created by centralized image repository. Propagation using copy-on-write volumes allows to quickly instantiate virtual machines once a virtual machine image has been pre-propagated on hypervisors nodes.

These two mechanisms can be combined together: broadcast chain propagation can be used to pre-propagate virtual machine images when they are required for the first time. Then, copy-on-write volumes can be used to create each virtual machine private storage file. Backing files can be kept in a local cache on each hypervisor node, and be readily available when instantiating other virtual machines based on the same image.

One possible downside of copy-on-write volumes is storage consumption. In the worst case, if a virtual machine overwrites all data of its virtual disk, storage consumption is the double of the size of the virtual disk, since the hypervisor node contains both the base image and a fully modified copy-on-write volume. This additional storage consumption decreases when more virtual machines with the same backing file are hosted on the same node. For instance, if four collocated virtual machines overwrite their whole virtual disk, they consume only 25% more data than non copy-on-write approaches. In the best case, copy-on-write volumes stay unmodified and the storage consumption is limited to the space used by the backing file. In conclusion, all mechanisms require to correctly manage available storage on hypervisor nodes, by taking into account the potential storage consumption of each approach.

3.5.5 Related work

Several related projects have looked at the issue of virtual machine image propagation performance. In this section, we review these propositions and compare them with our contributions.

After we implemented our enhanced propagation mechanisms, the Nimbus project developed LANTorrent [23], a file distribution protocol that sends virtual machine images to many nodes in a multicast manner. It provides improvements similar to the broadcast chain mechanism, although it uses a different technique: a centralized server acts as a tracker (like in BitTorrent [89]), informing nodes about which other nodes can be contacted to get a portion of the virtual machine image.

Nicolae et al. propose a virtual file system optimized for storing virtual machine images [165], backed by the BlobSeer distributed storage service [164]. It uses a lazy transfer scheme to transfer chunks of virtual machine images only when they are accessed by the guest operating systems. This system offers excellent performance, but relies on an additional component, the BlobSeer distributed storage service, which complicates its integration within a Nimbus cloud deployment.

Wartel et al. studied the performance of scp-wave, an image distribution mechanism based on SCP and a Fibonacci tree, and BitTorrent [219]. BitTorrent shows good performance by transferring a 10 GB image to 462 hypervisors in 23 minutes.

Schmidt et al. compare the performance of several propagation methods: unicast distribution, binary tree distribution, peer-to-peer distribution based on BitTorrent, and multicast [191]. They also evaluate a copy-on-write layer to avoid unnecessary transfer. Unlike our approach, they do not rely on copy-on-write volumes like QCOW, but on a copy-on-write file system implementation (UnionFS).

3.6 Scalability of Sky Computing Platforms

As presented earlier, we use Hadoop as the application framework in our sky computing experiments. Hadoop is designed to scale to large numbers of nodes and is used

Experiment	# Clouds	# VMs	# Cores	Speedup
A	3	330	660	502
B	6	750	1500	870

Table 3.2: Performance of BLASTx on sky computing platforms. The speedup is relative to the time to execute a sequential BLASTx execution.

in very large deployments by companies such as Yahoo! and Facebook. However, in our experience, using Hadoop with hundreds of resources already requires expertise. Known issues are that Hadoop can use more file descriptors than the default limit of most Linux distributions, or run out of DataNode threads. We had to customize our deployment of Hadoop (version 0.20.2) to tune these configuration settings.

On top of the Hadoop framework, we ran BLASTx, one of the programs of the BLAST application. BLASTx searches a protein database against a translated nucleotide query. BLASTx is embarrassingly parallel in nature, as each query sequence can be treated independently. In our experiments, the 3.5 GB non-redundant (NR) protein database was stored in the virtual machine images (to avoid network traffic when accessed). By using the propagation mechanisms presented in Section 3.5, this large database was efficiently transferred to all nodes as part of the virtual machine image.

We ran our experiments with 50,000 query sequences contained in a 15 MB file stored in the Hadoop Distributed File System (HDFS). With the default replication factor, this file was replicated to only three HDFS storage nodes. When BLASTx was started, all nodes of the Hadoop cluster would fetch portions of the sequence file from these three nodes. In order to make this part of the execution more scalable, we increased the HDFS replication factor of the sequence file to replicate it on all nodes.

Table 3.2 summarizes the performance of BLASTx in two different sky computing platforms by reporting their speedup compared to a sequential run. Experiment A was conducted with resources from three Nimbus clouds of FutureGrid: SDSC, UC, and UF, for a total of 660 cores. Experiment B added resources from three Nimbus clouds deployed on Grid'5000, for a total of 1,500 cores. These results show that the speedup/core ratio of the sky computing platform with six clouds is lower than the one with three clouds. This could be explained by the fact that the network infrastructure was limiting inter-cloud communication performance, as all network traffic went through a unique ViNe router connecting FutureGrid and Grid'5000. Further investigation is required in order to determine the exact performance limitations in these platforms, and to research how the execution environment can be adapted to improve them.

Nevertheless, these results show that sky computing platforms federating multiple clouds can lead to substantial speedups. Increasing the size of the sky computing platform by using six cloud instead of three led to a 73% increase in speedup.

3.7 Extending Elastic Sky Computing Platforms

The previous experiment presents results of computations deployed on static platforms, as the sky computing platforms remained the same during the whole execution of the BLASTx computation. In order to study the elasticity of sky computing platforms, we started a sky computing platform using FutureGrid resources spanning three different

clouds. Once a BLASTx computation was started on this platform, we extended it with Grid'5000 resources from four different sites. To perform this extension, we generated contextualization profiles matching the information of the existing platform (ViNe router IP, Hadoop master IP, etc.). When this contextualization profile is used to provision new virtual machines, they are automatically included into the existing sky computing platform. Once the new resources join the platform, they start executing MapReduce tasks of the BLASTx computation. This dynamic extension speeds up a computation after it has been started.

We compared the execution of two BLASTx jobs, each composed of 10,000 MapReduce tasks. When executing only on FutureGrid resources, the job finished in about 3,000 seconds. In the second execution, the execution platform was extended with resources from Grid'5000, which accelerated the progress of the BLASTx job, making it finish in about 2,300 seconds.

This elastic extension of sky computing platforms can be used in different scenarios. For example, when resources become available in a federation of private or community clouds and no new computation needs to be scheduled, they could be used to extend existing sky computing platforms instead of being unused. Another use case exists with job deadlines. If deadlines are modified and more resources are required to respect them, the sky computing platform executing the computation can be extended to get more resources in order to finish faster and respect the deadline.

3.8 Related Work

To our knowledge, few works have studied the performance of large scale execution platforms built on top of multiple federation clouds. Moreno-Vozmediano et al. [155] executed Many-Task Computing applications on top of multiple clouds: a local infrastructure, two Amazon EC2 regions, and ElasticHosts [12]. However, they restricted their evaluation to small cluster sizes, and used simulation to extend their results to 256 worker nodes. Vöckler et al. [215] executed an astronomy workflow application with an execution platform of 150 dual-core virtual machines provisioned from multiple FutureGrid clouds. They conclude that sky computing is a viable and effective solution for executing their application.

Elastic capabilities of cloud infrastructures are also a topic of interest in the community. The main area of focus has been on providing elasticity to web hosting platforms. These platforms leverage elasticity in order to adapt to changes in user traffic: peak hours, *Slashdot effects*, busy periods (Christmas holidays for online retail), etc. Amazon Web Services provides Auto Scaling [60], a service to scale up and down platforms built from Amazon EC2 resources. The Contrail project develops ConPaaS [7], a Platform-as-a-Service layer providing a set of elastic high-level services [90] and runtime environments [91]. The high-level services include Scalarix, a scalable transactional key-value store, and SQL databases. Runtime environments allow MapReduce computations, web hosting of Java servlets and PHP applications, and execution of Bag of Tasks computations based on the BaTS algorithm [169]. These services and environments will provide elastic capabilities to respect their SLAs. Finally, several works have studied architectures and policies to extend local clusters with cloud resources [95, 147].

3.9 Conclusion

Even though cloud computing may never be able to compete against dedicated infrastructures such as supercomputers and high performance clusters for running tightly coupled parallel programs with strong performance requirements, it is getting a lot of interest from scientists because of its flexibility.

In this chapter, we presented experimental results of creating large scale elastic execution platforms on top of multiple clouds using the sky computing federation approach. To perform these experiments, we used Nimbus clouds deployed on two experimental testbeds, Grid'5000 in France and FutureGrid in the USA. From resources of these clouds, we built large scale sky computing platforms, using the ViNe virtual network and the Nimbus contextualization tools. We studied the performance of this approach by running the BLAST scientific application on top of the Hadoop MapReduce framework inside these sky computing platforms. To our knowledge, the few other studies measuring the performance of multi-cloud scientific computations did not reach the same scale as our work. The results showed that federating cloud resources at a large scale can bring important execution speedups. However, further analysis is required to understand the exact factors limiting their scalability and propose solutions to improve the performance of these systems.

During our experiments, we determined that virtual machine image propagation performance was the limiting factor for the efficient creation and elastic extension of large scale sky computing platforms. To increase the scalability of these infrastructures, we proposed two improved propagation mechanisms, based on a broadcast chain and on copy-on-write volumes. We implemented these mechanisms in the Nimbus cloud toolkit and evaluated their performance, showing that they can drastically reduce the propagation time compared to a centralized approach. Future work includes comparing these mechanisms with alternative propositions such as LANTorrent and scp-wave.

Finally, we showed that sky computing platforms can be made elastic by dynamically extending them with new resources in order to accelerate computations. This elasticity presents great potential for increasing the resource usage of federated clouds or for reacting to changing requirements such as job deadlines. Future directions of research include work on autonomous management of the elasticity of such platforms, depending on user and application requirements. While our work was limited to sky computing platforms using Hadoop, future work would need to study elasticity in a more general context, independent of the application framework.

We demonstrated these experiments at the OGF-29 meeting in June 2010, which was covered by the iSGTW online publication [73]. We presented them in the poster session of the TeraGrid 2010 conference [184], and published in a special issue of ERCIM News on cloud computing [185]. We also contributed a book chapter in the *Advances in Parallel Computing* series [149].

Chapter 4

Resilin: Elastic MapReduce for Private and Community Clouds

Contents

4.1 Introduction **72**
4.2 Amazon Elastic MapReduce **72**
4.3 Architecture and Implementation **73**
4.3.1 Multi-Cloud Job Flows 76
4.4 Evaluation **77**
4.4.1 Deployment Time 78
4.4.2 Execution Time 79
4.4.3 Cloud Storage Performance 81
4.4.4 Financial Cost 82
4.5 Related Work **83**
4.6 Conclusion **84**

The flexibility and resource control offered by Infrastructure as a Service clouds come with a penalty. Expertise in building execution platforms from virtual machine instances is required to fully exploit these infrastructures. Potential cloud computing users, such as developers or scientists, may not have this expertise. In this chapter, we present Resilin, a system creating and managing execution platforms for running MapReduce computations. Resilin implements the Amazon Elastic MapReduce web service API while using resources from private and community clouds. Resilin takes care of provisioning, configuring and managing cloud-based Hadoop execution platforms, potentially using multiple clouds. In this chapter, we describe the design and implementation of Resilin. We also present the results of a comparative evaluation with the Amazon Elastic MapReduce system. This work was realized in collaboration with Ancuţa Iordache, Master student at West University of Timişoara, Romania, during her Master internship in the Myriads team.

4.1 Introduction

As presented in Section 2.2.1.2, the MapReduce programming model offers a simple way of performing distributed computation over large data sets. This model can be used by anyone using the Apache Hadoop framework.

The Apache Hadoop [1] project develops a free and open source implementation of the MapReduce framework. However, managing a Hadoop cluster requires expertise, especially when scaling to a large number of machines. Moreover, users who want to perform MapReduce computations in cloud computing environments need to instantiate and manage virtual resources, which further complicates the process.

To lower the entry barrier for performing MapReduce computations in the cloud, Amazon Web Services provides Elastic MapReduce (EMR) [58]. Elastic MapReduce is a web service to which users submit MapReduce jobs. The service takes care of provisioning resources, configuring and tuning Hadoop, staging data, monitoring job execution, instantiating new virtual machines in case of failure, etc.

However, this service has a number of limitations. First, it is restricted to Amazon EC2 resources. Users are not able to use Elastic MapReduce with resources from other public, private, or community clouds, which may be less expensive or even free of charge. This is especially true for scientists who have access to community clouds administrated by their institution and dedicated to scientific computing [131, 179]. Moreover, Elastic MapReduce is provided for an hourly fee, in addition to the cost of EC2 resources. This fee ranges from 17% to 21% of the price of on-demand EC2 resources. It is impossible to use a different virtual machine image than the one provided by Amazon, which is based on Debian Lenny 5.0.8. Finally, some users may have to comply with data regulation rules, forbidding them from sharing data with external entities like Amazon.

In this chapter, we present Resilin, a system implementing the Elastic MapReduce API with resources from any EC2-compatible cloud, such as private and community clouds. Resilin allows users to execute MapReduce computations on any EC2-compatible infrastructure, and offers more flexibility: users are free to select different types of virtual machines, different operating systems or newer Hadoop versions. The goal of Resilin is not only to be compatible with the Elastic MapReduce API. We also explore a feature going beyond the current Amazon Elastic MapReduce offering: performing MapReduce computations over multiple distributed clouds.

4.2 Amazon Elastic MapReduce

Amazon Elastic MapReduce is one of the products of Amazon Web Services. After signing up and providing a credit card number, users can submit MapReduce jobs through the AWS management console (a web interface), through a command line tool, or by directly calling the Elastic MapReduce API (libraries to access this API are available for several programming languages, such as Java and Python).

Users execute Hadoop jobs with Elastic MapReduce by submitting job flows, which are sequences of Hadoop computations. Before starting the execution of a job flow, Elastic MapReduce provisions a Hadoop cluster from Amazon EC2 instances. Each job flow is mapped to a dedicated Hadoop cluster: there is no sharing of resources between

job flows. A Hadoop cluster created by Elastic MapReduce can be composed of three kinds of nodes:

- the master node, which is unique, acts as a meta data server for HDFS (by running the NameNode service) and schedules MapReduce tasks on other nodes (by running the JobTracker service),
- core nodes provide data storage to HDFS (by running the DataNode service) and execute map and reduce tasks (by running the TaskTracker service),
- task nodes execute map and reduce tasks but do not store any data.

By default, a Hadoop cluster created by Elastic MapReduce is composed of one master node and several core nodes[1]. At any time during the computation, a Hadoop cluster can be resized. New core nodes can be added to the cluster, but core nodes cannot be removed from a running job flow, since removing them could lead to HDFS data loss. Task nodes can be added at any time, and removed without risk of losing data: only the progress of MapReduce tasks running on the terminated task nodes will be lost, and the fault tolerance capabilities of Hadoop will trigger their re-execution on other alive nodes.

After the Hadoop cluster has booted, the service executes bootstrap actions, which are scripts specified by users. These actions allow users to customize an Elastic MapReduce cluster to some extent. Amazon provides several pre-defined bootstrap actions, to modify settings such as the JVM heap size and garbage collection behavior, the number of map and reduce tasks to execute in parallel on each node, etc. Users can also provide custom bootstrap actions by writing scripts and uploading them in S3. Once a Hadoop cluster is ready for computation, Elastic MapReduce starts executing the associated job flow. A job flow contains a sequence of steps, which are executed in order. Internally, each step corresponds to the execution of a Hadoop program (which can itself spawn multiple Hadoop jobs). After all steps are executed, the cluster is normally shut down. Users can ask for the cluster to be kept alive, which is useful for debugging job flows, or for adding new steps in a job flow without waiting for cluster provisioning and paying extra usage (in Amazon EC2, instance cost is rounded up to the hour, which means a 1-minute and a 59-minute job flow cost the same price).

Typically, input data is fetched from Amazon S3, a highly scalable and durable storage system provided by Amazon. Intermediate data is stored in HDFS, the Hadoop Distributed File System. Output data is also saved in Amazon S3, since the termination of a Hadoop cluster causes its HDFS file system to be destroyed and all the data it contains to become unavailable.

4.3 Architecture and Implementation

Like the Amazon Elastic MapReduce service, Resilin provides a web service acting as an abstraction layer between users and Infrastructure as a Service (IaaS) clouds. Figure 4.1 presents how Resilin interacts with other components of a cloud infrastructure. Users execute client programs supporting the Elastic MapReduce API to interact with Resilin

[1] If a user requests a Hadoop cluster composed of only one node, the same virtual machine performs the roles of master node and core node.

(for instance, a Python script using the boto [5] library). Resilin receives Elastic MapReduce requests in the form of HTTP calls to the web service API, and translates them in two types of actions:

- interactions with an IaaS cloud using the EC2 API, to create and terminate virtual machine instances,
- remote connections to virtual machines using SSH, to configure Hadoop clusters and execute Hadoop programs.

To retrieve input data and store output data, Hadoop clusters interact with a cloud storage repository through the S3 API.

Resilin implements most actions supported by the Amazon Elastic MapReduce API. Users are able to execute the following actions:

RunJobFlow Submits a job flow. This request describes the Hadoop program to run with its arguments (specifying input and output data location). It also specifies the number and type of virtual machines to use.

DescribeJobFlows Queries job flow statuses.

AddJobFlowSteps Adds new steps to an existing job flow.

AddInstanceGroups/ModifyInstanceGroups Change the number of resources by adding or modifying instances groups.

TerminateJobFlows Terminates job flows.

However, only JAR-based and streaming Hadoop jobs are currently available in Resilin. We do not yet support job flows written using Hive and Pig, two SQL-like languages for data analysis.

We now describe how each type of Elastic MapReduce action is handled. When a new job flow request (RunJobFlow) is received by Resilin, its parameters are validated, and the service contacts an EC2-compatible IaaS cloud on behalf of the user to provision a Hadoop cluster. This is performed by making a RunInstances request through the EC2 API, asking for the number and type of virtual machines specified by the user. This request also includes the name of the virtual machine image to use. We rely on a customized Debian Squeeze image containing a Hadoop installation, which is to be uploaded to the cloud prior to the first instantiation. By leveraging the EC2 API, Resilin can make use of resources from clouds managed by open source toolkits supporting the EC2 API, such as Nimbus, OpenNebula, and Eucalyptus.

Once a Hadoop cluster is provisioned, Resilin connects to the virtual machines, using SSH, to execute bootstrap actions specified by the user, configure the Hadoop cluster (specifying the HDFS NameNode address, the MapReduce JobTracker address, etc.), and start the Hadoop daemons.

After Hadoop has been configured, the associated job flow is started and each step in the job flow is executed by invoking the `hadoop` command on the master node. Each step specifies the locations of its input and output data. These references can be S3 URLs: Hadoop supports reading and writing directly to Amazon S3. It would be technically possible to configure the Hadoop clusters created by Resilin to access input data from

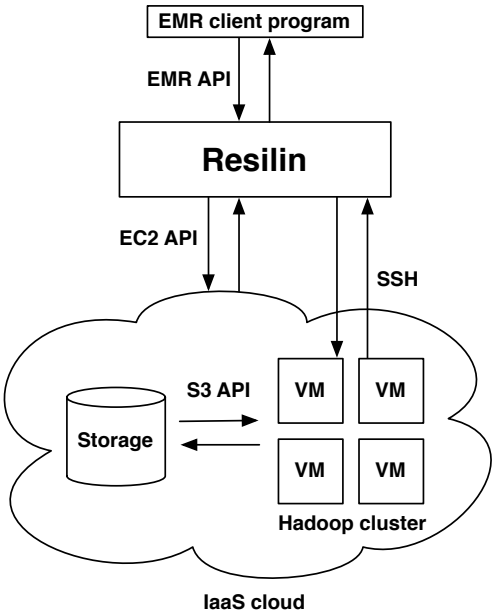


Figure 4.1: Overview of Resilin and its interactions with other cloud components

Amazon S3, like in Amazon Elastic MapReduce. However, this is ineffective for two reasons. First, bandwidth limitations between Amazon S3 and the private or community IaaS cloud running the Hadoop cluster drastically limit performance. Second, outgoing traffic from Amazon S3 is charged to customers, which would incur a high cost when performing MapReduce computations on large data sets.

We assume that clouds used by Resilin have a storage service available, like Walrus or Cumulus (see Section 2.3.3.2). Furthermore, Cumulus can be installed as a standalone system, making it available for any cloud deployment. Resilin can use these storage services to provide Hadoop with input data and to store output data. To make it possible, we extended the S3 protocol support of Hadoop 0.20.2. In addition to being able to specify URLs in the form of `s3n://bucket/key`, users can also provide URLs such as `cumulus://bucket.host:port/key`. When such URLs are detected, the library used by Hadoop to interact with S3 (JetS3t) is set up to contact the Cumulus server running on `host:port`. Additionally, we had to implement support for partial GET requests (HTTP Range header) in Cumulus. Hadoop uses this type of HTTP request to download subsets of input files to each task node.

Two types of steps are supported by Resilin. The first type, a custom JAR, is simply the location of a JAR and its required arguments. When executed, this JAR submits jobs to Hadoop. The JAR URL can be a S3 or Cumulus location. In this case, Resilin first downloads the JAR to the master node of the Hadoop cluster, using the `hadoop fs -copyToLocal` command, as Hadoop does not support directly executing a JAR from a remote location. The second type, streaming jobs, is defined by a mapper program, a reducer program, and the locations of input and output data. Both programs can be stored in S3 or Cumulus, and can be written in any programming language (Java, Ruby, Python, etc.). They apply their computation on data coming from standard input and stream their result to standard output. To run a streaming step, the Hadoop command

is invoked with the hadoop-streaming JAR included in the Hadoop MapReduce framework. We determined that Amazon made modifications to Hadoop to be able to fetch the mapper and reducer programs from S3 and add them to the Hadoop distributed cache (in order to provide them to all nodes). We did not make such modifications to the Hadoop source code. Instead, we rely on bootstrap actions to download the mapper and reducer programs to the master node. The Hadoop framework manages the distribution of these programs from the master node to the task nodes.

Resilin monitors the execution of the Hadoop computation. When the execution of a step is finished, the status of the job flow is updated, and the next step starts running. When all steps have been executed, the job flow is finished, which triggers termination of the cluster (unless the user requested the cluster to be kept alive). As long as a job flow has not terminated its Hadoop cluster, new steps can be queued for execution using the AddJobFlowSteps API request. To terminate the cluster, virtual machine instances are destroyed using the TerminateInstances EC2 API call.

Elasticity is managed through the AddInstanceGroups and ModifyInstanceGroups of the Elastic MapReduce API. Users of Resilin make these API requests to add or remove resources from an existing execution platform. New resources are automatically integrated in the platform to start executing computations.

Resilin is implemented in Python and is approximately 1,500 lines of code. It uses the Twisted [45] framework for receiving and replying to HTTP requests, the boto [5] library to interact with clouds using the EC2 API, and the paramiko [37] library for executing commands on virtual machines using SSH.

4.3.1 Multi-Cloud Job Flows

Besides targeting compatibility with the Amazon Elastic MapReduce API, we are experimenting with an additional feature in Resilin: the ability to execute job flows with resources originating from multiple clouds. With Amazon Elastic MapReduce, executing MapReduce computations across multiple Amazon EC2 regions does not present a lot of interest. The large number of resources available in each region makes it possible to deploy large Hadoop clusters in a single data center[2]. However, in the usage context of Resilin, many users would have access to several moderately sized private or community clouds. These clouds can be federated using the the *sky computing* approach [130], as presented in Chapter 3.

Although creating MapReduce clusters distributed over several clouds may not be efficient for all types of MapReduce jobs, because of the large amounts of wide area data transfer that it accesses and generates [81], it can be interesting for specific types of MapReduce jobs. For example, Matsunaga et al. [150] have shown that, for small numbers of resources, multi-cloud BLAST computations with MapReduce present good scalability with low performance degradation. In Chapter 3, we showed that with a larger and more distributed set of resources, the scalability of these computations is reduced, but can still provide interesting speedups. Bringing this idea further, Luo et al. present a MapReduce framework dedicated to executions on multiple sites [145].

[2]When scaling to very large clusters, Amazon EC2 can start running out of resources. In March 2011, the Cycle Computing company published details about how they provisioned a 4096-core cluster in the Amazon EC2 US East region [94]. When they scaled up the number of resources, 3 out of the 4 availability zones of the region reported Out of Capacity errors.

Resilin supports multi-cloud job flows by allowing users to dynamically add new resources from different clouds to a running job flow. As an example, let us assume that a user has access to two different clouds, *Cloud A* and *Cloud B*, and that *Cloud A* is her default choice. After a job flow has been started on *Cloud A*, Resilin will accept requests for adding new instances (AddInstanceGroups) with an instance type also specifying the cloud to use: instead of `m1.small`, the instance type of the request would be `m1.small@CloudB`. After new virtual machines are provisioned from *Cloud B*, they are configured to become resources of the cluster running in *Cloud A*. This addition is managed seamlessly by Hadoop. When Hadoop daemons are started in the newly provisioned virtual machines of *Cloud B*, they contact the master node in *Cloud A* to join the cluster and start receiving MapReduce tasks to execute.

4.4 Evaluation

For validating our implementation, we compare executions of the same job flows on Amazon Elastic MapReduce, using Amazon EC2 resources, and on Resilin, using resources provisioned from a cloud deployed on the Grid'5000 experimental testbed [79] with Nimbus Infrastructure version 2.8 [27, 128].

In Amazon EC2, we used High-CPU Medium instances (c1.medium) from the US East (Northern Virginia) region. These instances are supplied with 2 virtual cores whose speed is equivalent to 2.5 EC2 Compute Units each (one EC2 Compute Unit corresponds to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor), 1.7 GB of memory, 350 GB of instance storage, and are running 32-bit operating systems. As mentioned in other performance studies of Amazon EC2 [124], the same type of instances can be hosted on physical machines with different processors. By examining the content of `/proc/cpuinfo`, we determined that the physical processors of our c1.medium instances were either Intel Xeon E5410 running at 2.33 GHz or Intel Xeon E5506 running at 2.13 GHz.

On Grid'5000, we used physical machines part of the *genepi* cluster in Grenoble. These machines have 2 Intel Xeon E5420 QC processors (each with 4 cores running at 2.5 GHz), 8 GB of memory, and a Gigabit Ethernet connectivity. They were running a 64-bit Debian Lenny 5.0.4 operating system with a Linux 2.6.32 kernel, and were using QEMU/KVM version 0.12.5 as hypervisor. The virtual machines created on these physical machines were dual-core virtual machines, with 2 GB of memory, and were running a 32-bit Debian Squeeze 6.0.1 operating system with a Linux 2.6.32 kernel.

We evaluated the respective performance of each virtual machine type by running the benchmark tests of `mprime` [19]. Table 4.1 reports the mean, standard deviation, minimum and maximum of the average time required for computing a single thread 8192K FFT on each type of virtual machine. The results were obtained by running 5 iterations of the `mprime` benchmark on 3 instances of each virtual machine type, for a total of 30 runs. In the case of the c1.medium virtual machines, 2 instances had a 2.33 GHz processor and 1 had a 2.13 GHz processor.

These measurements show the lack of performance predictability of Amazon EC2 instances, already reported in other studies [124]. Not only did we observed different performance between instances of the same type, but we also experienced substantial performance variability within the same instance. For example, one instance performed 2 consecutive runs of `mprime` with the following results for the 8192K FFT: 220.829 ms

Instance type	c1.medium	Nimbus VM
Mean (ms)	206.587	177.241 (14.2% faster)
Standard deviation	6.247	0.181
Minimum (ms)	197.998	176.821 (10.7% faster)
Maximum (ms)	220.829	177.55 (19.6% faster)

Table 4.1: CPU performance of the virtual machine types used in our experiments, measured with `mprime` (single-thread 8192K FFT)

and 199.572 ms, a performance loss of 9.626%, even though these benchmarks runs were executed seconds apart. This behavior is likely caused by other instances sharing the same physical machine, which produced noise in the system and influenced the performance of our own instances.

Contrarily to Amazon EC2, the virtual machines executing in Nimbus clouds on Grid'5000 showed stable performance measurements, both within one virtual machine and among multiple virtual machines. When performing these benchmark runs, each virtual machine was executed on a dedicated physical machine, which minimized the noise produced in the system.

To compare Resilin with Amazon Elastic MapReduce, we separate the evaluation of a job flow execution in two parts. First, we compare the deployment time, which includes provisioning a virtualized Hadoop cluster and performing its configuration. Second, we compare execution time, from the submission of a Hadoop computation until its completion. These experiments are run with various cluster sizes. In each experiment, we had a dedicated master node. We used the following numbers of core nodes: 1, 2, 4, and 8. Thus, the total number of instances were 2, 3, 5, and 9.

Additionally, we perform a synthetic benchmark of the cloud storage repository by measuring its read performance. The cloud storage repository is an important part of the system, as it is used to store input and output data. Finally, we report the cost of the instances used on Amazon Elastic MapReduce.

4.4.1 Deployment Time

The deployment time corresponds to the time taken from the submission of a RunJobFlow request to the API until the Hadoop cluster is ready to start executing an application. This includes provisioning a virtualized Hadoop cluster and performing its configuration. The provisioning of the cluster contains two main phases: propagating the virtual machine images to the hypervisor nodes and starting the virtual machines. The deployment time of a Hadoop cluster is independent of the application submitted for execution.

To compare the deployment performance of Elastic MapReduce and Resilin, we submit job flows on both services and compare the time from the submission of the job flow until the start of the Hadoop computation. This time is computed as the difference between the CreationDateTime and the ReadyDateTime of the job flow, as reported by the DescribeJobFlows API request.

In the case of Amazon, the EC2 instances have a 5 GB root file system (the instances also have 350 GB of storage from local disks). However, since details of the Amazon EC2 architecture are not available, we cannot know how much of this data is propagated on

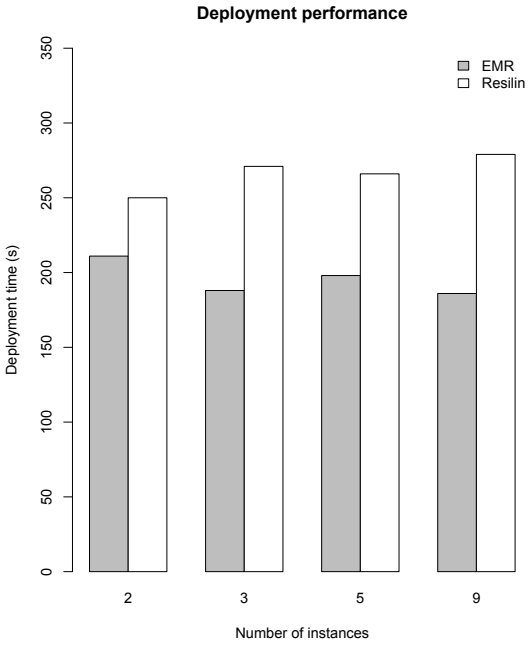


Figure 4.2: Deployment time of Hadoop clusters on Amazon Elastic MapReduce and Resilin with different number of instances

the network, or if hypervisors have a local cache of popular virtual machine images.

In the case of the Nimbus cloud, the deployment time is mostly dependent of the propagation of virtual machine images to the hypervisor nodes. The size of our virtual machine image is 4 GB. LANTorrent is used as the image propagation mechanism (see Section 3.5.5).

Figure 4.2 shows the deployment time for different cluster sizes. Deployment time in Amazon EC2 is fairly stable, even though it can be perturbed by other users of the infrastructure, like the CPU performance presented earlier. Resilin also presents a stable deployment time. This can be explained by the efficient propagation mechanism used by LANTorrent, which scales almost linearly. However, Resilin is constantly slower than Elastic MapReduce by approximately one minute. We believe that the deployment process of Resilin can be improved by several techniques:

- propagation performance can be improved by optimizing the size of the virtual machine image, or by using compression or caching,
- configuration performance can be improved by reducing the number of SSH connections to the virtual machines (Resilin currently does one connection per configuration action, while all actions could be done by executing a single script).

4.4.2 Execution Time

To compare the execution performance of Elastic MapReduce and Resilin, we submit the same job flows, each composed of only one step, on both services and compare the time from the start of the requested Hadoop computation until its completion. This time is computed as the difference between the StartDateTime and the EndDateTime of the step,

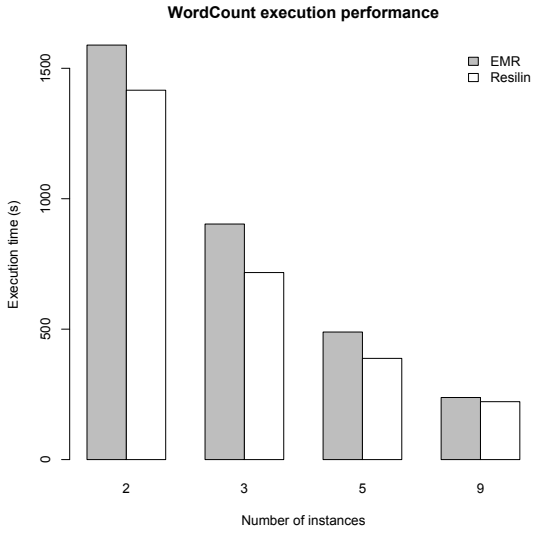


Figure 4.3: Execution time of Word Count on Amazon Elastic MapReduce and Resilin with different number of instances

as reported by the DescribeJobFlows API request. We evaluate 2 different applications: Word Count and CloudBurst. Resilin was configured with Hadoop settings similar to those used by Elastic MapReduce. For instance, the number of map and reduce tasks were chosen to be the same: 4 map tasks and 2 reduce tasks per instance.

4.4.2.1 Word Count

To evaluate the performance of a streaming job, we ran a Word Count program on a collection of text files. Word Count computes the number of occurrences of each word in a text collection. The Amazon Elastic MapReduce Word Count sample [63] uses a copy of *The World Factbook* as input data. This book is split in 12 files, for a total of 18 MB. However, the content is not split evenly: the last 3 files are small in comparison to the first 9 files which contain most of the data. This text collection is small: Hadoop was created to process data at a much larger scale. We copied 100 times the first 9 files to create a larger collection of 900 files, for a total of 1.8 GB.

Figure 4.3 presents the results of the execution time of Word Count. For all cluster sizes, Resilin is faster than Amazon Elastic MapReduce. This is not surprising, since we determined that the virtual machines used on Grid'5000 had higher CPU performance. As the cluster size is increased, the difference of performance between Elastic MapReduce and Resilin becomes smaller. We will analyze if this can be explained by contention to access the Cumulus server.

4.4.2.2 CloudBurst

CloudBurst [190, 61] is one of the sample workloads provided by Amazon for Elastic MapReduce. It implements a parallel read-mapping algorithm optimized for mapping next-generation sequence data to the human genome and other reference genomes. It is invoked as a custom JAR (`cloudburst.jar`), with the locations of input and output data

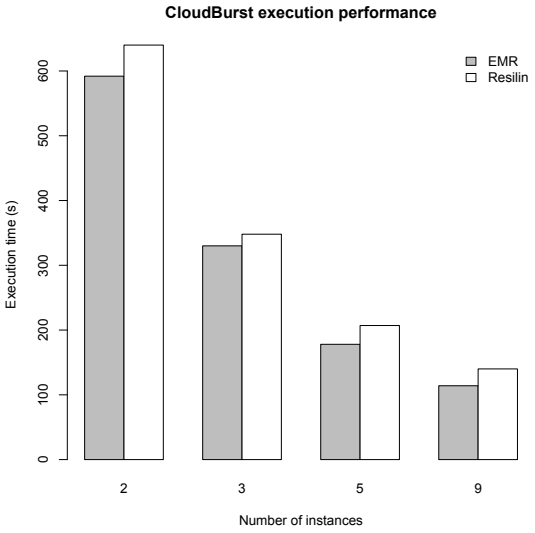


Figure 4.4: Execution time of CloudBurst on Amazon Elastic MapReduce and Resilin with different number of instances

and parameters for the execution (including the number of map and reduce tasks). We use the exact same program, input data and parameters as those provided by Amazon as a sample application (240 maps and 48 reduces). All input and output data is stored in the cloud storage repository.

Figure 4.4 presents the results of the execution time of CloudBurst. For this application, Resilin is consistently slower than Elastic MapReduce. Since Amazon uses a modified version of Hadoop with many tuned parameters, we will analyze if this difference in the execution environment is the source of the performance difference.

4.4.3 Cloud Storage Performance

In this experiment, we compare the performance of the cloud storage repository used in Amazon Elastic MapReduce and Resilin. We created a 1.8 GB file, uploaded it to S3 and to a Cumulus repository in Grid'5000, and compared the time required to read its content from Hadoop. The read operation was performed with the `fs -cat` option of the `hadoop` command using the `s3n` protocol. Output was sent to `/dev/null` to avoid any slowdown from I/O operations on the local hard drive.

Figure 4.5 compares the performance results obtained in Elastic MapReduce with one client and in Resilin (with Cumulus acting as a storage repository), first with one client and then with 2 parallel clients. Results were obtained by executing 10 iterations of the read process. The plot reports the minimum, lower quartile, median, upper quartile, and maximum values observed. As with CPU benchmarks, we observed important performance variations when reading data stored in Amazon S3 from an EC2 instance. The slowest read operation (4.2 MB/s) took more than 6 times longer than the fastest one (26.4 MB/s), while the mean bandwidth of the 10 iterations was 10.14 MB/s. When scaling up the number of parallel readers, we did not notice any strong change of performance behavior, as Amazon S3 is presumably backed by thousands of machines serving data to clients, with data replicated in several locations.

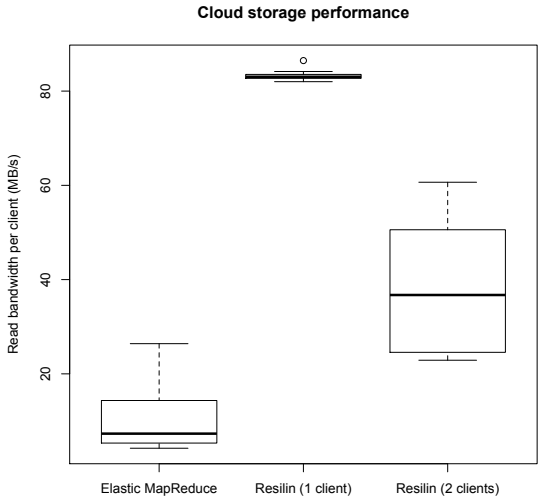


Figure 4.5: Comparison of cloud storage performance between Amazon Elastic MapReduce and Resilin (with the Cumulus cloud storage repository)

With Resilin and the Cumulus storage repository, reading the file from one client at a time lead to a very stable bandwidth averaging 83.37 MB/s. Contrarily to Amazon S3, the Cumulus server is implemented with a single machine, which becomes a bottleneck when more clients are accessing the cloud storage in parallel. Figure 4.5 shows that for 2 parallel readers, the per-client bandwidth is reduced. This result was expected, since the resources of the Cumulus server are shared between the 2 clients. A notable behavior of Cumulus is that the bandwidth was not shared fairly between the 2 clients. While one client was averaging a bandwidth of 24.76 MB/s, the other was receiving data at 51.42 MB/s. Also, the cumulative bandwidth with 2 clients (76.18 MB/s) shows some performance degradation compared to the one-client scenario.

If more than 2 clients are doing parallel requests to a single Cumulus server, the performance of each client will degrade further. We note that Cumulus can be configured to scale horizontally across many nodes [76] by using parallel file systems such as GPFS. Using this setup, it should be possible to retain a good level of performance even when increasing the number of Cumulus clients. However, we did not test such a configuration in our experiments.

4.4.4 Financial Cost

An important difference between a public cloud like Amazon Web Services and a private or community cloud is the charging model. A public cloud directly charges its customers for resources used, while a private or community cloud is paid for by organizations (companies, universities, laboratories, etc.) and is free of charge for its direct users.

In Table 4.2, we report the price of the Elastic MapReduce clusters used in our experiments. The price is composed of the cost of the Amazon EC2 instances plus the cost of the Elastic MapReduce service. In our case, the EC2 instances were c1.medium instances of the US East region, billed at $0.17 per hour. For this type of instance, the Elastic MapReduce service costs $0.03 per instance per hour. We note that each partial

Number of instances	2	3	5	9
Instance cost (per hour)	$0.34	$0.51	$0.85	$1.53
Elastic MapReduce cost (per hour)	$0.06	$0.09	$0.15	$0.27
Total cost (per hour)	$0.40	$0.60	$1	$1.8

Table 4.2: Per-hour financial cost of Elastic MapReduce computations on Amazon with c1.medium instances in the US East region (not including S3 cost)

hour is fully billed: a job flow executing for 4 minutes costs a full hour.

The prices reported in Table 4.2 do not include the cost of the Amazon S3 storage. Amazon charges for three types of resource consumption in S3. Users are charged for the amount of storage space they use, for the number of requests to the S3 service, and for outgoing data transfer. There is no charge for data transferred between S3 and EC2 in the same region. In the case of Elastic MapReduce, this is generally the case: users provision instances in the data center where their data is available. However, for users with very large data sets, the storage space consumption and the number of requests could lead to significant costs. For instance, storing 1 PB of data in the US Standard Region of S3 costs more than $100,000 per month.

We cannot compare the cost of Elastic MapReduce computations with the real cost of resources of a private or community cloud, since the latter depends on many variables, including the costs of hardware resources, power, maintenance contracts, administrative staff, Internet network connectivity, etc.

4.5 Related Work

Related work on running MapReduce computations in the cloud has been focused on adapting MapReduce to cloud characteristics, either by creating completely new implementations of MapReduce, or by modifying existing systems. For instance, Gunarathne et al. [111] proposed AzureMapReduce, a MapReduce implementation built on top of Microsoft Azure cloud services [152]. They leverage several services offered by Azure, such as Azure Queues for scheduling tasks, Azure blob storage for storing data, etc. Compared to a Hadoop cluster, their solution does not have a centralized master component. Thus, their implementation leads to increased levels of fault-tolerance and flexibility. Similarly, Liu and Orban [141] created Cloud MapReduce, a MapReduce implementation leveraging services from the Amazon Web Services platform, with data stored in Amazon S3, and synchronization and coordination of workers performed with Amazon SQS and Amazon SimpleDB. These contributions are orthogonal to our objectives. While their goal is to build MapReduce implementations taking advantage of features offered by specific cloud services, we aim to bring an easy to use MapReduce execution platform to many different cloud implementations. Furthermore, to our knowledge, no open source cloud implements the Azure API or the SQS and SimpleDB interfaces, making it impossible to run these MapReduce implementations outside of Microsoft Azure and Amazon Web Services.

Another type of proposition has been to take advantage of spot instances available in Amazon EC2 [56]. Spot instances are virtual machines that leverage unused capacity of the Amazon EC2 infrastructure. The price of spot instances varies over time, depending

on the infrastructure load. Users bid for a maximum price they are willing to pay for spot instances. As long as the current spot instance price stays lower than their bid, they are charged with the current spot price. When the spot instance price rises above their bid, their instances are terminated. Chohan et al. [85] propose using spot instances to reduce the cost of execution of MapReduce jobs. They study bidding strategies to optimize the effectiveness of using spot instances. Liu [140] uses spot instances in the Cloud MapReduce system. This system handles instance terminations more gracefully than a Hadoop cluster by saving computation progress before the spot instances are terminated. On August 18, 2011, Amazon unveiled spot instance support for Elastic MapReduce. However, no automatic bidding strategy or extended fault-tolerance is provided: customers are responsible for specifying how they want to use spot instances, knowing that it can lead to job failure. We do not yet support spot instances in Resilin. However, since Nimbus provides an implementation of spot instances, we consider implementing this feature in the future.

Another axis of research has been focusing on optimizing the number, type and configuration of resources allocated for a MapReduce computation. Kambatla et al. [127] study how to configure the number of map and reduce tasks running in parallel on each compute node. They propose to analyze the behavior of an application and to match it against a database of application signatures containing optimal Hadoop configuration settings, in order to derive efficient configuration parameters.

4.6 Conclusion

In this chapter, we presented Resilin, a system for creating execution platforms for MapReduce computations. Resilin implements the Amazon Elastic MapReduce API, with resources from any EC2-compatible cloud, such as private and community clouds. Resilin takes care of provisioning Hadoop clusters and managing job execution, allowing users to focus on writing their MapReduce applications rather than managing cloud resources. Resilin uses the EC2 API to interact with IaaS clouds, making it compatible with most open source IaaS toolkits. We evaluated our implementation of Resilin using Nimbus clouds deployed on the Grid'5000 testbed and compared its performance with Amazon Elastic MapReduce, both for deployment and execution time. Results show that Resilin offers a level of performance similar to Elastic MapReduce. An initial version of this work has been presented in the poster session of the CCA 2011 workshop [182].

The contribution of this work is a system for easily creating MapReduce execution platforms on top of cloud infrastructures. In the future, we will use the multi-cloud functionalities offered by Resilin as a framework for research on multi-cloud MapReduce computations. Future work will include determining the performance limitations of such computations, for different types of MapReduce applications. By identifying the behavior of computations in these distributed environments, we can research improvements to the MapReduce model in order to deal with more distributed resources. Additionally, we want to study how Resilin can be made autonomous. Currently, like with the Amazon Elastic MapReduce service, users have to decide how many resources are required and where to provision them. Instead, Resilin should make these decisions autonomously according to higher level user requirements, such as job deadlines or cost constraints.

On the implementation side, we plan to increase compatibility with Amazon Elastic MapReduce with support for Hive and Pig job flows, and to verify its behavior with other EC2-compatible clouds such as Eucalyptus and OpenNebula. We plan to release Resilin as open source software, allowing users with access to EC2-compatible clouds to benefit from the features of this system.

Part III

Dynamic Execution Platforms with Inter-Cloud Live Migration

Chapter 5

Network-level Support for Inter-Cloud Live Migration

Contents

5.1 Introduction 90
5.2 Problem Statement 90
5.3 Related Work 92
5.3.1 Mobile IP 93
5.3.2 Overlay Networks 93
5.3.3 Virtual Private Networks (VPN) 93
5.3.4 ARP and Tunneling 93
5.4 Architecture of ViNe 94
5.5 Live Migration Support in ViNe 96
5.5.1 Automatic Detection of Live Migration 96
5.5.2 Virtual Network Reconfiguration 96
5.5.3 ARP and tunneling mechanisms 97
5.5.4 Routing Optimizations 99
5.6 Implementation 99
5.7 Conclusion 100

Inter-cloud live migration of virtual machines is beneficial in a number of scenarios. However, in order to keep communications from and to virtual machines uninterrupted, inter-cloud live migration must rely on support from the network layer to handle the dynamic relocation of virtual machines between different networks. In this chapter, we detail this problem, present existing solutions, and describe the mechanisms we propose for detecting live migration and ensuring efficient communication after migration. We developed a prototype implementation of these mechanisms for the ViNe virtual network developed at University of Florida. This work was realized in collaboration with José Fortes, Andréa Matsunaga, and Maurício Tsugawa, during a 3-month visit in the ACIS laboratory at University of Florida, Gainesville, USA, in summer 2010.

5.1 Introduction

The increased popularity of Infrastructure as a Service (IaaS) offers users with a large choice of cloud computing infrastructures for creating execution platforms to run their applications. These infrastructures can differ in performance, availability, and cost. In the current ecosystem, execution platforms created from cloud resources are not fully dynamic. They can grow and shrink with the elastic capabilities of clouds, as showed in Part II of this dissertation. However, mechanisms to move computations between different cloud infrastructures are limited. When applications or operating systems support mechanisms to save and restart computations, it is possible to provision a completely new execution platform on a different cloud, save the computation state, transfer it to the new platform, and restart the computation. However, most applications and operating systems do not support such mechanisms.

Being able to move execution platforms between clouds would be interesting in a number of scenarios. For instance, in a hybrid cloud context where cloudbursting (see page 47) was used to offload a computation to a public cloud, a concurrent computation can terminate and free resources of the private cloud. It would be beneficial to move the offloaded computation back to the private cloud, in order to reduce its cost (assuming that using the private cloud is less expensive than using the public cloud).

Another scenario is related to the cost of cloud resources. Currently, most cloud resources are available for a fixed price. However, as cloud computing becomes more popular, we believe that more resources will start to follow a dynamic price model. For instance, Amazon EC2 already provides virtual machines for a price that varies over time (spot instances [56]). If the cloud computing ecosystem tends toward a dynamic market were most resources are obtained at a variable price, it will be interesting to have the capability to move computations between different clouds.

Moving computations between clouds could be realized by virtual machine migration, a mechanism working at the hypervisor level. This mechanism is transparent to the operating system and applications executing in the virtual machine. Migration can be performed in a non-live approach, by pausing the execution of the virtual machine, taking a snapshot, transferring it to another hypervisor, and resuming it. Live migration transfers virtual machine state while it continues executing, which reduces the pause time compared to the non-live approach.

However, as mentioned in Section 2.1.3.4, live migration between different networks is not possible without support from the network infrastructure. The problem is the same in the context of non-live migration. Although we restrict our work to live migration because of its increased performance, the same solutions could be applied to non-live migration of virtual machines.

In this chapter, we study the issue of supporting live virtual machine migration between different IP networks, which is one of the problems preventing inter-cloud live migration. We explain this issue, review existing solutions, present our contributed mechanisms, and describe their prototype implementation for the ViNe virtual network.

5.2 Problem Statement

In this section, we explain the problems occurring when live migration is performed between different IP networks without support from the networking layer. This expla-

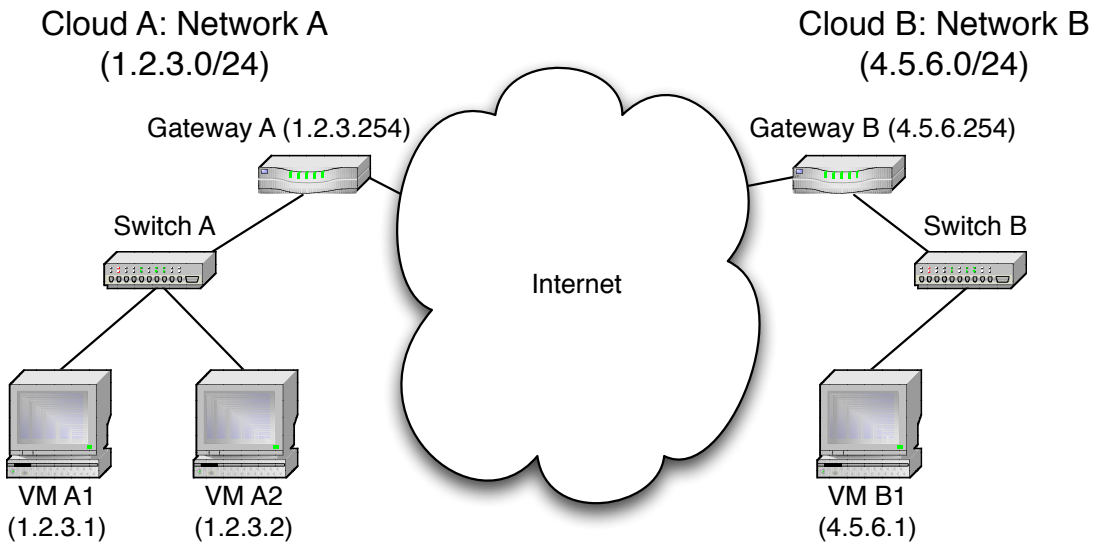


Figure 5.1: Example of two separate cloud IP networks

nation will help the reader to understand the related work and our contributions.

Figure 5.1 presents a simple view of two separate clouds, cloud A and cloud B, each with its own IP network (respectively network A and network B). These two cloud networks are connected through the Internet. This infrastructure will be used to present the problems occurring at the network level when using inter-cloud live migration. The first network, network A, contains the following resources:

- Gateway A (IP address 1.2.3.254) serves as gateway for network A,
- Virtual machine A1 (IP address 1.2.3.1) uses gateway A as its default gateway,
- Virtual machine A2 (IP address 1.2.3.2) uses gateway A as its default gateway.

The second network, network B, contains the following resources:

- Gateway B (IP address 4.5.6.254) serves as gateway for network B,
- Virtual machine B1 (IP address 4.5.6.1) uses gateway B as its default gateway.

All machines connected to the same switch communicate directly at layer 2 (Ethernet). For virtual machines of network A to communicate with VM B1 of network B, they send their data to gateway A. Gateway A forwards the traffic through the Internet, where it goes through a number of routers. When the data arrives to gateway B, it is delivered to VM B1. Communications from VM B1 to virtual machines of network A take the opposite route.

Let us assume that VM A2 is live migrated to the cloud corresponding to network B. VM A2 keeps it IP address unchanged: if the IP address was changed, it would break all communications, and potentially make applications behave incorrectly (for example, if they are configured to specifically bind to the original IP address). In this scenario, a number of issues arise.

Communications between the migrated virtual machine and its original network The migrated virtual machine may require to communicate with its original network (in our example, network A). It may need to contact other virtual machines composing the same execution platform (if the platform was not fully migrated to the same cloud). It may need to access services that cannot be relocated to cloud B, such as storage servers. Communications between VM A2 to VM A1 were performed using the layer 2 infrastructure of network A: Ethernet frames were sent between the two machines without going through the IP network infrastructure. After VM A2 has been migrated to network B, it will not be aware that it has changed IP network. As such, to communicate with VM A1, it will continue to send Ethernet frames using VM A1's MAC address (which is kept in VM A2's ARP cache) as destination address. Since VM A1 is not present on network B, these frames will not be delivered and will be lost. After several unsuccessful data transmissions from VM A2 to VM A1, the ARP cache information may be marked as invalid. In this case, VM A2 will start broadcasting ARP requests on network B, asking for the MAC address linked to VM A1's IP address (1.2.3.1). Alas, VM A2 will not receive any reply since VM A1 is not on the same network. The same events happen on network A: VM A1 will not be able to contact directly VM A2 after it has been migrated.

Communications between the migrated virtual machine and its new network The migrated virtual machine may also require to communicate with its new network (in our example, network B). It may need to contact other virtual machines composing the same execution platform, if the platform is spanning multiple clouds. Let us assume that VM A2 needs to communicate with VM B1. Whether or not VM A2 has already communicated with VM B1 in the past, its network configuration tells it that communications to network B must go through gateway A. VM A2 will send Ethernet frames with VM B1's IP address as IP destination, but with gateway A's MAC address as Ethernet destination. Similarly to the previous case, these frames will not be delivered. Even though VM A2 and VM B1 are now on the same physical network, VM A2 will never send ARP requests for VM B1, since VM A2 thinks that it is on network A. Similarly, VM B1 will contact gateway B when trying to communicate with VM A2.

Communications between the migrated virtual machine and the outside world The migrated virtual machine may require to contact other networks than networks A and B. It may need to communicate with clients, services or resources located in another cloud, or any machine connected to the Internet in general. This scenario is similar to the previous one: VM A2 will fail to send traffic to gateway A. Traffic sent from the Internet to VM A2 will arrive to gateway A and fail to be delivered.

5.3 Related Work

The issues created by inter-cloud live migration, described in the previous section, are equivalent to the problem of device mobility between different IP networks. This problem has been studied in the past, and solutions have been proposed. In this section, we present existing solutions that can be applied to our context (whether they are targeting mobility of machines in general or live virtual machine migration in particular).

5.3.1 Mobile IP

Mobile IP [173, 174] is a standard mechanism providing support for roaming of mobile devices on IPv4 and IPv6 networks. Each mobile device is assigned a permanent IP address known as the *home address*, belonging to its *home network*. Wherever Mobile IP devices are located, they are always reachable using their home address.

When a mobile device joins a new network (called a *foreign network*), it receives a new IP address belonging to this network: its *care-of address*. Connections to the mobile device are routed through an entity of the home network called the *home agent*, which tunnels them to the mobile device using the care-of address. Care-of addresses are announced by an entity called the *foreign agent*, residing on the foreign network. The home agent is required to stay reachable in order to continue forwarding incoming connections to the mobile device.

A downside of this approach is that Mobile IP support is required in home and foreign networks, and in IP stacks of mobile devices. In our scenario, any operating systems deployed in IaaS virtual machines would need to support Mobile IP.

5.3.2 Overlay Networks

Overlay networks are peer-to-peer networks built on top of a physical network infrastructure (e.g. the Internet). Several systems create independent and isolated virtual IP networks above overlay networks, including VIOLIN [224], *i3*/OCALA [126, 201], and IPOP [105]. The overlay networks route communication between the nodes of the virtual IP networks. Decoupling the low-level IP network (tied to physical network infrastructures) from the higher level IP network allows to support mobility. When virtual machines are migrated, the overlay system is reconfigured to forward traffic to the correct peer.

A downside of these systems is that all communications go through the peer-to-peer overlay, even when it is not strictly required, which adds overhead to communication performance.

5.3.3 Virtual Private Networks (VPN)

Another type of migration support relies on the presence of VPN connecting multiple sites. These VPNs unify multiple network together. Wood et al. present a mechanism that dynamically modifies VPN configurations in the presence of live migrations [221]. However, VPNs with reconfiguration capabilities are not always available, or they restrict their reconfiguration to network administrators.

5.3.4 ARP and Tunneling

The Address Resolution Protocol (ARP) is used to discover mappings between network layer addresses (IP addresses) and link layer addresses (MAC addresses for Ethernet). When a host needs to communicate with another host on the same network, it broadcasts an ARP request to discover the link layer address to be used as destination address.

Proxy ARP is a mechanism in which a host answers ARP requests on behalf of a host not present on the network. This technique is increasingly used in inter-cloud live migration, in order to capture traffic intended for a migrated virtual machine. After

being captured, traffic is forwarded to the correct destination using tunneling [114, 146, 195, 220]. An advantage of this mechanism is that it is supported by most operating systems that could be deployed in a cloud.

The inter-cloud live migration support proposed in this chapter is based on this technique. Additionally, we present how to detect when live migrations are happening, and describe how a virtual network infrastructure can be reconfigured in this scenario. We also describe our prototype implementation in the ViNe software.

5.4 Architecture of ViNe

ViNe is a virtual network system, developed at University of Florida, that aims to provide all-to-all connectivity in complex network infrastructures, such as those found in grids and clouds. These infrastructures can include networks using private IP addressing or firewalls to control access to resources. ViNe can support multiple isolated virtual networks sharing the same overlay.

The architecture of ViNe is based on a user-level software router. Software router processes are in charge of creating and maintaining the network overlay, and tunneling traffic between physical networks. We refer to a node running the ViNe software as a ViNe router (VR). Each VR acts as a gateway for machines connected to the same subnetwork.

Routing decisions in the ViNe overlay are based on tables managed by each VR. Two types of tables are maintained by VRs: Local Network Description Tables (LNDTs) and Global Network Description Tables (GNDTs). Each LNDT lists the nodes a VR is responsible for: nodes on the same subnetwork that are allowed to participate in the ViNe virtual network. GNDTs stores information about the physical networks and VRs participating in a ViNe virtual network: like a network routing table, it contains necessary data to forward network packets to their correct destinations.

Figure 5.2 illustrates a ViNe deployment. Two physical networks, 1.2.3.0/24 and 192.168.0.0/24, are participating in a ViNe virtual network. Each network contains a VR acting as a gateway between the local network and the rest of the virtual network. Each VR has a copy of the GNDT and of its corresponding LNDT (LNDT A for VR A, LNDT B for VR B).

When a packet is sent from 1.2.3.1 to 192.168.0.1, traffic is routed through VR A. First, VR A verifies that the source host is authorized to participate in the virtual network, by checking its LNDT. Since 1.2.3.1 is present in the LNDT, the packet is authorized. Then, VR A consults its GNDT: like a routing table, the most specific network entry is selected (in this case, 192.168.0.0./24), and the packet is forwarded to the corresponding VR, using the public address of VR B, 4.5.6.7. The ViNe overlay is used to tunnel packets, over UDP or TCP connections. Once received by VR B, the packet is delivered to 192.168.0.1. Note that even though network B is using NAT with private IP addresses, routing packets at the level of the virtual network allows all-to-all connectivity between nodes from networks A and B.

The configuration information (GNDT and LNDT) can be updated dynamically by exchanging information between VRs. This allows ViNe to react to changes in the network environment. This is the case for virtual machine live migration, as described in the next section.

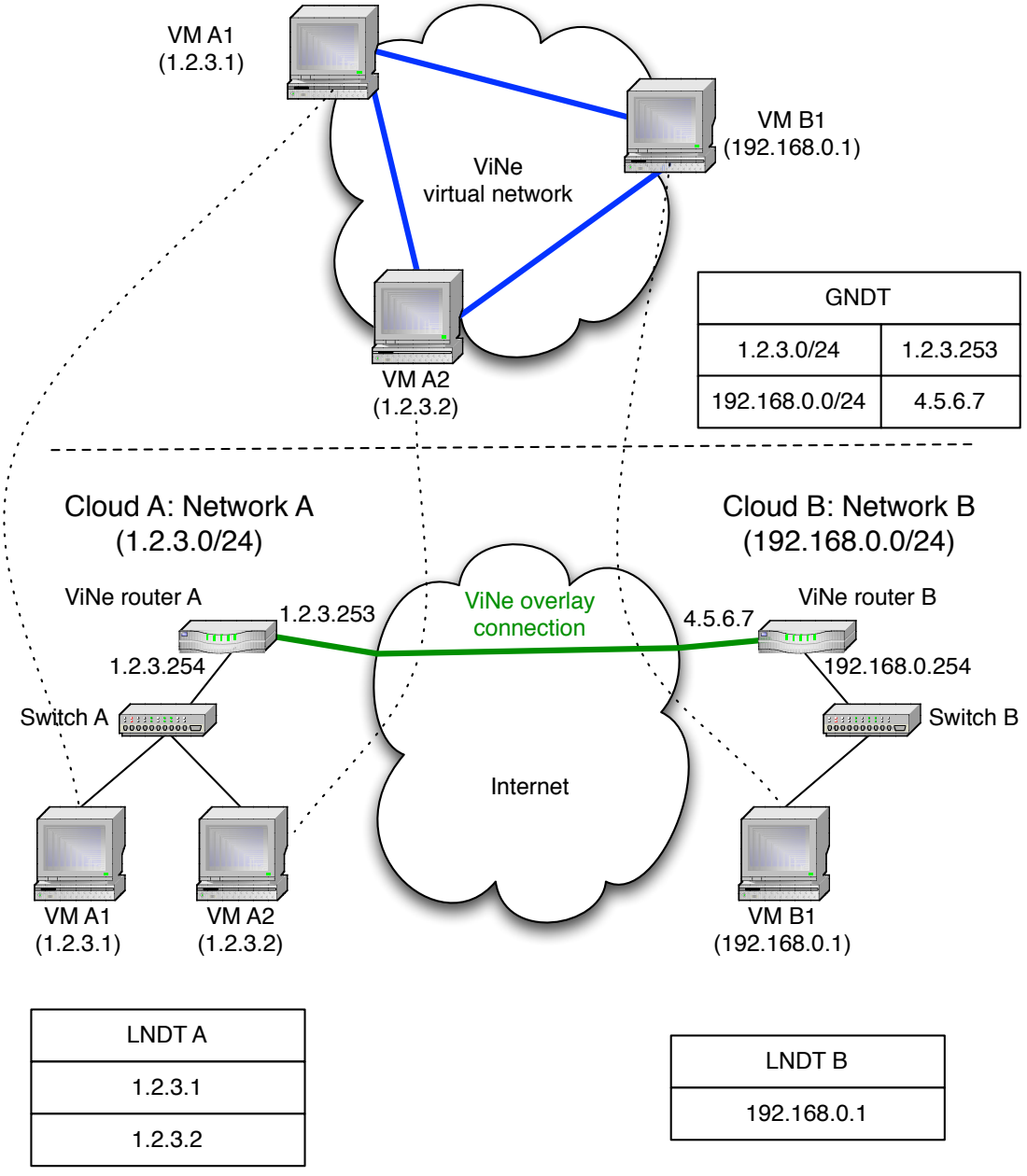


Figure 5.2: Example deployment of ViNe

5.5 Live Migration Support in ViNe

In this section, we present the design and prototype implementation of the mechanisms we proposed for adding virtual machine live migration support to ViNe. These mechanisms rely on ARP and tunneling techniques to allow virtual machines to migrate between two different clouds without interrupting their communications.

We present our live migration support in ViNe by describing the following three mechanisms:

Automatic detection of live migration By automatically detecting live migration events, ViNe does not require bidding with the cloud management stacks, making it more independent and portable.

Virtual network reconfiguration After detecting live migration events, the virtual network is reconfigured to learn the position of the migrated virtual machine.

ARP-based traffic redirection After migration, ARP mechanisms are used to redirect traffic that the migrated virtual machine sends and receives.

5.5.1 Automatic Detection of Live Migration

When a live virtual machine migration occurs, the destination hypervisor sends Ethernet frames on behalf of the migrated virtual machine in order to reconfigure the physical infrastructure (as explained in Section 2.1.3.4). Both KVM and VMware ESX send Reverse ARP (RARP) packets when a migrated virtual machine resumes execution, while Xen sends ARP packets. We modified the ViNe routing software to capture these packets in order to discover newly migrated virtual machines. RARP packets provide VRs with the MAC addresses of migrated virtual machines. To reconfigure the virtual network, additional information is required. We present how this information is retrieved in the next section, when describing the virtual network reconfiguration mechanism.

5.5.2 Virtual Network Reconfiguration

After a live migration has been detected, the virtual network needs to be reconfigured to route to the correct subnetwork any traffic originating sent or received by the migrated virtual machine. First, the IP address of the migrated virtual machine is discovered. When a VR detects a live migration, it contacts other VRs in the overlay network in order to discover the IP address of the migrated virtual machine. Each VR continuously maintains a table mapping IP and MAC addresses of virtual machines connected to its physical subnetwork, allowing to reply to such requests. This table is built from the ARP messages captured by each VR.

Once the IP address of the migrated virtual machine has been discovered, several actions are taken to reconfigure the virtual network. These actions are initiated by the VR of the destination site. First, the GNDT is modified, by inserting a host-only network (a network with a /32 netmask) corresponding to the migrated IP address and associating it with the destination site VR. This GNDT change is propagated from the destination site to all other VRs participating in the virtual network. The GNDT is used like a network routing table, and host-only networks have priority over larger networks. This allows the host-specific network to be selected in place of its source site network. This

action is the only one performed on VRs not directly involved in the migration, i.e. VRs that are not responsible for the source or destination network (i.e. that cannot reach the migrated virtual machine directly through link layer communication).

Additionally, the LNDTs of the source and destination VRs need to be updated. The migrated IP is removed from the source VR LNDT and added to the destination VR LNDT. Figure 5.3 illustrates the state of the ViNe deployment presented in Figure 5.2, after live migrating the virtual machine with IP address 1.2.3.2 from cloud A to cloud B.

5.5.3 ARP and tunneling mechanisms

After virtual network reconfiguration, the last step is to start intercepting and redirecting traffic from and to the migrated virtual machine. Before migration, this machine is communicating directly at link layer with hosts of its physical network, including the source network VR. After migration, it keeps trying to communicate in the same way, which is not possible as both physical networks are separated.

To allow the migrated virtual machine to continue communicating, we use ARP mechanisms to redirect traffic to the ViNe routers. First, on the source physical network, packets destined to the migrated virtual machine are captured by the source site VR and forwarded to the destination site VR. To capture such traffic, the source site VR needs to become the link layer destination address for these packets. This address is determined through the ARP protocol, which populates a table on each host called the ARP cache. For instance, before migration, the ARP cache of a host on network A, such as VM A1, includes the following entry: `(1.2.3.2) at <MAC address of VM A2>`. To make VM A1 send the traffic intended for VM A2 to VR A, VR A broadcasts gratuitous ARP replies on network A after live migration. Gratuitous ARP replies are responses to ARP requests that were never made, which broadcast the fact that an IP address is reachable with a specific link layer address. When a gratuitous ARP reply is received by a host, it updates its ARP cache with the information contained in the ARP packet. In our example, VR A sends an ARP reply containing its own MAC address and the IP address 1.2.3.2. Once received by VM A1, its ARP cache is modified to change the aforementioned entry to: `(1.2.3.2) at <MAC address of VR A>`. Afterwards, any traffic sent to the migrated virtual machine from hosts in the source network will be delivered to the source site VR at link layer. The VR captures all this traffic and tunnels it through the ViNe infrastructure to the destination site VR, which delivers it to the migrated virtual machine.

On the destination site, the ARP cache of the migrated virtual machine needs to be updated. This ARP cache potentially contains the MAC addresses of all hosts on the source network. Each entry needs to be updated to associate all IP addresses of the source network with the MAC address of the destination site VR. This is done by making the destination site VR broadcast a gratuitous ARP reply for each host of the source network that was communicating with the migrated virtual machine. In order to know the list of hosts on the source network, the destination site VR queries the source site VR, which sends back the list of IP addresses on the network. For each IP address obtained, the destination site VR sends a gratuitous ARP reply with its MAC address. After these packets have been received by the migrated virtual machine, the ARP cache has been updated, and traffic from the virtual machine to hosts on the source network is sent to the destination VR at link layer.

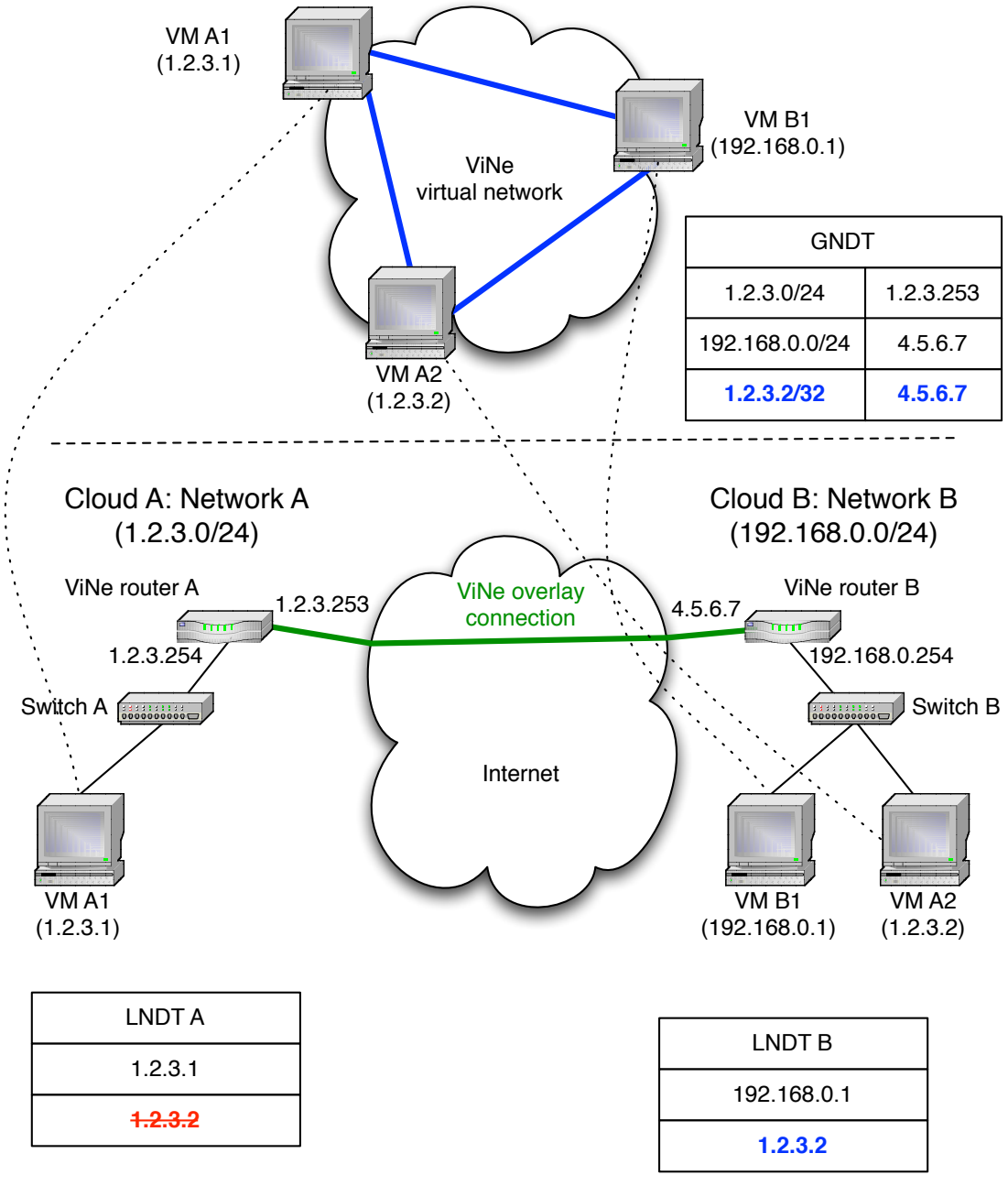


Figure 5.3: ViNe deployment after live migration
After live migration of the virtual machine with IP address 1.2.3.2 from cloud A to cloud B, the GNDT is modified to include a host-only network associated with VR B. Additionally, 1.2.3.2 is removed from LNDT A and added to LNDT B.

Additionally, the source site VR listens for ARP requests intended for the migrated virtual machine, and replies on behalf of it (as in proxy ARP). Similarly, the destination site VR listens for ARP requests originating from the migrated virtual machine, and replies on behalf of nodes of the source network.

5.5.4 Routing Optimizations

The mechanisms described in the previous section allow a virtual machine to be migrated between different networks without interrupting its communication. However, communication is non optimal. Any traffic between the migrated virtual machine and hosts of the destination network goes through the destination site VR. This is inefficient compared to direct communication. Moreover, migration can be prompted by a need for faster communication between the migrated virtual machine and hosts of the destination network.

To deal with this issue, we propose to configure virtual machines of a virtual network to use a single large subnetwork (for instance, 192.168.0.0/16). In reality, each physical network is assigned a dedicated part of this subnetwork (e.g. 192.168.0.0/24 and 192.168.1.0/24). As virtual machines are configured to be in the same large subnetwork, they send ARP requests to communicate with machines in other physical networks. Using proxy ARP, VRs respond to such requests with their own MAC addresses, and forward traffic to the correct destination using the ViNe infrastructure. When a virtual machine migration occurs, we use the same aforementioned mechanisms. Additionally, the destination network VR sends gratuitous ARP replies for each virtual machine on its LAN (including the migrated virtual machine), in order to update their ARP cache. This enables direct communication at link layer between virtual machines of the destination site, which allows to use the full available network bandwidth.

5.6 Implementation

Figure 5.4 presents the overall architecture of the implementation of our mechanisms in ViNe. In addition to the GNDT and LNDT, a new table is managed by each ViNe router: HardWare Description Table (HWDT). To populate and maintain this table, the packet capture module is extended to capture all ARP and RARP traffic arriving on the physical interface. This allows each ViNe router to be aware of the MAC addresses and IP addresses present on the network.

When a migration is detected (for instance when capturing a RARP packet), the VR first queries other VRs to discover the IP address of the migrated virtual machine, as described previously. These queries are replied by the HWDT of each VR. Then, the migration event is propagated to other VRs in the virtual network.

When a live migration event is received from another VR, the packet injection module builds gratuitous ARP replies and injects them on the physical network (as described in the previous section). Additionally, the GNDT and LNDT of each VR are updated.

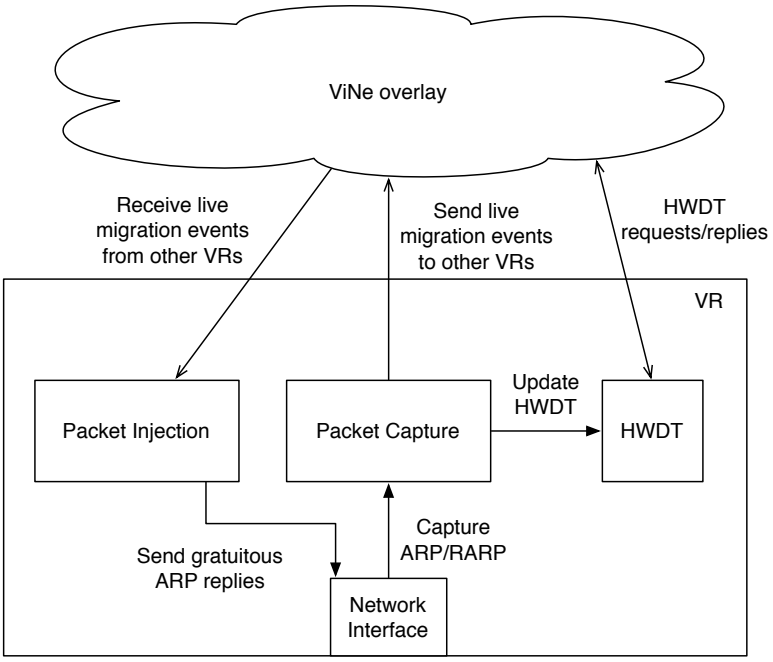


Figure 5.4: Architecture of our mechanisms implemented in ViNe

5.7 Conclusion

In this chapter, we presented the design of mechanisms to support inter-cloud live migration of virtual machines by reconfiguring virtual network infrastructures. By leveraging techniques based on ARP messages and reconfiguration of virtual networks, live migration can be realized without interrupting network communications. Our mechanisms detect live migration automatically from messages broadcasted on the network, making them independent of the cloud management infrastructure. Additionally, we proposed a routing optimization that allows to fully use network capabilities when virtual machines are located on the same physical network. We implemented a prototype of these mechanisms in the ViNe virtual network, developed at the University of Florida, and validated it experimentally. This work was published at the IEEE MENS 2010 workshop [209].

Future work will include performance evaluation of this prototype. We will measure several metrics: inter-cloud and intra-cloud network communication before and after live migration, detection time of live migration, and reconfiguration time. Additionally, it would be interesting to extend our contribution to other virtual networks than ViNe and study how different virtual networks can coordinate their reconfiguration mechanisms for live migration. This would allow clouds based on different virtual network solutions to benefit from inter-cloud live migration.

This contribution is one of the founding blocks for making execution platforms more dynamic. By allowing efficient inter-cloud live migration, execution platforms can be relocated between multiple clouds.

Chapter 6

Shrinker: Efficient Live Migration of Virtual Clusters over WANs

Contents

6.1 Introduction 102
6.2 Related Work 102
6.3 Architecture of Shrinker 103
6.3.1 Architecture Overview 103
6.3.2 Security Considerations 105
6.4 Implementation 106
6.4.1 Prototype Limitations 108
6.5 Evaluation 108
6.5.1 Evaluation Methodology 108
6.5.2 Evaluation with Different Workloads 109
6.5.3 Impact of Available Network Bandwidth 113
6.5.4 Impact of Deduplication Granularity 113
6.6 Conclusion 116

Inter-cloud live migration is a powerful mechanism to make execution platforms more dynamic, by allowing them to take advantage of changes in cloud computing infrastructures. However, the performance of live migration is limited in wide area networks, because of the amount of data that needs to be transferred over the network. This is further accentuated when migrating clusters of virtual machines instead of individual instances. In this chapter, we present Shrinker, a live migration system that leverages identical data among virtual machines to optimize the migration of virtual clusters over wide area networks. The design of Shrinker is based on distributed data deduplication and distributed content-based addressing. We implemented Shrinker in the KVM hypervisor and evaluated its performance on Grid'5000.

6.1 Introduction

The first live migration mechanisms were designed to operate solely in local area networks. As presented in Chapter 2 and 5, state of the art systems have been proposed to allow using live migration over WANs by migrating storage and network connections. However, the large amounts of data to migrate make live migration over WANs expensive to use, especially when considering migrations of virtual clusters rather than individual virtual machine instances.

Previous research showed that virtual machines running identical or similar operating systems have significant portions of their memory and storage containing identical data. In this chapter, we propose Shrinker, a live virtual machine migration system leveraging this common data to improve the efficiency of live virtual cluster migration between data centers interconnected by wide area networks. Shrinker detects memory pages and disk blocks duplicated in a virtual cluster to avoid sending multiple times the same content over WAN links. Virtual machine data is retrieved in the destination site with distributed content-based addressing. We implemented a prototype of Shrinker in the KVM hypervisor [132] and present a performance evaluation in a distributed environment. Our experiments show that it reduces both total data transferred and total migration time.

This chapter is organized as follows. First, we cover related work. Then, we present the design of Shrinker, based on distributed data deduplication and distributed content-based addressing. Finally, we describe our prototype implementation in the KVM hypervisor, and present the results of evaluations performed on Grid'5000.

6.2 Related Work

Numerous works have been proposed to improve live migration and optimize its performance metrics: total data transferred, total migration time, and downtime.

Optimizations include compression, delta page transfer, and data deduplication. Compression is an obvious method to reduce the amount of data transferred during live migration. Jin et al. [125] use adaptive compression to reduce the size of migrated data. Their system chooses different compression algorithms depending on memory page characteristics. Delta page transfer [113, 202, 221] optimizes the transmission of modified pages by sending difference between old pages and new pages, instead of sending full copies of new pages. Data deduplication [221, 226] detects identical data inside the memory and disk of an individual virtual machine and transfers this data only once.

All these systems were designed in the context of live migration of individual virtual machines, and do not take advantage of data duplicated across multiple instances. However, previous research [66, 112, 154, 216, 222] has shown that multiple virtual machines have significant portions of their memory containing identical data. This identical data can originate from having the same versions of programs, shared libraries and kernels used in multiple virtual machines, or common file system data loaded in buffer cache. These contributions leverage this identical data to decrease memory consumption of collocated virtual machines by sharing identical or similar memory pages. The same observation has been made for virtual machine disks [138, 160, 180], which is exploited to decrease storage consumption.

To our knowledge, Sapuntzakis et al. [189] were the first to leverage identical data between multiple virtual machines to improve migration. However, their work predates live migration and supported only suspend/resume migration, where the virtual machine is paused before being migrated. Additionally, they only took advantage of data available on the destination node, limiting the possibility of finding identical data. A similar approach was also proposed by Tolia et al. [205]. Recently, Deshpande et al. [98] proposed to use a combination of deduplication of identical memory pages with sub-page level deduplication and delta encoding of similar memory pages in order to improve the live migration of collocated virtual machines. Their prototype is implemented in the userland part of QEMU/KVM. Their approach is limited to leverage similarities found in virtual machines hosted on the same node, while Shrinker detects duplicated data among different hosts using distributed mechanisms.

6.3 Architecture of Shrinker

We propose Shrinker, a system that improves live migration of virtual clusters over WANs by decreasing total data transferred and total migration time. During live migration of a virtual cluster between two data centers separated by a WAN, Shrinker detects memory pages and disk blocks duplicated among multiple virtual machines and transfers identical data only once. In the destination site, virtual machine disks and memory are reconstructed using distributed content-based addressing. In order to detect identical data efficiently, Shrinker leverages cryptographic hash functions. These functions map blocks of data, in our case memory pages or disk blocks, to hash values of fixed size, also called digests. These functions differ from ordinary hash functions because they are designed to render practically infeasible to find the original block of data from a hash value, modify a block of data while keeping the same hash value, and find two different blocks of data with the same hash value. Using hash values to identify memory pages and disk blocks by content is interesting because hash values are much smaller than the data they represent. For instance, a 4 kB memory page or disk block is mapped to a 20 bytes hash value using the SHA-1 [161] cryptographic hash function, a size reduction of more than 200 times.

We first present the architecture of Shrinker, and then discuss security considerations caused by our use of cryptographic hash functions.

6.3.1 Architecture Overview

In a standard live migration of a virtual cluster composed of multiple virtual machines running on different hypervisors, each live migration is independent. Virtual machine content is transferred from each source host to the corresponding destination host, with no interaction whatsoever between the source hypervisors or between the destination hypervisors. As a consequence, when migrating a virtual cluster over a WAN, data duplicated across virtual machines is sent multiple times over the WAN link separating the source and destination hypervisors.

To avoid this duplicated data transmission, Shrinker introduces coordination between the source hypervisors during the live migration process. This coordination is implemented by a service, running in the source site, that keeps track of which memory pages and disk blocks have been sent to the destination site. Before sending a memory

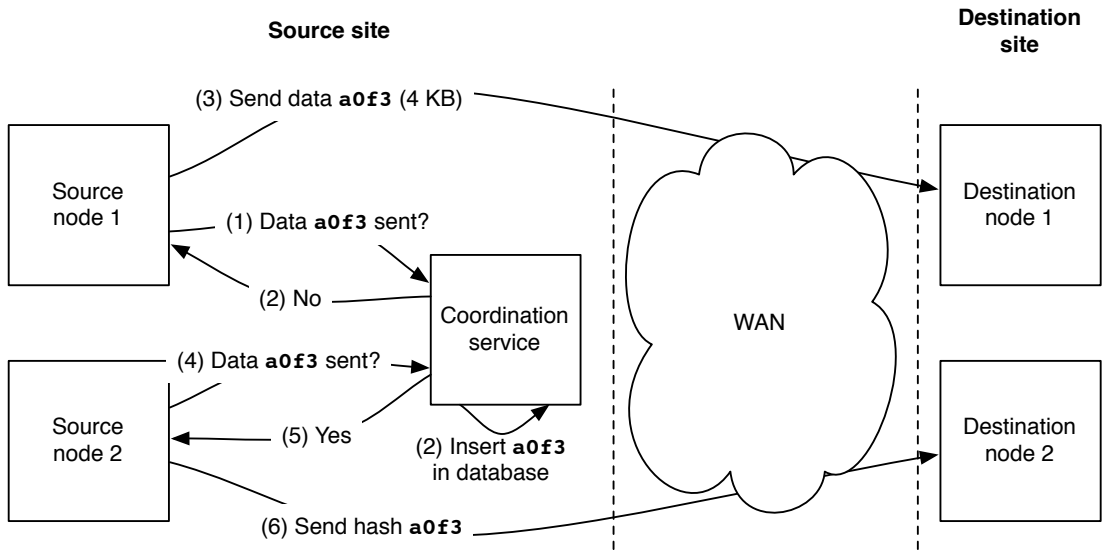


Figure 6.1: Distributed data deduplication algorithm

page or a disk block, a source hypervisor computes the hash value of this data and queries the service with this hash value. If no memory page or disk block with the same hash value has previously been sent to the destination site, the service informs the hypervisor that it should transfer the data. The service also registers the hash value in its database. Later, when the service receives a subsequent query for another memory page or disk block with the same hash value, it informs the querying hypervisor that the data has already been sent to the destination site. Based on this information, the hypervisor sends the hash value of the data to the destination host instead of the real content. This mechanism essentially performs data deduplication by replacing duplicated transmissions of memory pages and disk blocks by transmissions of much smaller hash values. Figure 6.1 illustrates a live migration between two pairs of hypervisors, with a cryptographic hash function creating 16-bit hash values[1].

Source node 1 has to send a memory page or disk block identified by the hash value `a0f3`. Instead of sending it directly like in a standard live migration, it first queries the coordination service (step 1). Since this is the first time that this data has to be sent to the destination site, the service informs source node 1 that it should send the data. It also updates its internal database to keep track of this hash value (step 2). After receiving the answer from the coordination service, source node 1 sends the data to destination node 1 (step 3). Afterwards, source node 2 queries the service for the same data (step 4). The service informs source node 2 that this data has already been sent to a node in the destination site (step 5), which prompts source node 2 to send the hash value `a0f3` (2 bytes) instead of the full content (4 kB) to destination node 2 (step 6).

Since destination nodes receive a mix of virtual machine data (memory and disk content) and hash values from source nodes, they need to reconstruct the full content of the virtual machine before resuming its execution. This is where distributed content-

[1]Note that this small hash value size is chosen to improve readability of the figure. In a real scenario, Shrinker cannot use such hash function, since 65,536 different hash values would likely create collisions even for virtual machines of moderate sizes.

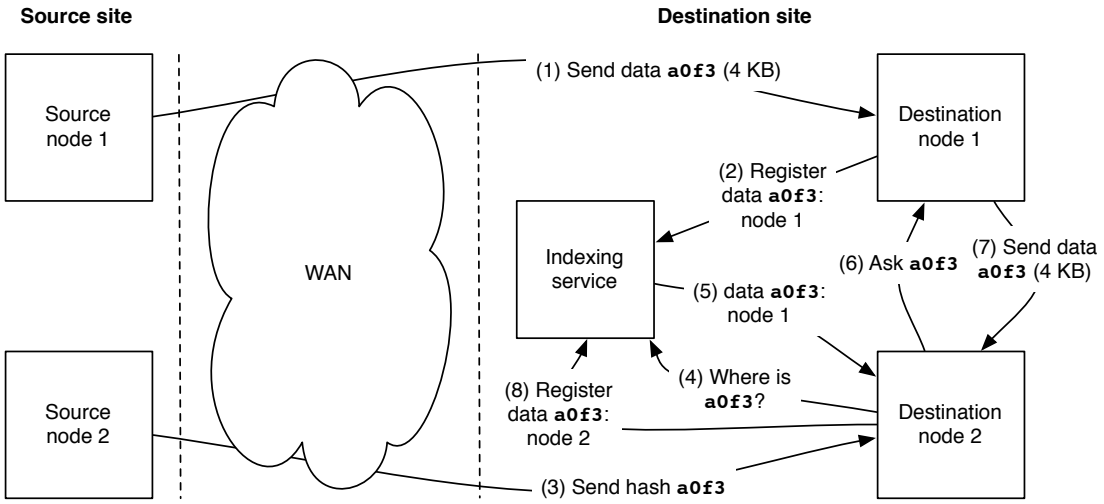


Figure 6.2: Distributed content-based addressing algorithm

based addressing is used. When a full memory page or disk block is received by a destination node, its hash value and the IP of the node are registered into an indexing service, running in the destination site. When destination nodes receive hash values from source nodes, they query the indexing service to discover a node that has a copy of the content they are requesting. After contacting a node and receiving a copy of the content, they register themselves in the indexing service for this data, which increases the number of hosts capable of sending this data to another hypervisor. This process is illustrated in Figure 6.2.

As in Figure 6.1, destination node 1 receives a memory page or disk block identified by hash `a0f3` (step 1). It registers itself as having a copy of this data in the indexing service (step 2). Afterwards, as in Figure 6.1, source node 2 sends hash value `a0f3` to destination node 2 (step 3). In order to get the data identified by this hash value, destination node 2 queries the indexing service to discover a host which has a copy of the corresponding content (step 4). The indexing service informs it that destination node 1 has a copy of this data (step 5). Destination node 2 asks destination node 1 for this content (step 6), receives it (step 7), and registers itself in the indexing server as having a copy (step 8).

Since the live migration is performed while the virtual machine is still executing, it is possible that data that was sent to the destination site has since been overwritten and is not accessible in any destination hypervisor. Additionally, it is possible that hash values arrive on destination nodes before the corresponding page or block has been registered in the indexing service. To solve this issue, all destination hypervisors open a communication channel with their corresponding source hypervisor. This channel is used to directly request data to the source node.

6.3.2 Security Considerations

The possible number of different 4 kB memory pages and disk blocks ($2^{4096\times 8}$) is higher than the number of possible hash values (2^{160} for SHA-1). As a consequence, the use of cryptographic hash functions opens the door to collisions: it is theoretically possible

that memory pages and disk blocks with different content map to an identical hash value. However, the properties offered by cryptographic hash functions allow us to use these hash values with a high confidence. The probability p of one or more collisions occurring is bounded by equation (6.1), where n is the number of objects in the system and b the number of bits of hash values [178]:

$$p \leq \frac{n(n-1)}{2} \times \frac{1}{2^b} \ . \tag{6.1}$$

If we consider a very large virtual cluster consisting of 1 exabyte (2^{60} bytes) of 4 kB memory pages and disk blocks migrated by Shrinker using the SHA-1 hash function, the collision probability is around 10^{-20}. This is considered to be much less than other possible errors in a computing system, such as data corruption undetectable by ECC memory. However, equation (6.1) gives the probability of an accidental collision. Although the theoretical number of operations to find a collision is approximately $2^{\frac{b}{2}}$ (birthday attack), attackers can exploit weaknesses of the hash function algorithm to find collisions more easily. For example, researchers have shown attacks against SHA-1 that decrease the number of operations to find collisions from 2^{80} to 2^{69} [218]. Assuming that finding collisions is possible, an attacker capable of storing arbitrary data in virtual machines (for instance, by acting as a client interacting with web servers) could inject colliding data in these virtual machines. After migrating them with Shrinker, memory content would be corrupted because two different pages would have been replaced by the same data. If such attacks become practically feasible in the future, Shrinker can use stronger cryptographic hash functions, such as those from the SHA-2 family. Even though these hash functions are more computationally expensive, Shrinker remains interesting as long as the hash computation bandwidth is larger than the network bandwidth available to each hypervisor. On the hardware used for our experiments, OpenSSL computes SHA-512 digests at more than 200 MB/s for 4 KB input size. This is larger than the bandwidth available with a Gigabit Ethernet interface.

6.4 Implementation

We implemented a prototype of Shrinker in the KVM hypervisor [132]. KVM is divided in two main parts. The first part is included in the Linux kernel, and is composed of two loadable kernel modules providing the low-level virtualization infrastructure: one is specific to the type of hardware support for virtualization (VT-x or AMD-V), and the other is generic and provides the core virtualization infrastructure. The second part is a modified version of QEMU [71] running in user space to provide higher level features, including virtual device emulation (disk, network, etc.) and live migration.

Our prototype is fully implemented in the user space component of KVM version 0.15.0 in approximately 2,400 lines of C code. We use Redis [188] version 2.4.2, a high performance key-value store, to implement the coordination and indexing service. Additional dependencies are OpenSSL, used to compute hash values, Hiredis, a C client library for Redis, and libev, a high-performance event loop library.

We now describe the implementation of our prototype. First, we focus on the source side, i.e. the hypervisor initiating a migration. Next, we present the implementation on the destination side, i.e. the hypervisor receiving a migration.

The standard migration procedure of KVM uses a pre-copy mechanism (see Section 2.1.3). It iterates over all memory pages to synchronize them with the destination hypervisor. First, all memory pages are sent. In later iterations, only modified pages are synchronized.

The first modification of the Shrinker implementation is to compute the cryptographic hash value of each page instead of sending it. One exception is done for pages that are composed of the same repeated byte value, for instance a page full of bytes with value zero. The transfer of these pages is already optimized by the original KVM code, which compresses the whole page to only one byte. It would be ineffective to use Shrinker in this case, as the digest size is always bigger than one byte.

As explained in Section 6.3, before sending a memory page, source hypervisors need to contact the coordination service to know if the same page as already been sent to the destination site. Performing this query in a sequential manner would drastically reduce the live migration throughput, because of the round-trip time between source hypervisors and the coordination service. Even with a 0.1 ms round-trip and assuming no server-side computation, the service would be limited to 10,000 requests per second. To overcome this problem, our implementation performs these queries in a pipelined manner. Queries for multiple memory pages are sent in parallel, and the decision of sending the full content or the hash is made when the answer is received. The same method is used on the destination site for hypervisors to locate memory content.

The coordination service is implemented using the Redis `SETNX` command, with a memory page hash value as key. If the hash is not already known by the service, it is registered and the return value notifies the source hypervisor that it was the first to register it. In this case, the hypervisor sends the full memory page like the original KVM code. Otherwise, the service notifies the source hypervisor that the hash was already registered. In this case, the hypervisor sends the hash value instead of the memory page to the destination hypervisor.

The indexing service is implemented using the Redis set data structure. For each digest of registered memory pages, there is a corresponding set which holds information about which hypervisors have copies of the page. When destination hypervisors register a memory page, they send a `SADD` command with the page hash as key and an IP/port pair as value. When other destination hypervisors need to get a page, they send a `SRANDMEMBER` command, which selects a random hypervisor in the set corresponding to the queried page and returns its IP/port information. After connecting on this IP and port, they query the content of the page. Finally, when a destination hypervisor does not hold any more copy of a page, it unregisters it with a `SREM` command.

To be able to deliver memory content to other nodes, each hypervisor keeps track of its memory pages in a local hash table. Pages are stored using their hash as key. As such, they can be found when a request for a particular hash value is received. Figure 6.3 presents the data structures used to keep track of local memory pages. Each memory page is identified by a cryptographic hash value, generated by a hash function. In this example, hash values are two hexadecimal digits long (one byte). Three pages are indexed in the hash table. Page 0 and page 2 are identical: their cryptographic hash value is AB. As such, they are indexed in the AB key of the hash table. Page 1 is indexed at the 54 key. The hash table buckets are created from a prefix of the keys, in this case 4 bits. An additional table references all pages in the hash table. This allows to retrieve the page structure inside the hash table after its content has changed (thus changing the

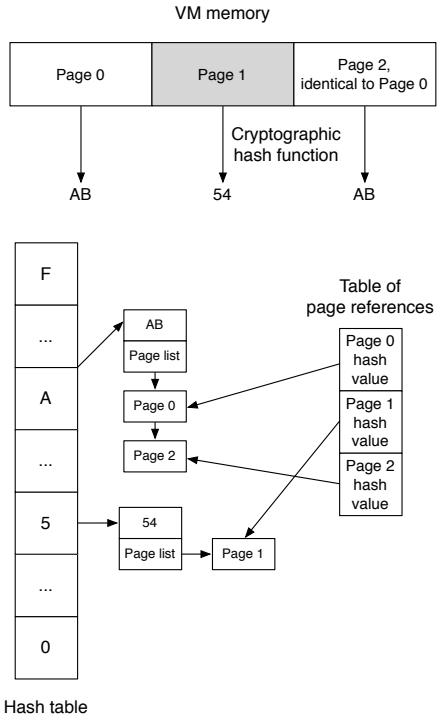


Figure 6.3: Data structures used to index local memory pages

associated hash value).

In our implementation, the length of the bucket array is equal to the number of memory pages of the virtual machine, in order to minimize the number of collisions in the hash table (different hash values with same prefix). The prefix is obtained by taking the first 32 bits of the hash value and dividing it by the number of buckets.

6.4.1 Prototype Limitations

Support for data deduplication of storage has not been fully finalized in our prototype. In all experiments presented in this chapter, we configured the destination nodes to access the virtual machine storage by connecting to the source node using a remote file system (SSHFS).

6.5 Evaluation

In this section, we present our experiments performed on the Grid'5000 testbed, and we analyze their results.

6.5.1 Evaluation Methodology

All experiments presented in this chapter were executed on the *parapent* cluster of the Grid'5000 site of Rennes. We used Carri System CS-5393B nodes supplied with 2 Intel Xeon L5420 processors (each with 4 cores at 2.5 GHz), 32 GB of memory, and Gigabit Ethernet network interfaces. Physical nodes and virtual machines used the x86-64 port

of Debian 5.0 (Lenny) as their operating system, with a 2.6.32 Linux kernel from the Lenny backports repository. Virtual machines were configured to use 1 GB of memory and one CPU core. Two instances of Redis were running to implement the coordination and indexing services, each on a dedicated node.

To study the performance of our prototype, we configured a dedicated node to act as a router between source and destination hypervisors. Using the `netem` Linux kernel module, this router emulated wide area networks with different bandwidth rates. The traffic to access the virtual machine storage from the destination hypervisors is not routed through this machine. As such, all the performance results presented in this chapter are restricted to live migration of memory data.

We evaluated the performance of live migration while running different workloads:

Idle virtual machines We migrate idle virtual machines after they have fully booted their operating system. Since the operating system data is common to all workloads, this serves as a reference for the minimum amount of data that can be deduplicated in any workload.

Memory-intensive workload We migrate virtual machines executing the CG MPI program from the NAS Parallel Benchmarks. The CG benchmark finds the largest eigenvalue of a random sparse matrix using an inverse power method. This workload is memory-intensive: it allocates memory to store the random matrix and frequently updates it. Since the matrix data is random, it can be seen as a worst case workload, since memory allocated for the matrix cannot be deduplicated.

I/O-intensive workload DBENCH reproduces the file system load of a Samba file server. This benchmark is highly I/O intensive. Even though we do not deduplicate data from storage in the prototype used for these experiments, this workload also exercises memory through the Linux page cache.

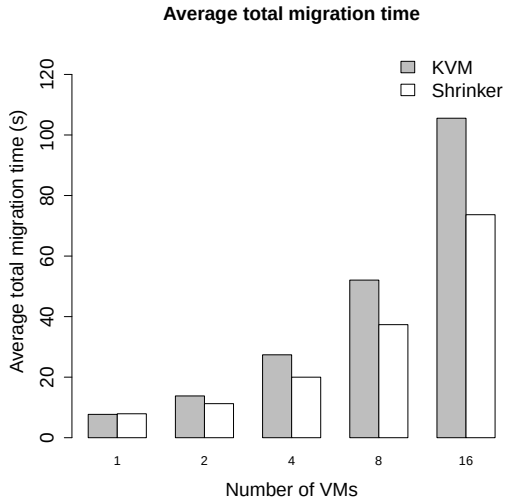
We evaluated performance metrics of live migration during the execution of these workloads, measuring total migration time and total data transferred by live migration. We discovered that the downtime caused by a live migration in KVM was overly important because of a miscalculation of the available bandwidth. For this reason, we do not report numbers for downtime, as they may be not representative of a correct behavior. However, we investigated the issue and submitted a patch to the KVM community.

For all workloads, live migration is performed 30 seconds after the start of the computation, in order to let benchmarks reach their typical memory usage.

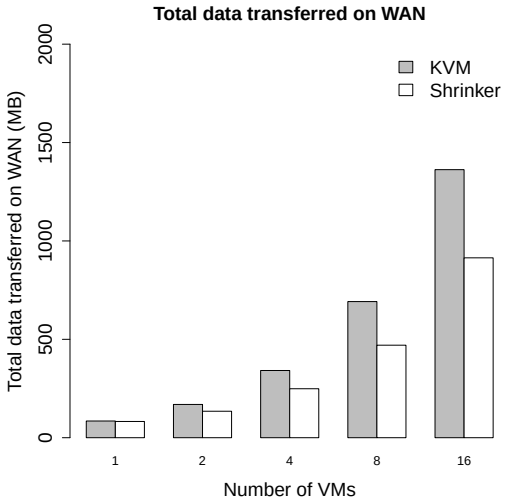
6.5.2 Evaluation with Different Workloads

6.5.2.1 Idle Virtual Machines

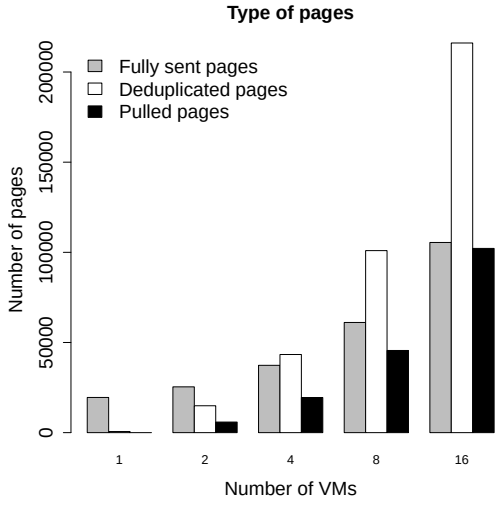
In this section, we study the performance of Shrinker when migrating idle virtual machines over a 100 Mbps wide area link. Figure 6.4(a) and Figure 6.4(b) respectively present the average total migration time and the total data transferred on the wide area network. Figure 6.4(c) details how the different types of pages are divided. Fully sent pages correspond to memory pages that are sent to destination hypervisors when they are identified as the first copies in the coordination service. Deduplicated pages correspond to memory pages that are identified as already sent in the coordination service



(a) Average total migration time



(b) Total WAN data transferred



(c) Types of pages transferred

Figure 6.4: Shrinker performance for idle virtual machines

and are replaced by a transmission of their hash value. The sum of the number of fully sent and deduplicated pages correspond to the total number of pages normally transferred by the live migration (in this case, approximately 20,000 pages per VM, or 80 MB). Pulled pages correspond to the subset of deduplicated pages that could not be found at the destination site and were requested from the source hypervisors.

We can make the following observations:

- When migrating only one virtual machine, a very small amount of pages are deduplicated (around 600). This corresponds to intra-VM deduplication, from memory pages with duplicated content present in a single virtual machine. Consequently, the amount of data transferred is only slightly reduced. However, the migration time is slightly increased, which can be explained by the additional overhead of the deduplication.

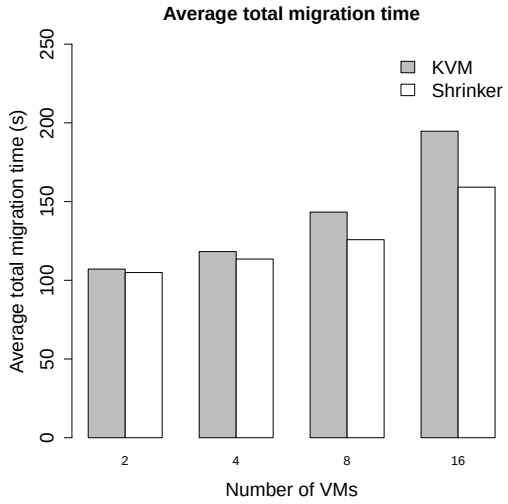
- Between 2 virtual machines and 4 virtual machines, we observe a three-fold improvement in the number of deduplicated pages. This is explained by the fact that, compared to the individual virtual machine scenario, we go from one additional virtual machine to three additional virtual machines. As a consequence, the possibilities of inter-VM deduplication are multiplied by 3. Similarly, 8 and 16 virtual machines respectively offer 7 times and 15 times more deduplication. For 16 virtual machines, we observe a reduction of transferred data of 33% and a decrease of migration time of 30.20%.

- The number of fully sent pages also increases with the size of the cluster, since the virtual machines content are not completely identical. This is explained by the operating system allocating memory for various purposes and modifying it differently in each virtual machine.

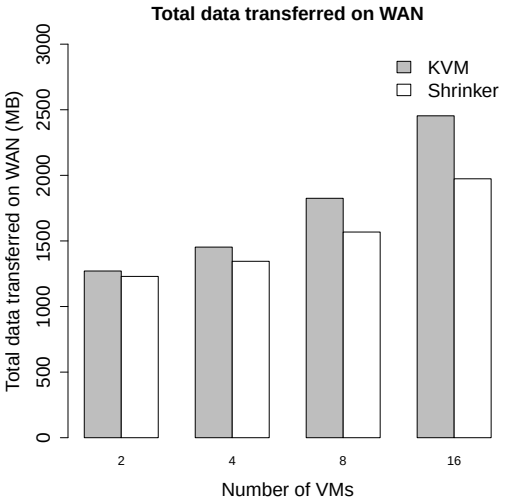
- The number of pulled pages augments with the size of the cluster. As concurrency in the live migration increases, more deduplicated pages arrive on the destination site before the original copy has been received and registered in the indexing service. As a consequence, they are requested to the source hypervisors. This result shows that our system should be improved to offer strategies to minimize the number of pulled pages. For instance, destination hypervisors could store such pages in a waiting list and periodically retry contacting the indexing service to locate the original copy.

6.5.2.2 Memory-intensive Workload

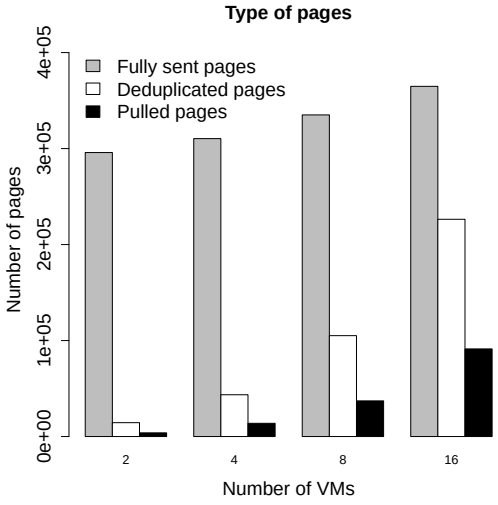
In this section, we study the results of Shrinker for a memory-intensive workload based on the NPB CG MPI program. This benchmarks also exhibits high CPU usage, and I/O activity related to the MPI communications. We migrate a cluster of virtual machines executing this benchmark with a class C problem (cg.C), for various cluster sizes, using a wide area bandwidth of 100 Mbps. Compared to the previous experiment, we do not use a cluster composed of one individual virtual machine, because the CG program cannot be built for only one process. As earlier, Figure 6.5(a) and Figure 6.5(b) respectively present the average total migration time and the total data transferred on the wide area network, and Figure 6.5(c) presents the distribution of the types of pages.



(a) Average total migration time



(b) Total WAN data transferred



(c) Types of pages transferred

Figure 6.5: Shrinker performance for the NBP cg.C memory-intensive workload

We observe that as the cluster size increases, Shrinker deduplicates more data and decreases the migration time. This is explained by the fact that the matrix data is of fixed size and is divided among all virtual machines participating in the computation. As such, the ratio between the memory allocated for the random matrix data (which has very low probability of deduplication) and the rest of the operating system (which can be deduplicated to some extent, as seen with the idle workload) decreases. We measured that each cg.C process used 528 MB of resident memory when executed with 2 processes, while for 16 processes, each one used only between 68 and 85 MB of resident memory. Consequently, for each additional virtual machine, the number of deduplicated pages greatly increases, while the amount of fully sent pages increases much more slowly.

6.5.2.3 I/O-intensive Workload

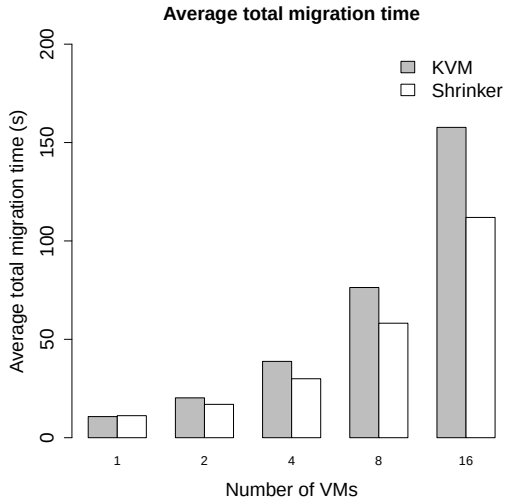
In this section, we study the performance of Shrinker for an I/O-intensive workload using the DBENCH benchmark. We migrate a cluster of virtual machines executing this benchmark for various cluster sizes, using a wide area bandwidth of 100 Mbps. Figure 6.6(a) and Figure 6.6(b) respectively present the average total migration time and the total data transferred on the wide area network. Figure 6.6(c) shows the distribution of the types of pages. These results are similar the idle virtual machine scenario presented in Section 6.5.2.1. The increased deduplication caused by each new virtual machine improves the total data transferred and average total migration time as the virtual cluster gets larger.

6.5.3 Impact of Available Network Bandwidth

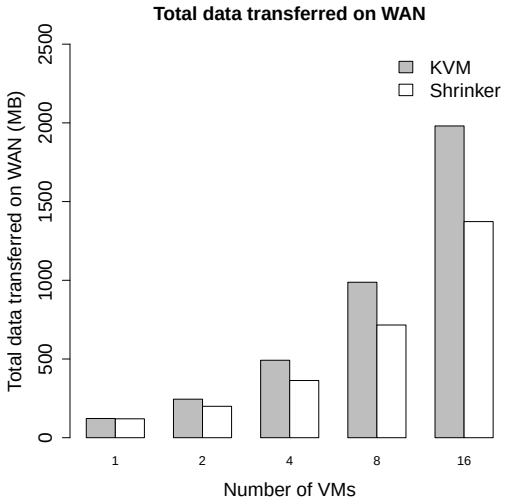
Figure 6.7 illustrates the impact of different available network bandwidths in the emulated wide area network. It shows the average total migration time of 16 idle virtual machines with various bandwidths from 100 to 1000 Mbps. We observe that Shrinker performs a similar reduction in migration time compared to KVM for all bandwidth rates.

6.5.4 Impact of Deduplication Granularity

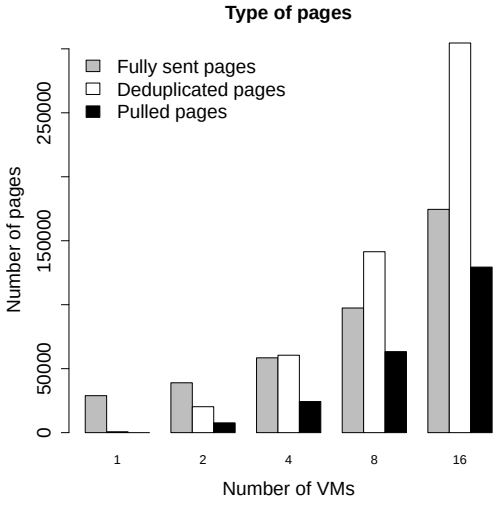
In this section, we study the impact of decreasing the chunk size used by deduplication (i.e. performing sub-page level deduplication). We extended our prototype to perform memory deduplication at a configurable granularity. We migrate 16 idle virtual machines for various chunk sizes, ranging from 512 to 4096 bytes. Figure 6.8 shows the amount of fully transferred memory per virtual machine compared with the amount of deduplicated memory per virtual machine. We observe that each decrease of the chunk size slightly improves deduplication. For a chunk size of 512 bytes, 58.6 MB per virtual machine are identified as duplicated, versus 52.7 MB for the native page size of 4 kB, an 11% increase in deduplication. However, the reduction in migration time is lower, approximately 7% between 4 KB and 512 bytes chunks. This can be explained by the additional metadata overhead (for each full chunk, an 8-byte memory address is transferred in addition to the content, and for each deduplicated chunk, an 8-byte memory address and a 20-byte SHA-1 hash are transferred), and also by the increased number of requests to locate and fetch chunks on the destination site using content-based addressing.



(a) Average total migration time



(b) Total WAN data transferred



(c) Types of pages transferred

Figure 6.6: Shrinker performance for the DBENCH I/O-intensive workload

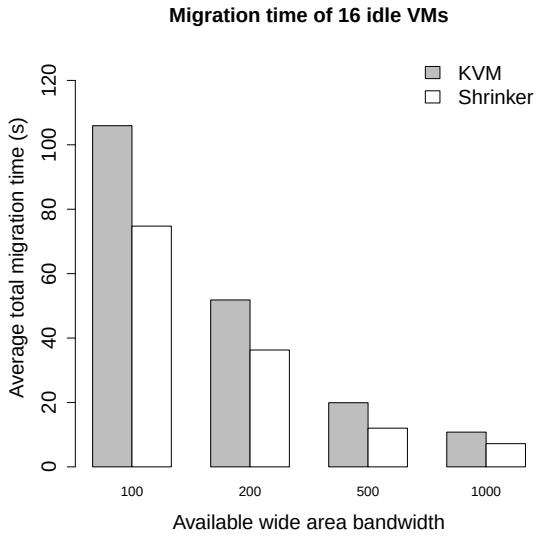


Figure 6.7: Average total migration time of 16 idle virtual machines

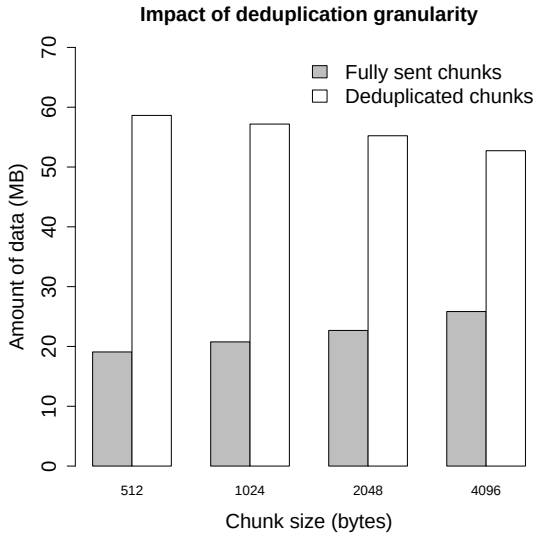


Figure 6.8: Comparison of the amount of memory fully transferred versus amount of memory deduplicated when migrating 16 idle virtual machines

6.6 Conclusion

In this chapter, we presented the design of Shrinker, a system to efficiently live migrate virtual clusters over wide area networks. Shrinker detects identical memory pages and disk blocks among multiple virtual machines and eliminates such duplicated data from the live migration. This allows to reduce total data transferred, and thus total migration time. Shrinker relies on distributed data deduplication on the source site to eliminate identical data. On the destination site, virtual machine data is retrieved with distributed content-based addressing. We use cryptographic hash functions to compute unique identifiers for each piece of data. We implemented a prototype of Shrinker in the KVM hypervisor. Our experiments show that it reduces both total data transferred and total migration time. This work was published at the Euro-Par 2011 conference [183].

Contrarily to other works leveraging data deduplication for improving live migration performance, Shrinker is a distributed mechanism, which allows to eliminate identical data contained in virtual machines hosted on different physical machines. In comparison, related contributions limit data deduplication to individual or collocated virtual machines.

In future work, we will finalize the support for storage migration in Shrinker, which should bring further improvement in live migration performance. We need to improve our system to offer better strategies to minimize the number of pages requested to source hypervisors when they cannot be located on the destination site. We also plan to study the integration of Shrinker with other sources of data in a cloud infrastructure, such as virtual machine image repositories. By making these repositories content addressable, virtual machine content could be obtained from storage services in addition to hypervisors nodes, further improving the performance of live migration. Moreover, this integration with virtual machine image repositories could bring benefits even for the migration of individual virtual machine instances, as identical data could be discovered between a virtual machine instance and existing virtual machine images stored in repositories.

Shrinker is one of the component of an efficient inter-cloud live migration system, allowing to improve the performance of live migration for large scale execution platforms. By reducing the total amount of data transferred by live migration, Shrinker reduces the cost of migration for environments in which data transfer is charged to users (like Amazon EC2). Furthermore, by decreasing the total migration time, it allows to reduce the time from migration decision to migration completion. By allowing rapid relocation when changes in resource availability, cost, or performance are detected, execution platforms can become more dynamic.

Part IV

Conclusion

Chapter 7

Conclusions and Perspectives

Contents

7.1 Contributions 120
7.2 Perspectives 122
7.2.1 Short Term Perspectives 122
7.2.2 Long Term Research Perspectives 124

This chapter concludes this dissertation, by summarizing our contributions and presenting future work directions. Our research was carried out in the emerging context of cloud computing and targeted issues related to the execution of computationally expensive parallel and distributed applications on such infrastructures. Our contributions followed two main topics. First, we studied how to create large-scale and elastic execution platforms from federated clouds, in an efficient and easy manner. Second, we proposed mechanisms to make cloud-based execution platforms more dynamic, based on inter-cloud live migration.

Short term work directions focus on addressing shortcomings of our contributions, while more long term directions are centered around using our contributions as founding blocks to build autonomous systems for efficiently and easily creating, managing, and migrating execution platforms in federated clouds. These systems would optimize usage of resources by using resource selection policies to best match application requirements. They would optimize the computations by detecting changes in cost, performance, and availability of resources among the many cloud infrastructures in their environment. They would make execution platforms resilient to failures of individual virtual machines or whole cloud infrastructures, by learning the fault tolerance requirements of applications, and leveraging resiliency across separate cloud infrastructures.

The needs for computational power have always been increasing since the inception of computer science. To satisfy these large requirements, scientists rely more and more on parallel and distributed computing. These computing paradigms work by decomposing problems in many small tasks and executing them in parallel on systems like *supercomputers* or *clusters* composed of a large number of processors. Furthermore, *grids* federate these large systems to create geographically distributed infrastructures belonging to different organizations. The shift towards parallel and distributed computing is made even stronger by the lack of progress in processor frequency in the recent years, replaced by an increase in the number of cores per processor.

In the last decade, a major change in the computing landscape has been the move towards virtualized infrastructures. By decoupling software from the underlying hardware platform, virtualization technologies bring more flexibility and isolation to computing infrastructures. The combination of virtualization technologies and large scale distributed infrastructures has put forward a new paradigm: cloud computing. In this model, computing resources are acquired in an on-demand, pay as you go fashion from commercial providers. The success of this type of service has also introduced cloud computing in non-public infrastructures, in the form of private and community clouds. Several abstraction levels are offered to users, allowing to provision raw virtualized resources (IaaS), develop applications for managed execution environments (PaaS), or directly use software via the Internet (SaaS). In particular, IaaS offers a flexible and powerful model with compatibility with existing applications and operating systems, and full control over the virtual resources.

This flexible availability of resources allows anyone to create execution platforms tailored exactly to their requirements. In this context, this PhD thesis focused on two objectives:

- making the creation of large-scale and elastic execution platforms on top of federated clouds easy and efficient,
- making these execution platforms more dynamic by allowing efficient inter-cloud live migration.

7.1 Contributions

To achieve these objectives, this PhD thesis made the following contributions:

Mechanisms to efficiently build large-scale and elastic execution platforms on top of federated clouds using sky computing. We studied elastic sky computing platforms and how they can be efficiently created and extended. We realized a series of experiments using resources from Nimbus clouds deployed on the Grid'5000 testbed in France and the FutureGrid testbed in the USA. In order to scale these cloud infrastructures, we developed new mechanisms to propagate virtual machine images, implemented them in Nimbus, and evaluated their performance. Propagation based on a broadcast chain allows to efficiently propagate the same virtual machine image to many nodes. Copy-on-write volumes allow to almost instantaneously provision new virtual machines from pre-propagated images. Additionally, we studied the effect of extending sky computing platforms with new cloud resources during execution in order to speed

up computations. This work was realized in collaboration with Kate Keahey (Argonne National Laboratory / Computation Institute, University of Chicago), Maurício Tsugawa, Andréa Matsunaga, and José Fortes (University of Florida). It was initiated during a 3-month visit in the ACIS laboratory at University of Florida, Gainesville, USA, in summer 2010. This creation of large scale and elastic platforms using sky computing has been demonstrated at the OGF-29 meeting, and was covered by the iSGTW online publication [73]. We presented these experiments in the poster session of the TeraGrid 2010 conference [184], and published in a special issue of ERCIM News on cloud computing [185]. We also contributed a book chapter in the *Advances in Parallel Computing* series [149]. Following these improvements to the Nimbus software, I was invited to join the Nimbus project team as an official committer. Additionally, we created a tool to perform large scale deployments of Nimbus on Grid'5000 and won the Large Scale Deployment Challenge at the 2010 Grid'5000 Spring School. This tool has been used extensively by several members of the Grid'5000 community to deploy Nimbus clouds for their own experiments.

Resilin, a system to easily create execution platforms over distributed cloud resources for running MapReduce computations. Users have now access to a large number of resources from multiple private and community clouds. In order to allow them to easily take advantage of these distributed resources, we propose Resilin, a system for creating execution platforms to run MapReduce computations. Resilin implements the Amazon Elastic MapReduce web service API while leveraging resources from private and community clouds. Resilin provisions, configures, and manages Hadoop execution platforms using resources from multiple EC2-compatible clouds. Resilin can manage these platforms in an elastic manner by changing the number of resources on demand, potentially using resources from different clouds. We compared Resilin with Amazon Elastic MapReduce and showed that it offers similar performance results. In addition to offering a service for easily performing MapReduce computations without having to deal with low level cloud interfaces, this contribution will serve as a base framework for future experiments with multi-cloud MapReduce computations. This work was realized in collaboration with Ancuţa Iordache, Master student at West University of Timişoara, Romania, during her Master internship in the Myriads team. An initial version of this work has been presented in the poster session of the CCA 2011 workshop [182]. We plan to release Resilin as open source software in the near future, allowing the community to benefit from our work.

Mechanisms to allow virtualized resources to perform efficient communications in the presence of inter-cloud live migration. Live migration was designed to operate in local area networks. As such, inter-cloud live migration requires reconfiguration support at the network layer. In the context of the ViNe virtual network, we proposed mechanisms to automatically detect live migrations and reconfigure the network layer, in order to keep communications uninterrupted. Additionally, we proposed a routing optimization to offer optimal communication performance when virtual machines are located on the same local area network. We integrated a prototype implementation of our mechanisms in the ViNe virtual network, and validated our approach experimentally. These contributions allow execution platforms to be freely relocated between different cloud infrastructures, while providing high performance for intra-cloud communication. This

work was realized in collaboration with José Fortes, Andréa Matsunaga, and Maurício Tsugawa, during a 3-month visit in the ACIS laboratory at University of Florida, Gainesville, USA, in summer 2010. This work was published at the IEEE MENS 2010 workshop [209].

Shrinker, a live migration protocol optimized to efficiently migrate virtual clusters between data centers connected by wide area networks. Live migration makes execution platforms more dynamic by relocating them between different infrastructures. Inter-cloud live migration is possible using state of the art contributions, including the mechanisms we proposed in the context of the ViNe virtual network. However, inter-cloud live migration generates large amounts of traffic on wide area networks, because of the large size of virtual machines. This problem is intensified when migrating clusters of virtual machines between different clouds. Shrinker is a live migration protocol leveraging identical data among a cluster of virtual machines in order to reduce the network traffic created by inter-cloud live migrations. We use distributed data deduplication and distributed content-addressing as the two core mechanisms of Shrinker, in order to detect identical pages and transfer only unique copies on wide area links. We implemented a prototype in the KVM hypervisor. Evaluations on the Grid'5000 testbed show that Shrinker reduces both traffic generated by live migration and total migration time. This contribution enables efficient migration of large scale execution platforms between clouds, otherwise restricted by the limited bandwidth offered by wide area networks. This work was published at the Euro-Par 2011 conference [183].

These contributions allow to easily and efficiently build large-scale elastic execution platforms by federating multiple cloud infrastructures, and allow these execution platforms to be live migrated between different clouds, making them more dynamic than traditional infrastructures. These mechanisms offer founding blocks for making next generation cloud systems more efficient, elastic, and dynamic. We presented these contributions in the IPDPS 2011 PhD forum and won one of the three Best Poster Awards [181].

7.2 Perspectives

The contributions presented in this dissertation present multiple research directions for future work. First, we describe short term perspectives to address the shortcomings of our contributions. Then, we present long term research directions leveraging our contributions to build the next generation of cloud-based execution platforms.

7.2.1 Short Term Perspectives

7.2.1.1 Improvements of Large Scale and Elastic Sky Computing Platforms

Our large scale experiments with sky computing platforms showed that increasing the size of the platforms yields substantial performance increase. However, our results showed lower improvements as the platforms get more distributed. It would be interesting to analyze the exact factors limiting the performance of these platforms. The results would be used to research new methods for optimizing such platforms at a large scale.

The dynamic extension of these platforms is an appealing topic of research. Sky computing platforms could be managed autonomously by a framework capable of elastically resizing them, depending on user and application requirements. Additionally, managing the elasticity of environments based on other frameworks than Hadoop would be required.

Finally, we need to compare the performance of the propagation mechanisms we proposed with available alternative contributions such as LANTorrent and scp-wave.

7.2.1.2 Extensions to Resilin

As future work, we plan to complete compatibility with the Amazon Elastic MapReduce API, by providing Hive and Pig job flows. We will also evaluate Resilin with other cloud implementations than Nimbus, such as Eucalyptus and its Walrus cloud storage repository. We also plan to benchmark Resilin with MapReduce applications using big data sets rather than sample applications. We will optimize the deployment process and analyze reasons for slowdowns compared to Elastic MapReduce. For users having access to multiple clouds, we will investigate how Resilin can automatically select which cloud infrastructures to use and how many resources to provision, instead of relying on users to choose where and how they want to provision resources. Finally, we plan to release the Resilin software as open source in the near future, making it available for the whole community.

7.2.1.3 Performance Evaluation of Inter-cloud Live Migration Network Support

The mechanisms we propose in this thesis for supporting inter-cloud live migration at the network level have been implemented in ViNe and validated experimentally. However, we did not perform extensive performance evaluation. Future work will involve evaluating the performance of our approach, characterizing the time of live migration detection, virtual network reconfiguration, and the resulting virtual machine downtime. We will also measure inter-cloud and intra-cloud communication performance in the presence of inter-cloud live migration.

7.2.1.4 Extended Evaluation and Improvements to Shrinker

We plan to finalize our Shrinker prototype to perform data deduplication on storage migration, and evaluate its performance. Improvements to our system are required in order to minimize the number of pages requested to source hypervisors when they cannot be located on the destination site, which would further increase the performance of Shrinker. We also plan to study how Shrinker can allow destination hypervisors to fetch data from local virtual machine image repositories found in most cloud computing infrastructures, to further decrease the amount of data sent over wide area networks. This could be performed by proposing content addressable storage for cloud repositories, and proving this concept by implementing content addressable storage in the Cumulus cloud storage repository.

7.2.2 Long Term Research Perspectives

The long term research perspectives focus on leveraging the contributions of this thesis as founding blocks for building the next generation of cloud-based execution platforms. We think that it is possible to build autonomous systems that can create, manage, resize, and migrate execution platforms on top of federated clouds in order to make the best use of resources, minimize computation cost, or reduce time to completion, depending on user-selected policies. These systems would interact with multiple clouds through well-defined interfaces in order to learn information about resource availability, performance, and cost. In this section, we outline the work and research required in designing and implementing such a system.

First, mechanisms are required to allow applications to deliver their resource requirements, and cloud platforms to provide cost and resource availability information. Being able to acquire such information is critical in order to make decisions that would modify the size and placement of execution platforms. These are the basic mechanisms necessary to create an autonomous management layer with policies capable of selecting cloud resources according to cost or performance constraints specified by users. To elastically resize or dynamically relocate execution platforms when detecting changes in the environment, the autonomous system would use self-adaptive techniques such as feedback loop mechanisms.

Second, universal inter-cloud live migration would require support at the cloud API level. Inter-cloud live migration services should be implemented in a secure way, and be minimally invasive to the cloud infrastructures. For instance, live migration mechanisms should not create direct connections between internal resources of each cloud, but should instead use proxying techniques. Additionally, live migration compatibility between different hypervisors technologies would increase the potential number of usable resources.

Additionally, the autonomous management system would require self-healing capabilities to deal with failures. Applications could provide requirements on the fault tolerance capabilities of execution platforms. For instance, they could specify which resources should be made fault-tolerant by the system, and which would already be fault-tolerant at the application level. The system could leverage multiple cloud infrastructures to create resilient execution platforms. Furthermore, when outages in cloud infrastructure would be detected, the management system could migrate execution platforms to other clouds.

Finally, live migration decisions should make intelligent virtual machine placement decisions that favor performance. Ideally, groups of virtual machines with low latency or high bandwidth constraints should be placed in the same infrastructures, while virtual machines with low communication requirements can be hosted on different infrastructures with low overhead. We started studying this problem with Djawida Dib, PhD student in the Myriads research team, and built a framework to transparently analyze communication patterns of distributed applications by capturing network traffic [99].

Bibliography

[1] Apache Hadoop. http://hadoop.apache.org/. 14, 38, 72, 145

[2] Apache Libcloud. http://libcloud.apache.org/. 47, 49

[3] AWS CloudFormation. http://aws.amazon.com/cloudformation/. 48

[4] BonFIRE. http://www.bonfire-project.eu/. 49

[5] boto: A Python interface to Amazon Web Services. http://boto.cloudhackers.com/. 49, 74, 76

[6] Cloud Data Management Interface (CDMI). http://www.snia.org/cdmi. 45

[7] ConPaaS. http://www.conpaas.eu/. 69

[8] Contrail Project. http://contrail-project.eu/. 49

[9] Deltacloud. http://incubator.apache.org/deltacloud/. 47

[10] Distributed Replicated Block Device. http://www.drbd.org/. 34

[11] Distributed Ruby. http://www.ruby-doc.org/stdlib/libdoc/drb/rdoc/. 38

[12] ElasticHosts. http://www.elastichosts.com/. 69

[13] Eucalyptus. http://www.eucalyptus.com/. 42, 146

[14] FreeBSD Jails. http://www.freebsd.org/doc/handbook/jails.html. 29

[15] FutureGrid. https://portal.futuregrid.org/. 13, 61, 143, 144

[16] gLite. http://glite.cern.ch/. 40

[17] Global Environment for Network Innovations (GENI). http://www.geni.net/. 60

[18] Globus Toolkit. http://www.globus.org/toolkit/. 40

[19] Great Internet Mersenne Prime Search. http://www.mersenne.org/freesoft/. 77

[20] Heroku. http://www.heroku.com/. 41

[21] Interacting with Walrus (2.0). http://open.eucalyptus.com/wiki/EucalyptusWalrusInteracting_v2.0. 45

[22] Java Remote Method Invocation (RMI). http://download.oracle.com/javase/tutorial/rmi/index.html. 38

[23] LANTorrent. http://www.nimbusproject.org/docs/current/admin/reference.html#lantorrent. 45, 67

[24] libvirt: The virtualization API. http://libvirt.org/. 46

[25] mOSAIC Cloud. http://www.mosaic-cloud.eu/. 49

[26] MPICH2. http://www.mcs.anl.gov/research/projects/mpich2/. 38

[27] Nimbus. http://www.nimbusproject.org/. 13, 42, 77, 143, 144, 146

[28] Open Cloud Computing Interface. http://occi-wg.org/. 44

[29] Open Grid Forum. http://www.ogf.org/. 40

[30] Open MPI. http://www.open-mpi.org/. 38

[31] openMosix. http://openmosix.sourceforge.net/. 39

[32] OpenMP. http://www.openmp.org/. 37

[33] OpenNebula. http://www.opennebula.org/. 42, 146

[34] OpenSSI. http://www.openssi.org/. 39

[35] OpenStack. http://www.openstack.org/. 42

[36] OpenVZ. http://wiki.openvz.org/Main_Page. 29

[37] paramiko. http://www.lag.net/paramiko/. 76

[38] POSIX Threads. POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995). 37

[39] Rackspace Cloud. http://www.rackspace.com/cloud/. 41

[40] SAGA. http://saga.cct.lsu.edu/. 40

[41] Salesforce.com. http://www.salesforce.com/. 42

[42] Swift. http://swift.openstack.org/. 45

[43] TOP500. http://www.top500.org/. 36

[44] TORQUE Resource Manager. http://www.clusterresources.com/pages/products/torque-resource-manager.php. 39, 61

[45] Twisted. http://twistedmatrix.com/trac/. 76

[46] Worldwide LHC Computing Grid. http://lcg.web.cern.ch/lcg/. 40

[47] XML-RPC Home Page. http://www.xmlrpc.com/. 38

[48] XtreemOS. http://www.xtreemos.eu/. 40

[49] The Future of Cloud Computing. Expert group report, European Commission, 2010. 40

[50] David Abramson, Jon Giddy, and Lew Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *14th International Parallel and Distributed Processing Symposium*, pages 520–528, 2000. 37

[51] Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes, Ivan Krsul, Andréa Matsunaga, Maurício Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, and Xiaomin Zhu. From virtualized resources to virtual computing grids: the In-VIGO system. *Future Generation Computer Systems*, 21(6):896–909, June 2005. 42

[52] Keith Adams and Ole Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. In *Proceedings of the 12th international conference on architectural support for programming languages and operating systems*, pages 2–13, 2006. 27

[53] Narasimha R. Adiga, George Almási, Yariv Aridor, Rajkishore Barik, Daniel K. Beece, Ralph Bellofatto, Gyan Bhanot, Randy Bickford, Matthias A. Blumrich, Arthur A. Bright, José R. Brunheroto, Calin Cascaval, José G. Castaños, W. Chan, Luis Ceze, Paul Coteus, Siddhartha Chatterjee, Dong Chen, George L.-T. Chiu, Thomas M. Cipolla, Paul Crumley, K. M. Desai, Alina Deutsch, Tamar Domany, Marc Boris Dombrowa, Wilm E. Donath, Maria Eleftheriou, C. Christopher Erway, J. Esch, Blake G. Fitch, Joseph Gagliano, Alan Gara, Rahul Garg, Robert S. Germain, Mark Giampapa, Balaji Gopalsamy, John A. Gunnels, Manish Gupta, Fred G. Gustavson, Shawn Hall, Ruud A. Haring, David F. Heidel, Philip Heidelberger, Lorraine Herger, Dirk Hoenicke, R. D. Jackson, T. Jamal-Eddine, Gerard V. Kopcsay, Elie Krevat, Manish P. Kurhekar, Alphonso P. Lanzetta, Derek Lieber, L. K. Liu, M. Lu, Mark P. Mendell, A. Misra, Y. Moatti, Lawrence S. Mok, José E. Moreira, Ben J. Nathanson, Matthew Newton, Martin Ohmacht, Adam J. Oliner, Vinayaka Pandit, R. B. Pudota, Rick A. Rand, Richard D. Regan, Bradley Rubin, Albert E. Ruehli, Silvius Rus, Ramendra K. Sahoo, Alda Sanomiya, Eugen Schenfeld, M. Sharma, Edi Shmueli, Sarabjeet Singh, Peilin Song, Vijay Srinivasan, Burkhard D. Steinmacher-Burow, Karin Strauss, Christopher W. Surovic, Richard A. Swetz, Todd Takken, R. Brett Tremaine, Mickey Tsao, Arun R. Umamaheshwaran, P. Verma, Pavlos Vranas, T. J. Christopher Ward, Michael E. Wazlowski, W. Barrett, C. Engel, B. Drehmel, B. Hilgart, D. Hill, F. Kasemkhani, David J. Krolak, C. T. Li, Thomas A. Liebsch, James A. Marcella, A. Muff, A. Okomo, M. Rouse, A. Schram, M. Tubbs, G. Ulsh, Charles D. Wait, J. Wittrup, M. Bae, Kenneth A. Dockser, Lynn Kissel, Mark K. Seager, Jeffrey S. Vetter, and K. Yates. An Overview of the BlueGene/L Supercomputer. In *Supercomputing '02*, pages 1–22, 2002. 35

[54] Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W. Moore, and Andy Hopper. Predicting the Performance of Virtual Machine Migration. In *2010 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 37–46, 2010. 32

[55] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990. 59, 144

[56] Amazon Web Services. Amazon EC2 Spot Instances. http://aws.amazon.com/ec2/spot-instances/. 83, 90

[57] Amazon Web Services. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/. 41, 146

[58] Amazon Web Services. Amazon Elastic MapReduce. http://aws.amazon.com/elasticmapreduce/. 72, 146

[59] Amazon Web Services. Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/. 43, 146

[60] Amazon Web Services. Auto Scaling. http://aws.amazon.com/autoscaling/. 69

[61] Amazon Web Services. CloudBurst. http://aws.amazon.com/articles/2272. 80, 146

[62] Amazon Web Services. VM Import. http://aws.amazon.com/ec2/vmimport/. 46

[63] Amazon Web Services. Word Count Example. http://aws.amazon.com/articles/2273. 80, 146

[64] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS '67*, 1967. 35

[65] David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 1–7, 2004. 39

[66] Andrea Arcangeli, Izik Eidus, and Chris Wright. Increasing memory density by using KSM. In *2009 Linux Symposium*, 2009. 102, 148

[67] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, University of California at Berkeley, February 2009. 40

[68] Ars Technica. Amazon's lengthy cloud outage shows the danger of complexity. http://arstechnica.com/business/news/2011/04/amazons-lengthy-cloud-outage-shows-the-danger-of-complexity.ars. 47

[69] Amnon Barak and Oren La'adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Future Generation Computer Systems*, 13(4-5):361–372, March 1998. 31

[70] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *19th ACM Symposium on Operating Systems Principles*, pages 164–177. ACM, 2003. 24, 27

[71] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *FREENIX Track: 2005 USENIX Annual Technical Conference*, pages 41–46, 2005. 106

[72] Andrew D. Birrell and Bruce Jay Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–54, February 1984. 38

[73] Miriam Boon. Reaching for sky computing. `http://www.isgtw.org/feature/feature-reaching-sky-computing`. 70, 121

[74] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments*, 2007. 34, 35

[75] John Bresnahan, Tim Freeman, David LaBissoniere, and Kate Keahey. Managing Appliance Launches in Infrastructure Clouds. In *TeraGrid '11*, 2011. 49

[76] John Bresnahan, Kate Keahey, David LaBissoniere, and Tim Freeman. Cumulus: An Open Source Storage Cloud for Science. In *2nd Workshop on Scientific Cloud Computing*, pages 25–32, 2011. 45, 82, 146

[77] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *10th International Conference on Algorithms and Architectures for Parallel Processing*, pages 13–31, 2010. 49

[78] Roy Campbell, Indranil Gupta, Michael Heath, Steven Y. Ko, Michael Kozuch, Marcel Kunze, Thomas Kwan, Kevin Lai, Hing Yan Lee, Martha Lyons, Dejan Milojicic, David O'Hallaron, and Yeng Chai Soh. Open Cirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. *HotCloud'09: Proceedings of the 2009 conference on Hot topics in cloud computing*, 2009. 49

[79] Franck Cappello, Eddy Caron, Michel Dayde, Frédéric Desprez, Yvon Jégou, Pascale Primet, Emmanuel Jeannot, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quétier, and Olivier Richard. Grid'5000: a large scale and highly reconfigurable Grid experimental testbed. In *6th IEEE/ACM International Workshop on Grid Computing*, pages 99–106, 2005. 13, 60, 77, 143, 144

[80] Franck Cappello and Daniel Etiemble. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (Supercomputing '00)*, pages 1–12, 2000. 38

[81] Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring MapReduce Efficiency with Highly-Distributed Data. In *MapReduce '11*, 2011. 76

[82] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, 2008. 60

[83] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. How to Enhance Cloud Architectures to Enable Cross-Federation. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 337–345, 2010. 49

[84] Damien Cerbelaud, Shishir Garg, and Jeremy Huylebroeck. Opening The Clouds: Qualitative Overview of the State-of-the-art Open Source VM-based Cloud Management Platforms. In *ACM/IFIP/USENIX 10th International Middleware Conference*, 2009. 42

[85] Navraj Chohan, Claris Castillo, Mike Spreitzer, Malgorzata Steinder, Asser Tantawi, and Chandra Krintz. See Spot Run: Using Spot Instances for MapReduce Workflows. In *2nd USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–7, 2010. 84

[86] Walfredo Cirne, Daniel Paranhos, Lauro Costa, Elizeu Santos-Neto, Francisco Brasileiro, and Jacques Sauvé. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. In *2003 International Conference on Parallel Processing*, pages 407–416, 2003. 37

[87] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *2nd Symposium on Networked Systems Design & Implementation*, pages 273–286, 2005. 32

[88] Benoit Claudel, Guillaume Huard, and Olivier Richard. TakTuk, Adaptive Deployment of Remote Executions. In *HPDC 2009*, pages 91–100, 2009. 64, 145

[89] Bram Cohen. BitTorrent. `http://www.bittorrent.com/`. 67

[90] Contrail Project. D8.1: Requirements and Architecture for the implementation of the High Level Services. Deliverable, 2011. 69

[91] Contrail Project. D9.1: Specification of requirements and architecture for runtime environments. Deliverable, 2011. 69

[92] Massimo Coppola, Yvon Jégou, Brian Matthews, Christine Morin, Luis Pablo Prieto, Óscar David Sánchez, Erica Y. Yang, and Haiyan Yu. Virtual Organization Support within a Grid-Wide Operating System. *IEEE Internet Computing*, 12(2):20–28, 2008. 40

[93] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norman Hutchinson, and Andrew Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *5th USENIX Symposium on Networked Systems Design & Implementation*, 2008. 24

[94] Cycle Computing. Lessons learned building a 4096-core Cloud HPC Supercomputer for $418/hr. `http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html`. 76

[95] Marcos Dias de Assunção, Alexandre di Costanzo, and Rajkumar Buyya. Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters. In *18th International ACM Symposium on High Performance Distributed Computing*, pages 141–150. ACM Press, 2009. 69

[96] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating Systems Design and Implementation*, pages 137–149, 2004. 14, 38, 145

[97] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, January 2008. 14, 38, 145

[98] Umesh Deshpande, Xiaoshuang Wang, and Kartik Gopalan. Live Gang Migration of Virtual Machines. In *20th International ACM Symposium on High Performance Parallel and Distributed Computing*, pages 135–146, 2011. 103

[99] Djawida Dib. Migration dynamique d'applications réparties virtualisées dans les fédérations d'infrastructures distribuées. Master's thesis, Université de Rennes 1, June 2010. 124

[100] Christian Engelmann, Geoffroy Vallee, Thomas Naughton, and Stephen Scott. Proactive Fault Tolerance Using Preemptive Migration. In *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009. 31

[101] Dror G. Feitelson and Larry Rudolph. Toward Convergence in Job Schedulers for Parallel Supercomputers. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–26, 1996. 36

[102] Roy T. Fielding and Richard N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002. 43

[103] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A Case For Grid Computing On Virtual Machines. In *23rd International Conference on Distributed Computing Systems*, pages 550–559, 2003. 28

[104] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 1999. 39

[105] Arijit Ganguly, Abhishek Agrawal, P. Oscar Boykin, and Renato Figueiredo. IP over P2P: Enabling Self-configuring Virtual IP Networks for Grid Computing. In *IPDPS '06*, pages 1–10, 2006. 93

[106] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *19th ACM Symposium on Operating Systems Principles*, 2003. 38

[107] Robert P. Goldberg. Survey of Virtual Machine Research. *Computer*, 7(6):34–45, June 1974. 26

[108] Google. Gmail. `http://www.gmail.com`. 42

[109] Google. Google App Engine. `http://code.google.com/appengine/`. 41

[110] Andrew S. Grimshaw and Anand Natrajan. Legion: Lessons Learned Building a Grid Operating System. *Proceedings of the IEEE*, 93(3):589–603, March 2005. 40

[111] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. MapReduce in the Clouds for Science. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 565–572, 2010. 83

[112] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat. Difference Engine: Harnessing Memory Redundancy in Virtual Machines. In *8th USENIX Symposium on Operating Systems Design and Implementation*, pages 309–322, 2008. 102, 148

[113] Stuart Hacking and Benoît Hudzia. Improving the Live Migration Process of Large Enterprise Applications. In *3rd International Workshop on Virtualization Technologies in Distributed Computing*, 2009. 102

[114] Fang Hao, T. V. Lakshman, Sarit Mukherjee, and Haoyu Song. Enhancing Dynamic Cloud-based Services using Network Virtualization. *SIGCOMM Computer Communication Review*, 40(1), January 2010. 94

[115] Eric Harney, Sebastien Goasguen, Jim Martin, Mike Murphy, and Mike Westall. The efficacy of live virtual machine migrations over the internet. In *VTDC '07: Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*, 2007. 35

[116] Qiming He, Shujia Zhou, Ben Kobler, Dan Duffy, and Tom McGlynn. Case study for running HPC applications in public clouds. In *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 395–401, 2010. 57

[117] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2009. 31

[118] Michael Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, March 2009. 33

[119] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Enabling Instantaneous Relocation of Virtual Machines with a Lightweight VMM Extension. In *10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 73–83, 2010. 33

[120] Takahiro Hirofuchi, Hidemoto Nakada, Hirotaka Ogawa, Satoshi Itoh, and Satoshi Sekiguchi. A live storage migration mechanism over wan and its performance evaluation. In *3rd International Workshop on Virtualization Technologies in Distributed Computing*, 2009. 34

[121] Takahiro Hirofuchi, Hirotaka Ogawa, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services on Clouds. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 460–465, 2009. 34

[122] Felix Hupfeld, Toni Cortes, Björn Kolbeck, Jan Stender, Erich Focht, Matthias Hess, Jesus Malo, Jonathan Marti, and Eugenio Cesario. The XtreemFS architecture—a case for object-based file systems in Grids. *Concurrency and Computation: Practice and Experience*, 20(17):2049–2060, December 2008. 34

[123] Intel. Threading Building Blocks. `http://threadingbuildingblocks.org/`. 37

[124] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 159–168, 2010. 57, 77

[125] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. Live Virtual Machine Migration with Adaptive Memory Compression. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing*, 2009. 102

[126] Dilip Joseph, Jayanth Kannan, Ayumu Kubota, Karthik Lakshminarayanan, Ion Stoica, and Klaus Wehrle. OCALA: An Architecture for Supporting Legacy Applications over Overlays. In *NSDI '06*, pages 267–280, 2006. 93

[127] Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards optimizing hadoop provisioning in the cloud. In *USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–5, 2009. 84

[128] Katarzyna Keahey, Ian Foster, Tim Freeman, and Xuehai Zhang. Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. *Scientific Programming*, 13(4):265–275, 2005. 13, 42, 77, 143, 144, 146

[129] Katarzyna Keahey and Tim Freeman. Contextualization: Providing One-Click Virtual Clusters. In *4th IEEE International Conference on e-Science*, 2008. 48, 58

[130] Katarzyna Keahey, Maurício Tsugawa, Andréa Matsunaga, and José A. B. Fortes. Sky Computing. *IEEE Internet Computing*, 13(5):43–51, 2009. 13, 48, 57, 76, 142, 144

[131] Kate Keahey and Tim Freeman. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In *First Workshop on Cloud Computing and its Applications*, pages 1–5, 2008. 41, 42, 72

[132] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux Virtual Machine Monitor. In *2007 Linux Symposium*, pages 225–230, 2007. 14, 25, 102, 106, 148, 149

[133] Michael Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 40–46, 2002. 32

[134] H. Andrés Lagar-Cavilla, Joseph A. Whitney, Roy Bryant, Philip Patchin, Michael Brudno, Eyal de Lara, Stephen M. Rumble, M. Satyanarayanan, and Adin Scannell. SnowFlock: Virtual Machine Cloning as a First-Class Cloud Primitive. *ACM Transactions on Computer Systems*, 29(1), February 2011. 33

[135] Horacio Lagar-Cavilla, Joseph Whitney, Adin Scannell, Philip Patchin, Stephen Rumble, Eyal Lara, Michael Brudno, and Mahadev Satyanarayanan. SnowFlock: rapid virtual machine cloning for cloud computing. In *Proceedings of the fourth ACM european conference on Computer systems*, April 2009. 33

[136] Kai Li and Paul Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, 7(4), November 1989. 37

[137] Anthony Liguori. The Myth of Type I and Type II Hypervisors. `http://blog.codemonkey.ws/2007/10/myth-of-type-i-and-type-ii-hypervisors.html`. 25

[138] Anthony N. Liguori and Eric Van Hensbergen. Experiences with Content Addressable Storage and Virtual Disks. In *First Workshop on I/O Virtualization*, 2008. 102, 148

[139] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live Migration of Virtual Machine Based on Full System Trace and Replay. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, 2009. 33

[140] Huan Liu. Cutting MapReduce Cost with Spot Market. In *3rd USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–5, 2011. 84

[141] Huan Liu and Dan Orban. Cloud MapReduce: a MapReduce Implementation on top of a Cloud Operating System. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 464–474, 2011. 83

[142] Pengcheng Liu, Ziye Yang, Xiang Song, Yixun Zhou, Haibo Chen, and Binyu Zang. Heterogeneous Live Migration of Virtual Machines. In *2008 International Workshop on Virtualization Technology*, pages 1–9, 2008. 46

[143] Renaud Lottiaux, Pascal Gallard, Geoffroy Vallée, Christine Morin, and Benoit Boissinot. OpenMosix, OpenSSI and Kerrighed: A Comparative Study. In *2005 IEEE International Symposium on Cluster Computing and the Grid*, pages 1016–1023, 2005. 39

[144] Yingwei Luo, Binbin Zhang, Xiaolin Wang, Zhenlin Wang, Yifeng Sun, and Haogang Chen. Live and incremental whole-system migration of virtual machines using block-bitmap. In *2008 IEEE International Conference on Cluster Computing*, pages 99–106, 2008. 34

[145] Yuan Luo, Zhenhua Guo, Yiming Sun, Beth Plale, Judy Qiu, and Wilfred W. Li. A Hierarchical Framework for Cross-Domain MapReduce Execution. In *Proceedings of the second international workshop on Emerging computational methods for the life sciences*, pages 15–22, 2011. 76

[146] Vittorio Manetti, Roberto Canonico, Giorgio Ventre, and Ioannis Stavrakakis. System-level virtualization and Mobile IP to support service mobility. In *2009 International Conference on Parallel Processing Workshops*, pages 243–248, 2009. 94

[147] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. In *10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 43–52, 2010. 69

[148] Ali Mashtizadeh, Emré Celebi, Tal Garfinkel, and Min Cai. The Design and Evolution of Live Storage Migration in VMware ESX. In *USENIX ATC '11*, 2011. 34

[149] Andréa Matsunaga, Pierre Riteau, Maurício Tsugawa, and José Fortes. Crosscloud Computing. In *High Performance Computing: From Grids and Clouds to Exascale*, volume 20 of *Advances in Parallel Computing*. IOS Press, 2011. 70, 121

[150] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In *4th IEEE International Conference on e-Science*, pages 222–229, 2008. 59, 76

[151] Microsoft. Microsoft Hyper-V Server. `http://www.microsoft.com/hyper-v-server/`. 45

[152] Microsoft. Windows Azure Platform. `http://www.microsoft.com/windowsazure/`. 83

[153] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process Migration. *ACM Computing Surveys*, 32(3):241–299, September 2000. 31, 32

[154] Grzegorz Milos, Derek G. Murray, Steven Hand, and Michael A. Fetterman. Satori: Enlightened page sharing. In *2009 USENIX Annual Technical Conference*, 2009. 102, 148

[155] Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente. Multicloud Deployment of Computing Clusters for Loosely Coupled MTC Applications. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):924–930, June 2011. 69

[156] Christine Morin, Pascal Gallard, Renaud Lottiaux, and Geoffroy Vallée. Towards an efficient single system image cluster operating system. *Future Generation Computer Systems*, 20(4):505–521, May 2004. 39

[157] Arun Nagarajan, Frank Mueller, Christian Engelmann, and Stephen Scott. Proactive fault tolerance for HPC with Xen virtualization. In *Proceedings of the 21st annual international conference on Supercomputing*, 2007. 31

[158] Kenneth Nagin, David Hadas, Zvi Dubitzky, Alex Glikson, Irit Loy, Benny Rochwerger, and Liran Schour. Inter-Cloud Mobility of Virtual Machines. In *The 4th Annual International Systems and Storage Conference*, pages 1–12, 2011. 34

[159] NASA. `http://nebula.nasa.gov/`. 42

[160] Partho Nath, Michael A. Kozuch, David R. O'Hallaron, Jan Harkes, Mahadev Satyanarayanan, Niraj Tolia, and Matt Toups. Design Tradeoffs in Applying Content Addressable Storage to Enterprise-scale Systems Based on Virtual Machines. In *2006 USENIX Annual Technical Conference*, 2006. 102, 148

[161] National Institute of Standards and Technology. Secure Hash Standard, April 1995. 103

[162] NCBI. BLAST: Basic Local Alignment Search Tool. http://blast.ncbi.nlm.nih.gov/Blast.cgi. 59, 144

[163] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast Transparent Migration for Virtual Machines. In *2005 USENIX Annual Technical Conference*, pages 391–394, 2005. 32

[164] Bogdan Nicolae, Gabriel Antoniu, Luc Bougé, Diana Moise, and Alexandra Carpen-Amarie. BlobSeer: Next-generation data management for large scale infrastructures. *Journal of Parallel and Distributed Computing*, 71(2):169–184, February 2011. 67

[165] Bogdan Nicolae, John Bresnahan, Kate Keahey, and Gabriel Antoniu. Going Back and Forth: Efficient Multideployment and Multisnapshotting on Clouds. In *20th International ACM Symposium on High Performance Parallel and Distributed Computing*, 2011. 67

[166] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009. 42, 146

[167] Lucas Nussbaum, Fabienne Anhalt, Olivier Mornard, and Jean-Patrick Gelas. Linux-based virtualization for HPC clusters. In *2009 Linux Symposium*, 2009. 28

[168] D. Ogrizovic, B. Svilicic, and E. Tijan. Open Source Science Clouds. In *33rd International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 1189–1192, 2010. 42

[169] Ana-Maria Oprescu, Thilo Kielmann, and Haralambie Leahu. Budget estimation and control for bag-of-tasks scheduling in clouds. *Parallel Processing Letters*, 21(2):219–243, 2011. 69

[170] Opscode. Chef. http://www.opscode.com/chef/. 62

[171] Oracle. VirtualBox. http://www.virtualbox.org/. 25

[172] Junjie Peng, Xuejun Zhang, Zhou Lei, Bofeng Zhang, Wu Zhang, and Qing Li. Comparison of Several Cloud Computing Platforms. In *Second International Symposium on Information Science and Engineering*, pages 23–27, 2009. 42

[173] C. Perkins. IP Mobility Support for IPv4, Revised. RFC 5944, November 2010. 35, 93, 147

[174] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. RFC 6275, July 2011. 35, 93, 147

[175] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *ACM SIGCOMM Computer Communications Review*, 33(1):59–64, January 2003. 60

[176] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, July 1974. 26

[177] Daniel Price and Andrew Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In *LISA '04*, pages 241–254, 2004. 29

[178] Sean Quinlan and Sean Dorward. Venti: A New Approach to Archival Storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, 2002. 106

[179] Lavanya Ramakrishnan, Piotr T. Zbiegel, Scott Campbell, Rick Bradshaw, Richard Shane Canon, Susan Coghlan, Iwona Sakrejda, Narayan Desai, Tina Declerck, and Anping Liu. Magellan: Experiences from a Science Cloud. In *2nd Workshop on Scientific Cloud Computing*, pages 49–58, 2011. 42, 72

[180] Sean Rhea, Russ Cox, and Alex Pesterev. Fast, Inexpensive Content-Addressed Storage in Foundation. In *2008 USENIX Annual Technical Conference*, 2008. 102, 148

[181] Pierre Riteau. Building Dynamic Computing Infrastructures over Distributed Clouds. In *2011 IEEE International Symposium on Parallel and Distributed Processing: Workshops and Phd Forum*, pages 2097–2100, 2010. 122

[182] Pierre Riteau, Kate Keahey, and Christine Morin. Bringing Elastic MapReduce to Scientific Clouds. In *3rd Annual Workshop on Cloud Computing and Its Applications: Poster Session*, 2011. 84, 121

[183] Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker: Improving Live Migration of Virtual Clusters over WANs with Distributed Data Deduplication and Content-Based Addressing. In *17th International Euro-Par Conference on Parallel Processing*, pages 431–442, 2011. 116, 122

[184] Pierre Riteau, Maurício Tsugawa, Andréa Matsunaga, José Fortes, Tim Freeman, David LaBissoniere, and Kate Keahey. Sky Computing on FutureGrid and Grid'5000. In *5th Annual TeraGrid Conference: Poster Session*, 2010. 70, 121

[185] Pierre Riteau, Maurício Tsugawa, Andréa Matsunaga, José Fortes, and Kate Keahey. Large-Scale Cloud Computing Research: Sky Computing on FutureGrid and Grid'5000. *ERCIM News*, (83):41–42, October 2010. 70, 121

[186] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development*, 53(4):535–545, July 2009. 49

[187] Rusty Russell. virtio: towards a de-facto standard for virtual I/O devices. *SIGOPS Operating Systems Review*, 42(5), July 2008. 28

[188] Salvatore Sanfilippo. Redis. `http://redis.io`. 106, 149

[189] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Optimizing the Migration of Virtual Computers. In *5th Symposium on Operating Systems Design and Implementation*, pages 377–390, 2002. 103

[190] Michael C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009. 80, 146

[191] Matthias Schmidt, Niels Fallenbeck, Matthew Smith, and Bernd Freisleben. Efficient Distribution of Virtual Machines for Cloud Computing. In *2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2010)*, pages 567–574, 2010. 67

[192] Peter Sempolinski and Douglas Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 417–426, 2010. 42

[193] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010. 38

[194] Konstantin V. Shvachko. Apache Hadoop: The Scalability Update. *;login:*, 36(3):7–13, June 2011. 38

[195] Ezra Silvera, Gilad Sharaby, Dean Lorenz, and Inbar Shapira. IP Mobility to Support Live Migration of Virtual Machines Across Subnets. In *SYSTOR '09*, 2009. 94

[196] James E. Smith and Ravi Nair. The Architecture of Virtual Machines. *Computer*, 38(5):32–38, 2005. 22

[197] Marc Snir, Steve W. Otto, David W. Walker, Jack Dongarra, and Steven Huss-Lederman. *MPI: The Complete Reference*. MIT Press, 1995. 38

[198] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In *EuroSys '07*, pages 275–287, 2007. 29

[199] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, 2009. 42, 146

[200] W. Richard Stevens and Stephen A. Rago. *Advanced Programming in the UNIX® Environment (2nd Edition)*. Addison-Wesley Professional, 2005. 29

[201] Ion Stoica, Daniel Adkins, Shelley Zhuang, and Scott Shenker. Internet Indirection Infrastructure. *IEEE/ACM Transactions on Networking*, 12(2):205–218, August 2004. 93

[202] Petter Svärd, Benoît Hudzia, Johan Tordsson, and Erik Elmroth. Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 111–120. ACM, 2011. 102

[203] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the tenth ACM symposium on Operating systems principles*, pages 2–12. ACM, 1985. 32

[204] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sarma, Raghotham Murthy, and Hao Liu. Data Warehousing and Analytics Infrastructure at Facebook. In *2010 ACM SIGMOD International Conference on Management of Data*, pages 1013–1020, 2010. 38

[205] Niraj Tolia, Thomas Bressoud, Michael A. Kozuch, and Mahadev Satyanarayanan. Using Content Addressing to Transfer Virtual Machine State. Technical report, Intel Corporation, 2002. 103

[206] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees de Laat, Joe Mambretti, Inder Monga, Bas van Oudenaarde, Satish Raghunath, and Phil Wang. Seamless Live Migration of Virtual Machines over the MAN/WAN. *Future Generation Computer Systems*, 22(8):901–907, October 2006. 35

[207] Maurício Tsugawa and José A. B. Fortes. A Virtual Network (ViNe) Architecture for Grid Computing. In *20th International Parallel and Distributed Processing Symposium*, pages 1–10, March 2006. 48, 58

[208] Maurício Tsugawa, Andréa Matsunaga, and José Fortes. User-level Virtual Network Support for Sky Computing. In *e-Science 2009*, pages 72–79, 2009. 48, 58

[209] Maurício Tsugawa, Pierre Riteau, Andréa Matsunaga, and José Fortes. User-level Virtual Networking Mechanisms to Support Virtual Machine Migration Over Multiple Clouds. In *2nd IEEE International Workshop on Management of Emerging Networks and Services*, pages 568–572, 2010. 100, 122

[210] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kägi, Felix H. Leung, and Larry Smith. Intel Virtualization Technology. *Computer*, 38(5):48–56, 2005. 27

[211] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, January 2009. 40

[212] VMware. VMware ESX. `http://www.vmware.com/products/vsphere/esxi-and-esx/index.html`. 24

[213] VMware. VMware vCenter Converter. `http://www.vmware.com/products/converter/`. 46

[214] VMware. VMware Workstation. `http://www.vmware.com/products/workstation/`. 25

[215] Jens-S. Vöckler, Gideon Juve, Ewa Deelman, Mats Rynge, and G. Bruce Berriman. Experiences Using Cloud Computing for A Scientific Workflow Application. In *ScienceCloud '11*, pages 1–10, 2011. 69

[216] Carl A. Waldspurger. Memory resource management in VMware ESX server. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, 2002. 102, 148

[217] Edward Walker. Benchmarking Amazon EC2 for High-Performance Scientific Computing. *;login:*, 33(5):18–23, October 2010. 57

[218] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *Proceedings of the 25th Annual International Cryptology Conference (CRYPTO 2005)*, 2005. 106

[219] Romain Wartel, Tony Cass, Belmiro Moreira, Ewan Roche, Manuel Guijarro, Sebastien Goasguen, and Ulrich Schwickerath. Image Distribution Mechanisms in Large Scale Cloud Providers. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 112–117, 2010. 67

[220] Hidenobu Watanabe, Toshihiro Ohigashi, Tohru Kondo, Kouji Nishimura, and Reiji Aibara. A Performance Improvement Method for the Global Live Migration of Virtual Machine with IP Mobility. In *Proceedings of the Fifth International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2010)*, 2010. 94

[221] Timothy Wood, Prashant Shenoy, K. K. Ramakrishnan, and Jacobus Van der Merwe. CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2011. 34, 35, 93, 102

[222] Timothy Wood, Gabriel Tarasuk-Levin, Prashant Shenoy, Peter Desnoyers, Emmanuel Cecchet, and Mark Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2009. 102, 148

[223] Xianghua Xu, Feng Zhou, Jian Wan, and Yucheng Jiang. Quantifying Performance Properties of Virtual Machine. In *ISISE '08: Proceedings of the 2008 International Symposium on Information Science and Engieering*, pages 24–28, 2008. 28

[224] Xuxian Jiang Xu and Dongyan. VIOLIN: Virtual Internetworking on Overlay Infrastructure. In *ISPA 2004*, pages 937–946, 2004. 93

[225] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems. In *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006. 28

[226] Xiang Zhang, Zhigang Huo, Jie Ma, and Dan Meng. Exploiting Data Deduplication to Accelerate Live Virtual Machine Migration. In *2010 IEEE International Conference on Cluster Computing*, pages 88–96, 2010. 102

Appendix A

Résumé en français

A.1 Préambule

Depuis les débuts de l'informatique, les progrès continus de la technologie ont rendu les ordinateurs toujours plus accessibles et plus puissants. Grâce à ces avancées technologiques, l'informatique est devenue ubiquitaire et universelle. Cependant, l'augmentation des capacités de calcul a mené à de nouveaux problèmes à résoudre avec des besoins de plus en plus importants. L'industrie du logiciel a grandit rapidement et crée des applications de plus en plus importantes afin de répondre aux demandes des secteurs de la science, de l'industrie et du divertissement. D'importantes quantités de puissance de calcul sont requises pour simuler des problèmes complexes tels que les phénomènes naturels ou la mécanique des fluides, pour analyser des génomes, pour modéliser les marchés financiers, pour faire de la synthèse d'image, etc.

La complexité de ces problèmes les rend non solvables par de simples ordinateurs. Ainsi, le parallélisme et le calcul distribué ont été développés. Ces approches divisent les problèmes à résoudre en de multiples tâches, qui peuvent être exécutées en parallèle sur des systèmes informatiques composés d'un grand nombre de processeurs, tels que les supercalculateurs ou les grappes de calcul. Ces systèmes peuvent également être fédérés sous forme de grilles, créant des infrastructures distribuées géographiquement et appartenant à différentes organisations. Acquérir et gérer de tels systèmes est coûteux, ce qui limite leur utilisation aux organisations académiques et industrielles ayant les ressources financières et humaines suffisantes.

Une autre avancée des systèmes informatiques est celle des technologies de virtualisation. La virtualisation permet de séparer un environnement d'exécution du matériel sous-jacent, offrant une plus grande flexibilité dans la gestion des infrastructures informatiques. Bien que la virtualisation ait été populaire dans les années 1970, elle est restée limitée aux ordinateur centraux (*mainframes*) et ne s'est pas propagée au reste de l'industrie pendant plusieurs décennies. Ceci peut s'expliquer par la disponibilité de systèmes à temps partagé efficaces pour gérer les serveurs, ainsi que par l'omniprésence des ordinateurs personnels. Cependant, dans les années 2000, les technologies de virtualisation ont connu un formidable regain d'intérêt. Cet engouement s'explique par la disponibilité de systèmes de virtualisation avec un faible impact sur les performances, ainsi que par la transition vers des processeurs composés d'un grand nombre de cœurs, offrant ainsi la possibilité d'exécuter plusieurs systèmes en parallèle. En particulier, la virtualisation a été de plus en plus utilisée pour gérer les infrastructures informatiques

distribuées.

La combinaison des avancées des technologies de virtualisation avec les infrastructures informatiques distribuées, ainsi que la démocratisation des accès à Internet à haut débit, ont fait émerger un nouveau paradigme : l'informatique en nuage (*cloud computing*). Ce modèle offre des ressources informatiques à la demande, en utilisant des services accessibles par Internet. Il repose sur des idées circulant dans les milieux informatiques depuis des années, tels que le *utility computing* et les grilles de calcul. Les utilisateurs de fournisseurs d'informatique en nuage peuvent obtenir des ressources informatiques correspondant exactement à leurs besoins, sans avoir à investir dans du matériel coûteux, et en ne payant que pour les ressources réellement consommées.

Dans ce nouveau contexte, plusieurs problèmes se posent. Comment peut-on facilement et efficacement exécuter des applications nécessitant de grandes quantités de puissance de calcul sur des infrastructures d'informatique en nuage ? Comment peut-on tirer avantage des nouvelles possibilités offertes par les technologies de virtualisation dans ces infrastructures ?

A.2 Objectifs

Étant donné la popularité et la disponibilité croissante des nuages informatiques, ainsi que des systèmes virtualisés en général, cette thèse soutient qu'*il est possible de créer facilement et efficacement des plates-formes d'exécution élastiques à large échelle en fédérant de multiples infrastructures d'informatique en nuage*, et que *ces plates-formes d'exécution peuvent être migrées à chaud entre différents nuages informatiques de façon efficace, les rendant plus dynamiques que les infrastructures traditionnelles*.

Les utilisateurs devant résoudre des problèmes nécessitant de grandes quantités de puissance de calcul, qu'ils soient des domaines de la simulation scientifique, de la finance, de la fouille de données, etc., ont maintenant accès à de multiples infrastructures d'informatique en nuage de différents types. Ces infrastructures offrent beaucoup de contrôle aux utilisateurs, qui n'ont pas nécessairement les connaissances pour utiliser au mieux ces ressources. Ces multiples infrastructures d'informatique en nuage offrent le potentiel d'être utilisées simultanément, en suivant une approche appelée *sky computing* [130]. De plus, ces infrastructures offrent une élasticité avec des moyens pour créer et détruire des ressources à la demande, afin de répondre à des changements de besoins des utilisateurs ou des applications. Le premier objectif de cette thèse est de *facilement* et *efficacement* créer des *plates-formes d'exécution élastiques à large échelle* à partir de *ressources de nuages informatiques fédérés*, appartenant potentiellement à différents fournisseurs et institutions, en utilisant l'approche *sky computing*.

Une différence importante entre les infrastructures de calcul traditionnelles et les nuages informatiques est l'utilisation de la virtualisation comme mécanisme fondamental pour la création et la gestion des ressources. En séparant le logiciel de l'infrastructure matérielle sous-jacente, les environnements virtuels ne sont plus liés au matériel utilisé pour leur exécution. Cette indépendance permet aux plates-formes d'exécutions d'être migrées entre machines distantes à travers des connexions réseaux, tout comme des transferts de données classiques. Cette propriété, mise en œuvre de manière transparente par un mécanisme appelé *migration à chaud*, permet aux plates-formes d'exécution d'être dynamiquement relocalisées entre infrastructures d'informatique en nuage, afin

de tirer partie de changements de disponibilité, de coût ou de performance des ressources, et ainsi d'optimiser l'exécution des calculs. Dans ce contexte, le second objectif de cette thèse est de rendre *plus dynamiques* les plates-formes d'exécutions fondées sur l'informatique en nuage en permettant leur migration à chaud efficace entre différents nuages informatiques.

A.3 Contributions

Afin de remplir les objectifs de cette thèse, nous proposons l'ensemble de contributions suivant :

- Des mécanismes pour la création efficace de plates-formes d'exécution élastiques à large échelle, au-dessus de nuages informatiques fédérés en utilisant l'approche *sky computing*,
- Resilin, un système pour facilement créer des plates-formes d'exécution au-dessus de ressources de nuages informatiques distribués, afin d'exécuter des calculs MapReduce,
- Des mécanismes permettant aux machines virtuelles de réaliser des communications réseaux efficaces en présence de migrations à chaud entre nuages informatiques,
- Shrinker, un protocole de migration à chaud optimisé pour la migration à chaud efficace de grappes de machines virtuelles entre nuages informatiques connectés par des réseaux étendus.

Ces travaux ont été réalisés dans l'équipe de recherche Myriads (précédemment équipe-projet PARIS), qui fait partie du laboratoire IRISA ainsi que du centre de recherche Inria Rennes - Bretagne Atlantique, ainsi que dans le cadre de collaborations avec l'équipe du projet Nimbus à Argonne National Laboratory / University of Chicago Computation Institute et avec le laboratoire ACIS à l'Université de Floride.

A.3.1 Mécanismes pour la création efficace de plates-formes d'exécution élastiques à large échelle, au-dessus de nuages informatiques fédérés en utilisant l'approche *sky computing*

De nombreux nuages informatiques sont maintenant disponibles, présentant des types de déploiement différents : publics, privés ou communautaires. Dans ce contexte, la fédération de ressources de calcul provenant de multiples nuages informatiques permet de construire des plates-formes d'exécution à large échelle. L'approche dite de *sky computing* fédère des nuages informatiques de manière transparente, afin de créer des plates-formes d'exécution pour des applications nécessitant d'importantes quantités de puissance de calcul.

Dans ces travaux, nous avons réalisé une série d'expérimentations utilisant l'approche *sky computing* pour créer des plates-formes d'exécution intercontinentales à large échelle. Pour cela, nous avons utilisé des nuages informatiques Nimbus [27, 128] déployés sur les infrastructures expérimentales Grid'5000 [79] en France et FutureGrid [15]

aux États-Unis. Nous avons rendu élastiques ces plates-formes de type *sky computing* en les étendant afin d'accélérer les calculs. Les leçons tirées de ces expérimentations nous ont mené à concevoir deux nouveaux mécanismes de propagation de machines virtuelles et à les mettre en œuvre dans le système de nuage informatique Nimbus. Ces mécanismes, fondés sur une chaîne de diffusion et sur des volumes utilisant la copie sur écriture, permettent de réduire le temps requis pour créer et étendre de façon élastique des plates-formes d'exécution à large échelle sur des nuages informatiques.

Ces travaux ont été réalisés en collaboration avec Kate Keahey (Argonne National Laboratory / Computation Institute, University of Chicago), Maurício Tsugawa, Andréa Matsunaga et José Fortes (University of Florida). Ils ont été démarrés durant une visite de 3 mois au laboratoire ACIS à l'Université de Floride à Gainesville, États-Unis, pendant l'été 2010.

A.3.1.1 L'approche *sky computing*

L'approche *sky computing*, proposée par Keahey et al. [130], fédère de multiples nuages informatiques de façon transparente. Les ressources de différents nuages sont rassemblées dans un environnement réseau unifié, afin de répliquer une plate-forme traditionnelle de grappe de calcul. Ainsi, une plate-forme de type *sky computing* peut exécuter des applications non modifiées tout comme une grappe de calcul. Cette approche utilise principalement deux outils : le réseau virtuel ViNe et les outils de contextualisation de Nimbus.

ViNe est un logiciel de réseau virtuel fondé sur un réseau *overlay*, permettant d'unifier de multiples réseaux physiques en un seul réseau virtuel. Il permet d'offrir une connectivité directe entre tous les participants d'un réseau virtuel, même lorsque certains réseaux physiques utilisent un adressage IP privé ou un pare-feu, ce qui est généralement le cas pour les nuages informatiques privés. Cette connectivité directe est un besoin essentiel pour de nombreuses applications conçues pour des environnements de grappes de calcul.

Les outils de contextualisation de Nimbus permettent d'automatiser le déploiement et la configuration de machines virtuelles dans les nuages informatiques. Ces outils assignent des rôles aux membres d'une grappe de machines virtuelles, afin de créer une plate-forme d'exécution cohérente.

A.3.1.2 Expérimentations de plates-formes de type *sky computing* à large échelle

Lors de nos expérimentations, nous avons créé des plates-formes de type *sky computing* à large échelle, en utilisant des nuages informatiques Nimbus [27, 128] déployés sur les infrastructures expérimentales Grid'5000 [79] en France et FutureGrid [15] aux États-Unis. Sur ces plates-formes, nous avons exécuté BLAST (Basic Local Alignment Search Tool [55, 162], une application localisant les régions similaires dans des séquences génomiques.

Les nuages informatiques offrent la possibilité de modifier à la volée les demandes de ressources, rendant les plates-formes d'exécution élastiques. Nous nous sommes intéressés aux capacités d'extension élastiques de ces plates-formes de type *sky computing*. Nous avons étendu de manière élastique ces plates-formes, pendant l'exécution de l'application BLAST, afin de fournir plus de puissance de calcul et ainsi d'accélérer le

progrès de l'application.

A.3.1.3 Passage à l'échelle des infrastructures d'informatique en nuage

Lors de ces expérimentations, il est apparu que l'élément principal limitant les performances de la création et de l'extension élastique de plates-formes d'exécution était la propagation des images de machines virtuelles. En effet, chaque machine virtuelle nécessite sa propre copie de l'image de machine virtuelle contenant l'installation du système d'exploitation et des applications. Le système Nimbus utilisait un système de copie centralisé depuis un serveur de stockage d'images vers chacun des nœuds hébergeant les machines virtuelles. Ainsi, les performances sont limitées par les capacités de transfert de ce serveur de stockage.

Pour résoudre ce problème, nous avons conçu deux nouveaux mécanismes de propagation d'images de machines virtuelles. Ces mécanismes améliorent les performances de la propagation de la même image de machines virtuelles vers de multiples nœuds physiques. Le premier mécanisme est fondé sur les programmes Kastafior et TakTuk [88] développés à Inria. Ces outils permettent de créer une chaîne de diffusion de données entre de nombreux nœuds. Nous utilisons cette chaîne pour diffuser le contenu des images de machines virtuelles. Lorsque la même image est requise sur tous les nœuds physiques, le serveur centralisé n'a besoin de réaliser qu'une seule copie, les données étant ensuite relayées entre chaque nœud de l'infrastructure. Ce mécanisme divise le temps de propagation par le nombre de copies requises de l'image.

Le second mécanisme est fondé sur des volumes utilisant la copie sur écriture. Afin d'éviter la création d'une nouvelle copie d'image pour chaque allocation de machine virtuelle, cette approche crée un volume utilisant la copie sur écriture. La copie sur écriture stocke les données modifiées par chaque machine virtuelle dans un fichier séparé, permettant ainsi de partager la même image de base en lecture seule. En utilisant une pré-propagation des images de base sur chaque nœud physique, nous avons montré que des grappes de machines virtuelles peuvent être allouées en quelques secondes au lieu de plusieurs minutes ou même heures avec les approches classiques.

A.3.2 Resilin, un système pour facilement créer des plates-formes d'exécution de calculs MapReduce au-dessus de ressources de nuages informatiques distribués

Afin de permettre l'utilisation facile de ressources de nuages informatiques distribués, nous avons construit Resilin, un système pour créer des plates-formes d'exécution de calculs MapReduce [96, 97]. Resilin met en œuvre l'API du service web Amazon Elastic MapReduce et utilise des ressources de nuages informatiques privés et communautaires. Resilin prend en charge l'allocation, la configuration et la gestion de plates-formes d'exécution Hadoop [1], en utilisant potentiellement plusieurs nuages informatiques. Dans ce document, nous décrivons la conception et la mise en œuvre de Resilin. Nous présentons également les résultats d'une étude comparative avec le service Amazon Elastic MapReduce. Ces travaux ont été réalisés en collaboration avec Ancuţa Iordache, étudiante de Master à West University of Timişoara en Roumanie, durant son stage de Master dans l'équipe Myriads.

A.3.2.1 Amazon Elastic MapReduce

Amazon Elastic MapReduce [58] est un service web permettant d'exécuter des calculs MapReduce sur l'infrastructure Amazon EC2 [57]. Les utilisateurs soumettent des séquences de calculs MapReduce sous forme de *job flows*. Le service prend en charge la création d'une plate-forme d'exécution Hadoop sur l'infrastructure EC2 afin d'exécuter les calculs MapReduce. Les données d'entrée sont lues directement depuis le système de stockage Amazon S3 [59]. Le système de fichiers distribué de Hadoop, HDFS, stocke les données intermédiaires des calculs. Enfin, les données résultat sont également stockées dans Amazon S3.

Amazon Elastic MapReduce offre un service simple pour les utilisateurs ne souhaitant pas configurer et gérer eux-mêmes une plate-forme d'exécution MapReduce. Cependant, il est limité à l'utilisation de ressources de Amazon EC2, empêchant les utilisateurs de tirer partie de ressources de nuages informatiques privés ou communautaires. Les possibilités de personnalisation de l'environnement d'exécution Hadoop sont limitées. Enfin, ce service est offert pour un coût additionnel à celui des ressources Amazon EC2.

A.3.2.2 Architecture, mise en œuvre et évaluation de Resilin

Pour répondre à ces besoins, nous avons conçu Resilin, un système mettant en œuvre l'API du service Amazon Elastic MapReduce avec des ressources de nuages informatiques privés ou communautaires. Resilin permet également d'exécuter des calculs MapReduce avec des ressources provenant de multiples nuages informatiques.

En se fondant sur l'utilisation de l'API d'Amazon EC2 pour l'allocation de machines virtuelles, Resilin est compatible avec de nombreux systèmes de nuages informatiques *open source*, tels que Eucalyptus [13, 166], Nimbus [27, 128] et OpenNebula [33, 199]. Pour mettre en œuvre Resilin, nous avons modifié Hadoop pour permettre son intégration avec le système de stockage Cumulus [76] utilisé dans les nuages informatiques Nimbus. Ainsi, les données d'entrée et résultat des calculs MapReduce peuvent être stockées dans l'infrastructure d'un nuage informatique privé ou communautaire.

Nous avons évalué les performances de Resilin avec deux applications exemples fournies par Amazon, Word Count [63] et CloudBurst [190, 61] en utilisant des nuages informatiques Nimbus déployés sur Grid'5000. Les résultats montrent que Resilin offre des performances similaires à Amazon Elastic MapReduce.

A.3.3 Des mécanismes permettant aux machines virtuelles de réaliser des communications réseaux efficaces en présence de migrations à chaud entre nuages informatiques

Lors de la migration à chaud de machines virtuelles entre différents nuages informatiques, l'infrastructure réseau doit être adaptée pour permettre aux machines virtuelles de continuer à communiquer. Nous proposons des mécanismes pour automatiquement détecter les migrations à chaud et reconfigurer les réseaux virtuels afin de garder les communications ininterrompues. Nous proposons également un schéma de routage optimisant les performances des communications réseaux lorsque les machines virtuelles sont situées sur le même réseau physique. Nous avons mis en œuvre un prototype de nos mécanismes pour le réseau virtuel ViNe et l'avons validé de manière expérimentale.

Ces travaux ont été réalisés en collaboration avec Maurício Tsugawa, Andréa Matsunaga et José Fortes durant une visite de 3 mois au laboratoire ACIS à l'Université de Floride à Gainesville, États-Unis, pendant l'été 2010.

A.3.3.1 Support réseau pour la migration à chaud entre nuages informatiques

La migration à chaud de machines virtuelles a été initialement conçue dans le cadre de réseaux locaux. Dans ce cas, le réseau physique est reconfiguré par de simples mécanismes de diffusion de messages ARP ou RARP sur le réseau physique. La migration à chaud entre différents nuages informatiques ne peut utiliser de tels mécanismes, car les réseaux physiques des nuages sont séparés. Ainsi, des mécanismes de gestion de la mobilité des machines virtuelles sont requis afin d'utiliser la migration à chaud entre nuages informatiques.

Différentes approches ont été proposées afin de résoudre ce problème, fondées sur le protocole Mobile IP [173, 174], sur des réseaux overlay, ou sur des mécanismes utilisant des messages ARP et des tunnels. Les mécanismes utilisant des messages ARP offrent l'avantage de ne pas nécessiter de support de protocoles tels que Mobile IP dans les systèmes d'exploitation des machines virtuelles, ainsi que de minimiser l'impact sur les performances réseaux.

A.3.3.2 Mécanismes de reconfiguration du réseau pour la migration à chaud

Dans le cadre de nuages informatiques connectés par un réseau virtuel, nous avons proposé un ensemble de mécanismes afin de détecter automatiquement les migrations à chaud dans un nuage informatique et ainsi d'entraîner la reconfiguration du réseau virtuel afin de garder les communications ininterrompues.

Nos mécanismes détectent les migrations en capturant les messages ARP ou RARP envoyés par les hyperviseurs lors de migrations à chaud. Ceci entraîne la reconfiguration du réseau virtuel par l'échange de messages entre les nœuds gérant l'infrastructure réseau virtuelle. Nous utilisons des réponses ARP non sollicitées afin de rediriger le trafic envoyé et reçu par la machine virtuelle migrée.

De plus, nous proposons un schéma de routage permettant de maximiser les performances des communications entre machines virtuelles présentes sur le même réseau physique. En configurant les machines virtuelles pour qu'elles appartiennent à un même sous-réseau virtuel rassemblant les réseaux physiques de différents nuages, la machine virtuelle migrée peut communiquer directement avec les machines virtuelles situées sur le même réseau physique, sans passer par l'infrastructure du réseau virtuel. Nous avons mis en œuvre ces mécanismes dans le réseau virtuel ViNe, développé à l'Université de Floride.

A.3.4 Shrinker, un protocole de migration à chaud optimisé pour la migration à chaud efficace des grappes de machines virtuelles entre nuages informatiques connectés par des réseaux étendus

La migration à chaud permet de relocaliser de façon dynamique des plates-formes d'exécution entre différentes infrastructures d'informatique en nuage. La migration à chaud entre multiples nuages informatiques offre de nombreux avantages, aussi bien pour les

utilisateurs que pour les administrateurs. Cependant, la grande quantité de trafic générée lors du transfert de grappes de machines virtuelles rend la migration à chaud inefficace sur les réseaux étendus. Nous proposons Shrinker, un protocole de migration à chaud qui élimine les données dupliquées entre machines virtuelles afin de réduire le trafic généré par la migration à chaud entre nuages informatiques. Shrinker utilise une déduplication de données distribuée ainsi qu'un adressage distribué par contenu afin de réaliser cet objectif. Nous avons mis en œuvre un prototype de Shrinker dans l'hyperviseur KVM [132]. Nos évaluations sur Grid'5000 montrent que Shrinker réduit aussi bien le trafic généré par la migration à chaud que le temps total de migration.

A.3.4.1 Présence de données dupliquées dans une grappe de machines virtuelles

Une grappe de machines virtuelles est généralement fondée sur de multiples instanciations du même environnement d'exécution, comprenant le système d'exploitation ainsi que les applications. Ainsi, de nombreuses études ont montré que la mémoire vive de telles machines virtuelles contient d'importantes quantités de données dupliquées [66, 112, 154, 216, 222]. De même, les disques des machines virtuelles contiennent aussi de telles données dupliquées [138, 160, 180].

Lors de la migration à chaud d'une grappe virtuelle entre différents nuages informatiques, ces données doivent être transférées sur un réseau étendu. Ainsi, de nombreuses copies identiques des mêmes données sont transférées sur le réseau. L'approche proposée par Shrinker est d'éliminer ces données dupliquées lors de la migration à chaud. Ceci diminue la quantité de données à transférer sur le réseau étendu, permettant ainsi de réduire le temps de migration.

A.3.4.2 Architecture de Shrinker

Shrinker est fondé sur deux mécanismes : une déduplication de données distribuée sur le site d'origine de la migration à chaud, ainsi qu'un adressage distribué par contenu sur le site de destination. Le contenu des données (pages mémoire ou blocs disque) est identifié par des fonctions de hachage cryptographique.

Sur le site d'origine, les hyperviseurs interagissent avec un service de coordination pour déclarer les données qu'ils souhaitent envoyer sur le site de destination. Lorsqu'une page est détectée comme ayant déjà été transférée, l'hyperviseur envoie à la place l'empreinte résultant du calcul de la fonction de hachage sur le contenu à envoyer. Cette empreinte est utilisée comme identifiant unique des données. Comme les empreintes sont de taille largement inférieure à celle des données, l'utilisation du réseau étendu est réduite.

Sur le site destination, les empreintes sont utilisées pour localiser les données après réception. Les hyperviseurs du site destination interagissent avec un service d'indexation qui recense la localisation des données pour chaque empreinte. Ainsi, les hyperviseurs s'échangent les pages mémoire et les blocs disque sur le réseau local de destination plutôt que sur le réseau étendu.

A.3.4.3 Mise en œuvre et évaluation de Shrinker

Nous avons mis en œuvre un prototype de Shrinker dans l'hyperviseur KVM [132]. Les services de coordination et d'indexation sont réalisés avec Redis [188], un système de stockage clé-valeur à haute performance.

Nous avons évalué Shrinker sur l'infrastructure Grid'5000 en émulant des réseaux étendus avec différentes bandes passantes. Nous avons migré des plates-formes d'exécution sur ces réseaux émulés et mesuré les quantités de données transférées ainsi que le temps total de migration. Les résultats montrent que Shrinker permet de réduire ces deux métriques, augmentant ainsi les performances de la migration à chaud de grappes virtuelles entre nuages informatiques.

A.4 Publications

Les contributions présentées dans ce manuscrit ont été publiées dans plusieurs livres, journaux, conférences, workshops et magasines avec comité de lecture.

Chapitres de livres

- Andréa Matsunaga, Pierre Riteau, Maurício Tsugawa, and José Fortes. Crosscloud Computing. In *High Performance Computing : From Grids and Clouds to Exascale*, volume 20 of *Advances in Parallel Computing*, pages 94–108. IOS Press, 2011.

Articles de journaux internationaux

- Christine Morin, Yvon Jégou, Jérôme Gallard, and Pierre Riteau. Clouds : a New Playground for the XtreemOS Grid Operating System. *Parallel Processing Letters*, 19(3) :435–449, September 2009.

Publications dans des conférences internationales

- Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker : Improving Live Migration of Virtual Clusters over WANs with Distributed Data Deduplication and Content-Based Addressing. In *17th International Euro-Par Conference on Parallel Processing*, pages 431–442, 2011.

Publications dans des workshops internationaux

- Maurício Tsugawa, Pierre Riteau, Andréa Matsunaga, and José Fortes. User-level Virtual Networking Mechanisms to Support Virtual Machine Migration Over Multiple Clouds. In *2nd IEEE International Workshop on Management of Emerging Networks and Services*, pages 568–572, 2010.

Posters dans des conférences et workshops internationaux

- Pierre Riteau. Building Dynamic Computing Infrastructures over Distributed Clouds. In *2011 IEEE International Symposium on Parallel and Distributed Processing : Workshops and PhD Forum*, pages 2097–2100, 2011. Best Poster Award.

- Pierre Riteau, Kate Keahey, and Christine Morin. Bringing Elastic MapReduce to Scientific Clouds. In *3rd Annual Workshop on Cloud Computing and Its Applications : Poster Session*, 2011.
- Pierre Riteau, Maurício Tsugawa, Andréa Matsunaga, José Fortes, Tim Freeman, David LaBissoniere, and Kate Keahey. Sky Computing on FutureGrid and Grid'5000. In *5th Annual TeraGrid Conference : Poster Session*, 2010.

Articles de magazines internationaux

- Pierre Riteau, Maurício Tsugawa, Andréa Matsunaga, José Fortes, and Kate Keahey. Large-Scale Cloud Computing Research : Sky Computing on FutureGrid and Grid'5000. *ERCIM News*, (83) :41–42, Octobre 2010.

Rapports de recherche

- Pierre Riteau, Ancuţa Iordache, and Christine Morin. Resilin : Elastic MapReduce for Private and Community Clouds. Research report RR-7767, *Institut national de recherche en informatique et en automatique (Inria)*, Octobre 2011.
- Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker : Efficient Wide-Area Live Virtual Machine Migration using Distributed Content-Based Addressing. Research report RR-7198, *Institut national de recherche en informatique et en automatique (Inria)*, Février 2010.
- Christine Morin, Yvon Jégou, Jérôme Gallard, and Pierre Riteau. Clouds : a New Playground for the XtreemOS Grid Operating System. Research report RR-6824, *Institut national de recherche en informatique et en automatique (Inria)*, Février 2009.

Rapports techniques

- Eliana-Dina Tîrşa, Pierre Riteau, Jérome Gallard, Christine Morin, and Yvon Jégou. Towards XtreemOS in the Clouds – Automatic Deployment of XtreemOS Resources in a Nimbus Cloud. Technical report RT-0395, *Institut national de recherche en informatique et en automatique (Inria)*, Octobre 2010.

A.5 Organisation du manuscrit

Le reste de ce manuscrit est organisé en quatre parties. La première partie, constituée du chapitre 2, présente l'état de l'art du contexte de nos travaux. Nous faisons une vue d'ensemble de la virtualisation, et nous intéressons plus spécifiquement aux technologies que nous utilisons dans nos contributions. Nous décrivons le parallélisme et le calcul distribué, ainsi que les différents types d'applications et d'architectures qui sont utilisées pour résoudre des problèmes nécessitant d'importantes quantités de puissance de calcul. Ensuite, nous introduisons le concept de l'informatique en nuage, combinant la virtualisation et les systèmes distribués. Enfin, nous nous intéressons aux efforts actuels portant sur la fédération de nuages informatiques.

La deuxième partie présente nos contributions pour la création facile et efficace de plates-formes d'exécution élastiques à large échelle à partir de ressources de nuages

informatiques fédérés, en utilisant l'approche *sky computing*. Le chapitre 3 présente nos expérimentations avec des plates-formes de type *sky computing* à large échelle, réalisées sur les infrastructures expérimentales Grid'5000 et FutureGrid. À partir des résultats obtenus durant ces expérimentations, nous proposons deux nouveaux mécanismes de propagation d'images de machines virtuelles. Ils permettent de créer plus efficacement des plates-formes d'exécution dans les nuages informatiques. Nous rendons également ces plates-formes de type *sky computing* élastiques, en les étendant afin d'accélérer les calculs. Le chapitre 4 présente Resilin, un système pour facilement exécuter des calculs MapReduce sur des ressources de nuages informatiques distribués. Nous présentons l'architecture et la mise en œuvre de Resilin, ainsi qu'une évaluation comparative avec le système Amazon Elastic MapReduce.

La troisième partie décrit nos contributions pour rendre ces plates-formes d'exécution plus dynamiques, ceci en utilisant la migration de machines virtuelles entre nuages informatiques. Le chapitre 5 décrit un ensemble de mécanismes qui permettent aux machines virtuelles de continuer à communiquer efficacement en présence de migrations à chaud entre nuages informatiques, ainsi que leur mise en œuvre pour le logiciel de réseau virtuel ViNe. Le chapitre 6 présente Shrinker, un système optimisant la migration à chaud de grappes de machines virtuelles entre nuages informatiques connectés par des réseaux étendus. Nous décrivons l'architecture de Shrinker, sa mise en œuvre dans l'hyperviseur KVM, ainsi que les résultats de son évaluation sur Grid'5000.

La quatrième et dernière partie termine ce manuscrit avec le chapitre 7 qui résume nos contributions et présente des perspectives de recherche pour le futur.

Dynamic Execution Platforms over Federated Clouds

Abstract

The increasing needs for computing power have led to parallel and distributed computing, which harness the power of large computing infrastructures in a concurrent manner. Recently, virtualization technologies have increased in popularity, thanks to hypervisors improvements, the shift to multi-core architectures, and the spread of Internet services. This has led to the emergence of cloud computing, a paradigm offering computing resources in an elastic, on-demand approach while charging only for consumed resources.

In this context, this thesis proposes four contributions to leverage the power of multiple clouds. They follow two directions: the creation of elastic execution platforms on top of federated clouds, and inter-cloud live migration for using them in a dynamic manner. We propose mechanisms to efficiently build elastic execution platforms on top of multiple clouds using the sky computing federation approach. Resilin is a system for creating and managing MapReduce execution platforms on top of federated clouds, allowing to easily execute MapReduce computations without interacting with low level cloud interfaces. We propose mechanisms to reconfigure virtual network infrastructures in the presence of inter-cloud live migration, implemented in the ViNe virtual network from University of Florida. Finally, Shrinker is a live migration protocol improving the migration of virtual clusters over wide area networks by eliminating duplicated data between virtual machines.

Keywords

Cloud computing, parallel and distributed computing, large-scale systems, elasticity, dynamic execution platforms, live migration, MapReduce, federated clouds

Résumé

Les besoins croissants en ressources de calcul ont mené au parallélisme et au calcul distribué, qui exploitent des infrastructures de calcul large échelle de manière concurrente. Récemment, les technologies de virtualisation sont devenues plus populaires, grâce à l'amélioration des hyperviseurs, le passage vers des architectures multi-cœur, et la diffusion des services Internet. Cela a donné lieu à l'émergence de l'informatique en nuage, un paradigme qui offre des ressources de calcul de façon élastique et à la demande, en facturant uniquement les ressources consommées.

Dans ce contexte, cette thèse propose quatre contributions pour tirer parti des capacités de multiples nuages informatiques. Elles suivent deux directions : la création de plates-formes d'exécution élastiques au-dessus de nuages fédérés, et la migration à chaud entre nuages pour les utiliser de façon dynamique. Nous proposons des mécanismes pour construire de façon efficace des plates-formes d'exécution élastiques au-dessus de plusieurs nuages utilisant l'approche de fédération *sky computing*. Resilin est un système pour créer et gérer des plates-formes d'exécution MapReduce au-dessus de nuages fédérés, permettant de facilement exécuter des calculs MapReduce sans interagir avec les interfaces bas niveau des nuages. Nous proposons des mécanismes pour reconfigurer des infrastructures réseau virtuelles lors de migrations à chaud entre nuages, mis en œuvre dans le réseau virtuel ViNe de l'Université de Floride. Enfin, Shrinker est un protocole de migration à chaud améliorant la migration de grappes de calcul virtuelles dans les réseaux étendus en éliminant les données dupliquées entre machines virtuelles.

Mots clés

Informatique en nuage, parallélisme et calcul distribué, systèmes à large échelle, élasticité, plates-formes d'exécution dynamiques, migration à chaud, MapReduce, nuages informatiques fédérés