Contributions to Large-scale Distributed Systems The infrastructure Viewpoint - Toward Fog and Edge Computing as The Next Utility Computing Paradigm?

Adrien Lebre

Université Bretagne Loire

Habilitation à Diriger des recherches

Presentée par

Adrien Lebre

École Doctorale Sciences et technologies de l'information, et mathématiques
Discipline : Informatique et applications, section CNU 27
Unité de recherche : IMT Atlantique, Inria, LS2N
Soutenue le 1er Sept. 2017

Contributions to Large-scale Distributed Systems The infrastructure Viewpoint

Toward Fog and Edge Computing as The Next Utility Computing Paradigm?

Contributions aux systèmes distribués à large échelle la vision infrastructure

Vers le Fog et l'Edge computing comme nouveau modèle d'informatique utilitaire ?

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

M. Claude Jard, Président
Professor in Computer Science at University of Nantes
Head of Nantes Digital Science Laboratory (LS2N).
M. Erik Elmroth, Rapporteur
Professor in Computing Science at Umea University, Sweden
Head of Umea University research on distributed systems.
M. Manish Parashar, Rapporteur
Distinguished Professor of Computer Science at Rutgers University, USA
Director of the Rutgers Discovery Informatics Institute (RDI2).
M. Pierre Sens, Rapporteur
Professor at the University of Pierre et Marie Curie (Paris 6), France
Head of the REGAL Resarch Group.
M. Frédéric Desprez, Examinateur
Senior Researcher at Inria
Deputy Scientific Director at Inria

It does not matter how slowly you go
as long as you do not stop.

— Confucius

Acknowledgments

Because I see now how overloaded can be the schedule of senior researchers, I would like to start these acknowledgments by expressing my gratitude to all the members of the committee for accepting to evaluate m work despite their respective obligations. It is a real honor for me to have such a prestigious committee.

This document covers most of the activities I made since the defense of my Phd in 2006. During these ten years, I had the opportunities to work with many French as well as Foreign colleagues. While these are acknowledged throughout the document, I would like, here, to warmly thank all of them for the collaborations we had and that have made the researcher I am today. I should highlight that I am much indebted to

- Dr. Christine Morin for showing me what a community can accomplish ;
- Prof. Mario Südholt for the freedom and trust he has given me in each action I have wanted to achieve since my arrival in his team ;
- Dr. Frédéric Desprez for his confidence and guidance he has never stopped to give me during this period.

Special thanks go to the Phd candidates, Post-docs and Engineers I had the chance to work with. In addition to performing the real work that makes research progressing, they have been key persons that through their competences and efforts have kept my enthusiasm and my energy to conduct research activities intact. The contributions to the field of distributed computing I am glad to present in this document could not have been achieved without them.

Finally, my most affectionate thanks go to:

- My Friends who are throught our weekend outings and meals the best way for me to “recharge my batteries” ;
- My parents-in-law who indirectly have participated to the work presented in this manuscript, in particular by taking care of my kids each time it has been necessary ;
- My parents who after 39 years still continue to impress me for the devotion they have for their five children ;
- My brothers, sister(s) (in-law) and their kids with whom I should have spent much more times than what I have done ;
- My children, Elie-Lou and Jonas who are, by their daily smiles, an endless source of happiness ;
- And, my lovely wife, Dorothée, who is, in good times and bad, the most wonderful person I have met.

Table of Contents

Part I

INTRODUCTION

This habilitation thesis presents an overview of the major research activities I conducted as part of the ASCOLA Research Group of the Ecole des Mines de Nantes (now IMT Atlantique), where I have held an Associate Professor position since October 2008. ASCOLA is a joint team of the Automation, Production and Computer Science Department of IMT Atlantique, the Inria research center in Rennes, and the "Laboratoire des Sciences du Numérique de Nantes" (LS2N, UMR CNRS 6004) in France. The team addresses the general problem of the implementation and evolution of large and complex software architectures. In particular, we pursue the objective of new language abstractions and implementation techniques for large-scale applications in cloud- and grid-based systems, both on the level of (service-based) applications and (virtualized) infrastructures. The document summarizes the main contributions I achieved on this topic.

I highlight that each part includes, at the beginning, a short preamble such as this one. The goal is to give additional information regarding the work that is presented. In particular, for each contribution, I mention the different persons that took part to the activities and, as appropriate, the sources of funding that have been associated.

Last but not the least, all URLs that are given in this manuscript have been validated on March 2017.

Introduction

1

In this introduction, I first present the scientific context of my research and second highlight three main actions I have done since I have joined the ASCOLA Research Group.

To conclude this introduction, I voluntary chose to give a brief overview of the activities I have been doing along the ones presented in this manuscript. Reading of this last section is not mandatory to understand the scientific aspects that will be discussed later on. The goal is to highlight a few details that I consider as key elements in my professional life such as my engagement in the Grid'5000 consortium or more generally my involvement in the scientific community – Being involved in the community has been and will be a key element of my research activities – I have always been convinced of the importance to promote collaborations and exchanges between researchers: collaborations is the answer to tackle scientific challenges that would be almost impossible to address alone.

1.1 General Context

To accommodate the ever-increasing demand for computation and storage capabilities, computing infrastructures have continuously evolved since the 1960's. Figure 1 provides a chronology of major advances in communication and computation technologies. It depicts in particular the different computing architectures starting from the historical mainframe computers to the advent of Cloud Computing solutions that host most IT-services nowadays.

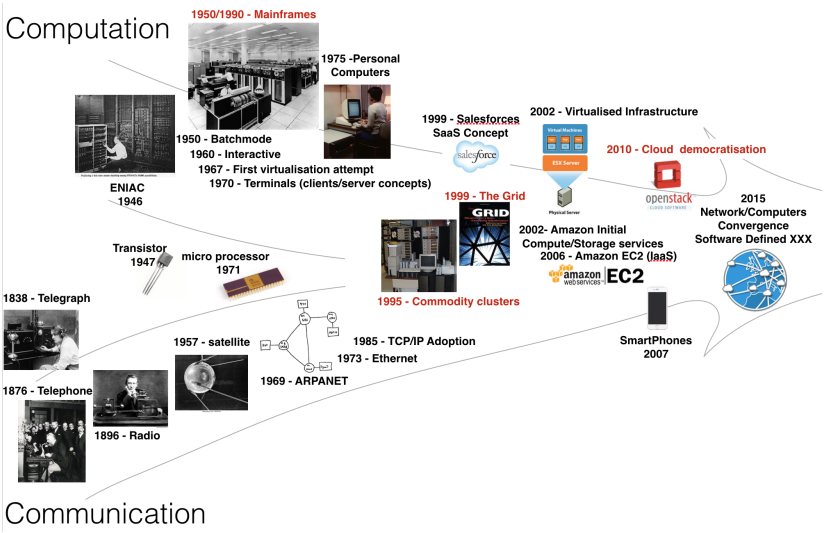


Figure 1: *Evolution of Communication/Computation Facilities*
As early as the advent of the TCP/IP network, Utility Computing infrastructures have been leveraging network backbones to deliver more efficient solutions.

Although these architectures radically differ in their design and the way of using them, mainframes, commodity clusters, Grids and more recently Cloud Computing facitilies have all been targeting the *Utility Computing* concept for IT capabilities as defined by John McCarthy in 1961.

Because it enables Utility Computing providers to go beyond the capacity of a single machine, distributed computing became the prevalent infrastructure for delivering IT as a service in the middle of the 1990's. However, due to the complexity to operate and use Distributed Computing Platforms, it has taken almost two decades

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 - John Mc Carthy, Speaking at the MIT centennial in 1961

of research activities and technology transfers (Foster and Kesselman, 2010) to popularize them in all domains of science and industry (Armbrust et al., 2010; Low, Chen, and Wu, 2011). Among the key enablers of this democratization are virtualization technologies. Proposed as early as Mainframes (Popek and Goldberg, 1974) but used for about a decade in distributed computing (Fallenbeck et al., 2006; Sotomayor, Keahey, and Foster, 2008), system virtualization had a huge impact on distributed computing.

Virtual Machines have finally arrived. Dismissed for a number of years as merely academic curiosities, they are now seen as cost-effective techniques for organizing computer systems resources to provide extraordinary system flexibility and support for certain unique applications. Goldberg, 1974, A Survey of Virtual Machine Research, Computer, 7(6), 34-45.

System Virtualization can be seen an independent abstraction between hardware resources and software applications that enables the division of physical resources into several independent virtual elements. These elements can be assigned "on-the-fly" to applications according to their requirements, thus improving the flexibility in terms of resource management. Among virtualization technologies, the concept of virtual machine is probably best known (Barham et al., 2003). A Virtual Machine (VM) provides a substitute of a physical machine with its own operating system, libraries and applications. A hypervisor acts as the independent abstraction that enables the execution of several VMs simultaneously on a physical machine. Using VMs offers strong advantages from the end-users/developers point of view. First, they can customize a VM with the runtime they want, save it in a persistent manner and start it each time it is needed. Second, by splitting a service into multiple VMs, it is possible to adapt the number of running VMs in order to satisfy the expected quality of service on time. Finally, using VMs provides a stronger isolation between the different tenants that use the distributed infrastructure. Indeed, applications running inside a VM cannot access resources that have been assigned to other VMs.

While Virtualization technologies paved the way of the Cloud Computing revolution, in particular of the computation dimension of the Infrastructure-as-a-Service model (IaaS, *i.e.*, a distributed infrastructure where developers/end-users can provision VMs on demand) (Sotomayor, Rubén S Montero, et al., 2009), it also led to new scientific and technical challenges in terms of resource management (Foster, Y. Zhao, et al., 2008). These challenges are related to the classical management of all resources consumed and provided by the hosting infrastructure (servers, storage, and network) and on the management of the VM lifecycle (creation, suspend/resume and migration operations). Since 2008, my research activities have been focusing on these challenges with three major contributions (i) on the dynamic scheduling of VMs in cloud computing infrastructures (Hermenier, Lebre, and Menaud, 2010; Quesnel and Lebre, 2011; Balouek, Lebre, and Quesnel, 2013; Quesnel, Lebre, Pastor, et al., 2013; Quesnel, Lebre, and Südholt, 2013; Pastor et al., 2014), (ii) on delivering abstractions and accurate models to study VM related challenges through simulations (Hirofuchi and Lebre, 2013; Hirofuchi, Pouilloux, and Lebre, 2013; Hirofuchi, Pouilloux, and Lebre, 2015; NGuyen and Lebre, 2017) and, finally, (iii) on the design of a new generation of Utility Computing that can cope with new requirements of Internet of Things and Mobile Computing applications (Lebre, Anedda, et al., 2011; Bertier et al., 2014; Lebre, Pastor, Simonet, et al., 2017; Confais, Lebre, and Parrein, 2017). It is noteworthy that although these three parts can be considered as independent activities, they have nurtured each other as explained in the following.

1.2 Scientific Contributions

At coarse-grained, the red thread of the activities I conducted is the following one: While investigating the challenge related to the dynamic scheduling of VMs, in particular under the scalability perspective (see Section 1.2.1), I remarked that our community lacked of a simulator toolkit to evaluate and compare state-of-the-art's solutions at large scale (see Section 1.2.2). Addressing these challenges at large scale has been a wise choice with respect to the current trend toward large-scale and widely distributed Cloud Computing Infrastructures (see Section 1.2.3).

1.2.1 Dynamic scheduling of VMs 2008 - 2015

Among challenges IaaS providers soon faced was the VMs proliferation dilemma (*a.k.a.*, the VM sprawl (Sarathy, Narayan, and Mikkilineni, 2010)). Due to the fact that the creation of VMs was becoming straightforward and the possibility to provision and free VMs on demand, administrators as well as developers have built new services/applications on a VM-basis. The main issue of this software architecture choice is that it may lead to over-provisioning situations where some VMs are consuming resources that they do not longer need, preventing other VMs to access resources they require. To mitigate resource waste as well as shortages, the distributed computing community has been investigating new mechanisms in charge of maximizing system utilization while ensuring the Quality of Service (QoS) of each VM.

Among the different solutions available, the ASCOLA research group worked on the Entropy proposal (Hermenier, Lorca, et al., 2009), a framework built on the concept of a central MAPE loop (Monitor-Analyze-Plan-Execute) (Huebscher and McCann, 2008) and constraint programming concepts (Rossi, Van Beek, and Walsh, 2006) to deal with the VMs placement problem. The first contribution we made at my arrival in ASCOLA has been a generalization of the Entropy proposal to manipulate VMs across a cluster in the same manner as processes on a laptop (Hermenier, Lebre, and Menaud, 2010). We proposed the concept of *cluster-wide context switches of virtualized jobs*. By encapsulating jobs into VMs and by using VM operations such as migration, suspend and resume it was possible to resolve under-used and over-loaded situations dynamically by means of autonomous mechanisms that reassign resources in case of service degradation. This fine-grained management of VMs enabled the execution of a maximum number of jobs simultaneously and non-intrusively (VMs were considered as black boxes).

Leveraging this first result, I proposed to investigate how this cluster-scale solution could be extended to handle larger infrastructures composed of thousands of servers potentially spread over multiple-sites. The key idea I proposed has been to leverage P2P mechanisms while keeping the constraint programming approach to address the optimization problem. First, we introduced DVMS (*Distributed VM Scheduler*), a manager that schedules and supervises VMs cooperatively and dynamically in distributed systems (Quesnel and Lebre, 2011; Quesnel, Lebre, and Südholt, 2013; Quesnel, Lebre, Pastor, et al., 2013; Balouek, Lebre, and Quesnel, 2013). DVMS relies on a ring-based algorithm that allows nodes to cooperate for solving under and over-loaded situations: when a node cannot guarantee the QoS for the VMs it is hosting or when it is under-utilized, it starts an iterative scheduling procedure by querying its neighbor to find a better placement. If the request cannot be satisfied by the neighbor, it is forwarded to the following free one until the procedure succeeds. This approach allows each scheduling procedure to send requests only to a minimal number of nodes, thus decreasing the scheduling as well as the reconfiguration time (*a.k.a.*, the aforementioned VM context switch operation) without requiring a central point. The second extension we made focused on the locality of nodes hosting the VMs. The objective was to mitigate inter-site VM relocations that are costly in the context of a cloud infrastructure spread over several sites (Pastor et al., 2014). To tackle this problem, we designed a generic building block that enables the reification of the *locality* aspects at the level of the scheduling algorithm. The locality has been estimated through a cost function of the latency/bandwidth tuple between peers in the network, thus enabling each peer to select its closest neighbors. We rely on Vivaldi (Dabek et al., 2004), a simple decentralized protocol allowing to map a network topology onto a logical space while preserving locality. On top of Vivaldi, a shortest path construction, similar to Dijkstra's well-known algorithm, is performed each time there is a need for cooperation between two nodes that take part in the schedule. We evaluated the advantage of this new building block by changing the overlay network in the DVMS proposal.

It is noteworthy that for each proposal, we implemented an open-source framework and evaluated it through in-vivo experiments, as further discussed in the document.

All pieces of software are available online: http://beyondtheclouds.github.io/DVMS/.

1.2.2 Virtualization and Simulation toolkits 2013 -

While evaluating our VM placement algorithms, I observed there was no instrument to evaluate and compare the different approaches that have been proposed in the literature. Although the VM placement problem has been a hot topic for almost one decade, most proposals were evaluated either using limited scale in-vivo experiments or ad-hoc simulator techniques. Such validation methodologies are unsatisfactory. First they do not model precisely enough real production platforms (size, workload representativeness, etc.). Second, they do not enable the fair comparison of each approach.

SimGrid is a scientific instrument to study the behavior of large-scale distributed systems such as Grids, HPP, P2P and more recently Cloud systems thanks in part to the work I did. It can be used to evaluate heuristics and prototype applications. All this as a free software. Further information available at the Simgrid website.

The second contribution we made has consisted in extending SimGrid, a toolkit for the simulation of potentially complex algorithms executed on large-scale distributed systems (Casanova, Legrand, and Quinson, 2008). Although it has been intensively used in HPC and P2P areas, it did not provide the VM abstractions that were required to study and compare VM placement strategies and more generally VM-related challenges. We proposed SimGrid VM, a highly-scalable and versatile simulation framework supporting VM environments (Hirofuchi, Pouilloux, and Lebre, 2015). SimGrid VM allows users to launch hundreds of thousands of VMs on their simulation programs and control VMs in the same manner as in the real world (e.g., suspend/resume and migrate). Users can execute computation and communication tasks on physical machines and VMs through the same SimGrid API, which provides a seamless migration path to IaaS simulations for hundreds of users. Moreover, SimGrid VM includes a live migration model that implements the precopy migration algorithm. This model correctly calculates the migration time as well as the migration traffic, taking account of resource contention caused by other computations and data exchange within the whole system.

Leveraging these extensions, we developed VMPlaceS (Lebre, Pastor, and Südholt, 2015), a dedicated simulation framework to perform in-depth investigations and fair comparisons of VM placement algorithms. VMPlaceS provides generic programming support for the definition of VM placement algorithms, execution support for their simulation at large scales, as well as new means for their trace-based analysis. The accuracy of VMPlaceS has been validated by comparing simulated and *in-vivo* executions of a centralized scheduling strategy (Hermenier, Lorca, et al., 2009). We have also illustrated the relevance of VMPlaceS by evaluating and comparing algorithms representative of three different classes of virtualization environments: centralized, hierarchical and fully distributed placement algorithms. The corresponding experiments have provided the first systematic results comparing these algorithms in environments including up to one 1K nodes and 10K VMs.

Although the efforts I put in this action are less important now than what I did between 2013 and 2015, I continue to contribute to this area. In addition to being involved in the Simgrid community as much as I can (Lebre, Legrand, et al., 2015), I recently started a study on the boot time of VMs (NGuyen and Lebre, 2017). Most cloud simulators often ignored the VM boot time because they assumed that the duration is negligible. Because the VM booting process consumes resources (CPU, I/O ...), it has been rather simple to disprove through experiments this assumption. Booting a VM can last from tens of seconds up to minutes depending on co-located workloads. We expect to validate a first-class VM boot time model soon and integrate it into SimGrid.

All pieces of software are available online: http://simgrid.gforge.inria.fr and http://beyondtheclouds.github.io/VMPlaceS/.

1.2.3 Beyond the Clouds
2014 -

To satisfy the demand for Cloud Computing resources while realizing economies of scale, the production of computing resources has been concentrated for the last ten years in mega data centers (DCs) where the number of physical resources that one DC can host is limited by the capacity of its energy supply and its cooling system. To meet these critical needs in terms of energy supply and cooling, the trend has been toward building DCs in regions with abundant and affordable electricity supplies or taking advantage of free cooling techniques available in regions close to the polar circle (Gary Cook, 2013). While this model of mega DCs still prevails, the advent of new usages related to Internet of Things applications (IoT) (Atzori, Iera, and Morabito, 2010), Mobile Edge Computing (MEC) (Fernando, Loke, and Rahayu, 2013) and Network Function Virtualization (NFV) (Mijumbi et al., 2016) is strongly challenging this approach. As an example, the projection regarding the number of connected devices and applications that will consume cloud services and generate massive amounts of data is a severe scalability challenge for the current model (B. Zhang et al., 2015). To cope with these changes in terms of usage, the cloud and network communities are now advocating for going from a few central platforms to massively distributed small sized DCs that are deployed at the edge of the network, thus closer to end-users and their related devices, and applications (Bonomi et al., 2012).

the Discovery initiative is an Open-Science Initiative aiming at implementing a fully decentralized IaaS manager. Since 2015, the initiative is mainly supported through the Inria Project Labs program and the I/O labs, a joint lab between Inria and Orange Labs. I am the Principal Investigator of this actions and the current leader. Further information available at the DISCOVERY website.

The drivers of this (r)evolution lay in the development of appropriate software mechanisms that will enable an operator to aggregate, supervise and expose such massively distributed resources on the first hand, and the implementation of new kinds of services deployed and managed by the operator itself or by third-party users on the other hand. I have been investigating this challenge for the last 4 years through preliminary studies. Built on these initial investigations I proposed the creation of the DISCOVERY Inria Project Lab that I have led since 2015. DISCOVERY gathers academic and industry experts in research areas such as large-scale infrastructure management systems, network and P2P algorithms.

From the scientific and technical point of view, the novelty of DISCOVERY lies in a new way of designing a system in charge of operating a massively distributed cloud. The system we envisioned can be viewed as a distributed operating system, manipulating Virtual Machines in a geo-distributed infrastructure. The major difference with respect to the state of the art is related to our bottom-up approach: instead of designing a system in charge of orchestrating independent cloud instances, we target to revise and extend mechanisms of the OpenStack software suite with P2P and self-* techniques. Although this activity is rather young in comparison to the two previous ones, it is important for me to deal with it in this manuscript. First, we obtained a few interesting results in particular around the OpenStack community that deserve to be highlighted (Lebre, Pastor, Simonet, et al., 2017; Cherrueau et al., 2017). Second, this activity paves the way for several activities I plan to do for the next couple of years. In particular, this action should lead to set-up a new research group in Nantes that will focus on the design of a complete software stack to operate and use next generation of Utility Computing infrastructures: The Fog, the Edge and beyond!

1.3 Looking Back to the Future

At the end of my post-doctorate position, I led a preliminary work that investigated how suspend/resume VM operations can benefit the best effort mode of the Grid'5000 infrastructure (Gallard, Lebre, and Morin, 2010). Grid'5000 is a dedicated testbed

for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including Cloud, HPC and Big Data (Bolze et al., 2006). In Grid'5000, users book resources leveraging a batch-scheduler approach. Our proposal consisted in encapsulating jobs into VMs so that it became possible to suspend/resume them each time it was needed. Although this study did not have a strong impact on the scientific community because the innovative part with respect to the state-of-the art was not important enough (Sotomayor, Keahey, and Foster, 2008), it completely changed the perception I had regarding how a distributed computing infrastructure should be operated and used (Gallard, Lebre, Morin, et al., 2012; Lebre, 2010). From my point of view, the notion of sandboxes provided by the VM abstraction would radically change the distributed computing perspective and it was urgent to extend several mechanisms of Grid'5000 in order to be able to investigate virtualization challenges.

Leveraging the aforementioned study and achievements made by the distributed systems community, I have been underlining the importance of VM technologies in distributed computing infrastructures. As early as 2008, I proposed to create a dedicated working group at the French national level. The efforts of this group have been then consolidated through several actions such as the Hemera Inria Large-Scale initiative[1] (2010-2014) whose virtualization activities I led. We added the virtualization capabilities to Grid'5000, enabling researchers and engineers to investigate virtualization related challenges (Balouek, Amarie, et al., 2012).

Created in 2006, Grid'5000 is a large-scale and versatile testbed for experiment-driven research. With more than 500 active users per year since 2006, it is one of the most recognised testbeds worlwide. I have been involved in different strategic committees since 2008. For Further information available at the Grid'5000 website.

Along with these activities, I organized several events such as the French events "Journée Thèmes Emergents" on Virtualization in Distributed Architectures organized in 2010 and 2014 in Nantes.[2] At the international level, I had the opportunity to co-chair the ACM Virtualization Technologies in Distributed Computing workshop (VTDC) between 2011 and 2015.[3] Co-located with the ACM HPDC conference, VTDC workshop series has been an important forum for the exchange of ideas and experiences on the use of virtualization technologies in distributed computing. In addition to VTDC, I also took part in several technical program committees such as ACM SuperComputing, IEEE/ACM CCGrid, Europar, IEEE CloudCom and IEEE IC2E to name a few.

Being strongly involved in the community has been a decisive point to strengthen my expertise and to consolidate the vision I have regarding how next generations of utility computing infrastructures should be designed. In particular, I have been among the first researchers to promote and work on the concept of massively distributed cloud infrastructures deployed on network point of presences (Bertier et al., 2014). This has been a wise choice as this new paradigm referred to as the Fog/Edge Computing nowadays, is attracting growing interest by the network as well as the distributed computing communities. In addition to taking part in several events that dealt with this topic (program and track chairs of recent events such as ICFEC, CloudCom and Europar to name a few), I have been invited to co-chair the transversal action "Virtualization and Clouds" of the RSD GDR[4] since 2015. This action aims at further bridging the gap between the communication networks and distributed systems experts of the RSD GDR (as example, we gathered 80 people in Paris for the 1st edition of the CloudDay events). Although the use of virtualization technologies has been leading to exciting developments for almost ten years now, current solutions for operating virtual infrastructures still need to provide a tighter integration between network, computation and storage resources at a geographically wide area network scope. This integration is mandatory to build the next generation of Internet backbones as well as Utility Computing infrastructures. This is urgent as the new requirements of IoT and mobile computing applications cannot be satisfied by cur-

1 https://www.grid5000.fr/mediawiki/index.php/Hemera.
2 http://web.emn.fr/x-info/ascola/doku.php?id=internet:jte-virtualization-2010 and http://people.rennes.inria.fr/Adrien.Lebre/PUBLIC/CloudDay/.
3 http://people.rennes.inria.fr/Adrien.Lebre/VTDC/vtdc15.html.
4 https://rsd-cloud.lip6.fr.

rent cloud computing offers. Finally, I would like to underline that I'm still deeply involved in the Grid'5000 consortium taking part in the architecture and executive committees where we are discussing how Grid'5000 should evolve with respect to recent challenges related to the Industrial Internet and the Internet of Skills. These two internets can be seen as a convergence between data centers, network and IoT devices.

1.4 Organization of the Document

The major part of this manuscript is devoted to the presentation of the three scientific activities that have been highlighted in this introduction (one part for each of them). I underline that all these works have been published and are here gathered, standardized, and revised to make the manuscript more consistent. The last chapter presents an overview of my ongoing activities and my research program for the coming years.

In detail, the documents is structured in 11 chapters. Chapter 2 presents the VM cluster-wide context-switch concept. Chapter 3 discusses the extensions that have been proposed around the DVMS proposal. Chapter 4 concludes the first part. Chapter 5 describes the extension we proposed in the SimGrid toolkit, including the elementary VM abstractions and the live migration model. Chapter 6 gives an overview of the VMPlaceS framework that has been built on top of the SimGrid extensions. Chapter 7 concludes this second part and also discusses preliminary results about an on-going work on the VM boot operation. Chapter 8 presents the DISCOVERY initiative. Chapter 9 and Chapter 10 present the first results we obtained within the framework of DISCOVERY and the OpenStack community. Finally, Chapter 11 concludes this manuscript by summarizing ongoing activities and giving a few perspectives on my future works.

Note that the appendix, attached at the end of this manuscript, includes a detailed Curriculum Vitae, the complete list of my publications and and invited talks.

Part II

DYNAMIC VIRTUAL MACHINE SCHEDULING

This part presents a compilation of the following publications: (Hermenier, Lebre, and Menaud, 2010) and (Quesnel, Lebre, and Südholt, 2013; Quesnel, Lebre, Pastor, et al., 2013) (Pastor et al., 2014).

These results have been possible thanks to collaboration with:

- Jérome Gallard, Phd in the MYRIADS Resarch Group (2007-2010), Rennes, now Research Engineer à Orange Labs ;
- Fabien Hermenier, Phd in the ASCOLA Research Group (2006-2009), Nantes, now Ass. Professor at University of Nice ;
- Jean Marc Menaud, Professor in the ASCOLA Research Group, IMT Atlantique Nantes ;
- Flavien Quesnel, Phd in the Ascola Research Team (0ct 2010- Feb 2013), Nantes, now Research Engineer at System-X Institute ;
- Mario Südholt, Professor and Head of the ASCOLA Research Group, IMT Atlantique, Nantes ;
- Daniel Balouek-Thomert, Research Engineer, Hemera Inria Large Scale Initiative, (Dec 2011 - May 2012), Lyon, now PostDoc at Rutgers University ;
- Jonathan Pastor, Phd in the Ascola Resarch Team (Oct 2012 - Oct 2016), Nantes, now PostDoc at Chicago University ;
- Frédéric Desprez, Senior researcher and Deputy Scientific Director at Inria, Grenoble ;
- Cédric Tedeschi, Ass. Profesor in the MYRIADS Research Group, University of Rennes 1, Rennes ;
- Marin Bertier, Ass. Professor in the ASAP Research Group, INSA, Rennes.

The activities have been mainly supported by the Ecole des Mines de Nantes (now IMT Atlantique), which succesively granted the Phd scholarships of Fabien Hermenier, Flavien Quesnel and Jonathan Pastor. I underline that Dr. Fabien Hermenier has been supervised by my colleague Prof. Jean-Marc Menaud. In the following, I co-supervised, first, the Phd of Dr. Flavien Quesnel with Prof. Mario Südholt and, second, the Phd of Dr. Jonathan Pastor with Dr. Frédéric Desprez.

Major Results as well as software code are available at: `http://beyondtheclouds.github.io/DVMS/`.

VM Cluster-Wide Context Switch

2

Distributed infrastructures such as clusters are mostly handled by resources management systems with a static allocation of resources for a bounded amount of time. Those approaches are insufficient for an efficient use of resources. To provide a finer management, job preemption, migration and dynamic allocation of resources are required. However due to the complexity of developing and using such mechanisms, advanced scheduling strategies have rarely been deployed in production. This trend has been evolving since 2008 thanks to the use of migration and preemption capabilities of VMs. However, even though the manipulation of VMs-based jobs enables the reconfiguration of the system according to the scheduling objective, changing the state and the location of numerous VMs at each decision is tedious and degrades the overall performance. In addition to the scheduling policy implementation, developers have to focus on the feasibility of the actions (such as suspend, resume and migrate) while executing them in the most efficient way.

In this section, we present the cluster-wide context switch*, a building block that leverage virtualization capabilities to facilitate the development of advanced scheduling strategies (Hermenier, Lebre, and Menaud, 2010). Our proposal relies on the manipulation of virtualized-jobs (vjobs), i.e. jobs composed of VMs, through their life cycle. As a proof of concept, we have integrated the cluster-wide context switch module into the Entropy consolidation manager (Hermenier, Lorca, et al., 2009). Developers of scheduling policies can now only focus on the algorithm to select the vjobs to run. Then the cluster-wide context switch infers and plans the actions to perform on the VMs. It ensures the feasibility of the process and selects, among the multiple satisfying solutions, the one implying the fastest change. We evaluated the cluster-wide context switch through a FCFS dynamic scheduler. Transparently for developers, it uses dynamic allocation of resources, preemption and migration technics to maximize the resources usage.*

The chapter concludes by highlighting three research axes raised by this work.

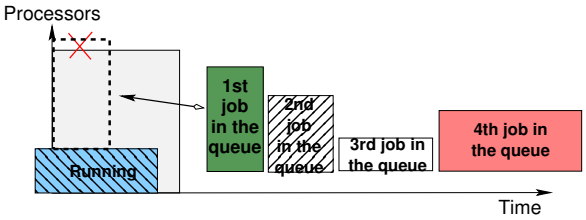
2.1 Challenge Description

Scheduling jobs has been and still is a major concern in distributed computer systems. According to their size and their objectives, computing infrastructures are exploited in different ways. However, few of them are dedicated to one particular application and the most common way of exploiting large common computing infrastructures has mainly consisted in using Resources Management Systems (RMS) (Krauter, Buyya, and Maheswaran, 2002) where users request resources for a specific duration according to their estimated needs. Such an allocation of resources lead to a coarse-grain exploitation of infrastructures. In the best case, the allocated time slot is larger than the estimate and resources are underused. In the worst case, running applications can be withdrawn from their resources which may lead to the loss of all the performed calculations. Although strategies such as back-filling approaches have been proposed to improve the resources usage, preemption mechanisms such as the ones used for context switching in traditional operating systems are valuable to deliver an efficient management of resources. Thanks to preemption mechanisms, jobs can be processed, even partially, and suspended according to the scheduler objectives (Feitelson et al., 1997): Whenever necessary, the RMS can suspend some jobs and start, or resume from previously saved images, other ones. Such an allocation scheme is presented in Figure 2. Unfortunately due to the problem of residual dependencies (Clark

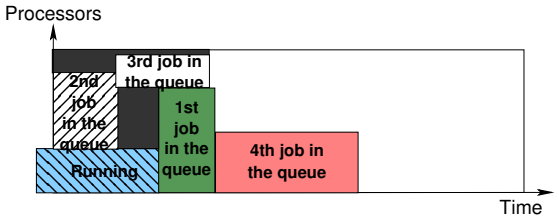
et al., 2005), the development of such mechanisms is tedious and only few RMS allow such a level of scheduling (Etsion and Tsafrir, 2005).

Figure 2: *Toward a Dynamic Scheduling Strategy*

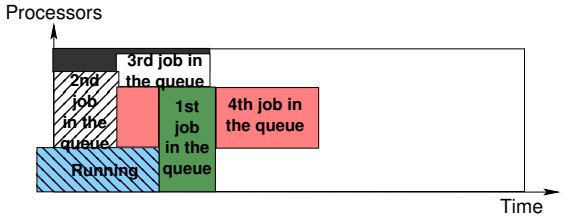
Figure 2a depicts a minimalist scheduling process: jobs are enqueued one after the other and are scheduled according to their estimated duration and resource requirements. Figure 2b illustrates the backfilling mechanism improvement, which deals with fragmentation issues while guaranteeing a time reservation for the first job in the queue. Although there exist more advanced backfilling strategies, such as Conservative which provides time guarantee for each waiting job in the queue, these strategies suffer from limitations that prevent an optimal usage of the resources without exploiting advanced mechanisms such as preemption. Figure 2c presents such an approach that enables the execution of a job, even partially, each time it is possible.



(a) *Jobs 1, 2, 3 and 4 arrive in the queue and have to be scheduled*



(b) *FCFS + EASY back filling: jobs 2 and 3 have been back filled. Some resources are unused (dark gray areas).*



(c) *EASY back filling with preemption. The job 4 is started earlier without impacting job 1. A small piece of resources is still unused (dark gray areas). Thanks to the consolidation and preemption mechanisms, it is possible to improve the whole cluster usage.*

The increasing popularity of virtualization technologies in 2008 led to several initiatives (Fallenbeck et al., 2006; Sotomayor, Keahey, and Foster, 2008) that investigated the use of VM migration and preemption capabilities to improve the resource usages of distributed infrastructures: live migration (Clark et al., 2005) aims to adapt the assignments of VMs according to their current requirements (Bobroff, Kochut, and Beaty, 2007; Hermenier, Lorca, et al., 2009; Ruth et al., 2006; Verma, Ahuja, and Neogi, 2008), while the suspend and the resume actions provide preemption (Anedda et al., 2010).

However manipulating VMs needs to be performed carefully. First, suspending or resuming VMs can impact the correctness of a job, especially if the VMs are interconnected. Second, migrating a VM from one node to another one can be impossible without violating VM expectations in terms of resource or placement constraints (*e.g.*, migrating a "replica" VM to the same host that the master instance should not be allowed) (Grit, D. Irwin, Yumerefendi, et al., 2006; Hermenier, Lorca, et al., 2009; Wood et al., 2009). Finally, each context switch (or permutation) of VMs should be performed in an efficient manner. Migrating a VM can last from a few seconds up to minutes according to the executed workload while suspending/resuming VMs are I/O intensive operations that can negatively impact the performance of the whole infrastructure.

Although considering permutation issues is fundamental, most developers of schedulers did not consider it as a primary concern. In most cases, they simply focused on the efficiency of the algorithms to select the jobs to run while ignoring the mechanisms to perform the transition between the current situation and their solution.

The remainder of the chapter is organized as follows: Section 2.2 describes the VM context switch proposal overall. It first introduces fundamentals and second gives an overview of the framework and how a VM context switch is performed clusterWide.

Section 2.3 discusses an evaluation of a sample scheduler that enabled us to validate our concept. Section 2.4 addresses related work. Finally Section 2.5 concludes this chapter and gives some perspectives.

2.2 Our Proposal: A Generic VMs Context Switch

Back to 2009, the first contribution we did has been to deliver the *VM cluster-wide context switch* operation as a fundamental building block (Hermenier, Lebre, and Menaud, 2010). We showed that the permuation operation between two allocations of VMs is independent from the policy itself and can be addressed through a generic mechanism. Thanks to it, developers can implement sophisticated algorithms to schedule "virtualized" jobs without handling the issues related to their manipulations.

2.2.1 Fundamentals

This section deals with the fundaments of our proposal. First, we define the *virtualized job* abstraction and the different actions that change its state. Second, we present preliminary experiments on the cost of each action.

Virtualized job

The first change we did has been to reconsider the batch scheduler granularity from the usual job to the virtualized one. We defined the *virtualized job* (*vjob*) abstraction. According to the nature of the application to execute (centralized or distributed), a *vjob* can be composed of several VMs. Implementing a dynamic scheduling policy consists in manipulating vjobs through their four different states described in Figure 3: `waiting`, `sleeping`, `running` or `terminated`. The transition between the different states consists in applying the associated action to all the VMs composing the vjob. Actions are supposed to be atomic: all the VMs composing a vjob are in the same state. The action *migrate* slightly differs from others. It relocates a VM from one node to another with a negligible downtime using the live migration capability (Clark et al., 2005). This action can be performed on subsets of vjobs as it does not change the state of the manipulated VMs.

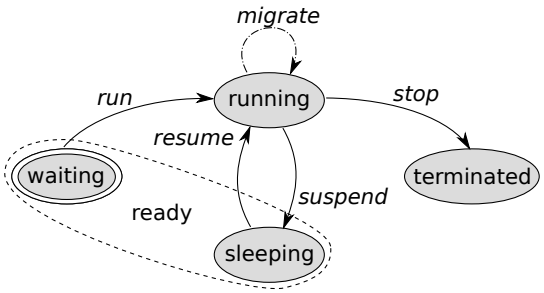


Figure 3: *The Life Cycle of a Virtualized Job.*

Each transition but migrate implies to execute the associated action on each VM attached to the vjob

In detail, the action *run* starts the selected vjob and puts it into the state `running`. It consists in launching the VMs of the vjob on the cluster. A vjob can be suspended on a persistent device to free resources with the action *suspend*. This lets the vjob into the state `sleeping`. Reciprocally, a suspended vjob can be resumed with the action *resume*: each VM is resumed from its state file previously saved. Finally, the action *stop* shuts down a vjob and puts it into the state `terminated`. The pseudo state `ready` combines the states `waiting` and `sleeping`. Both precede the state `running` and if the actions to perform the transition are different, these states are equivalent regarding to the availability of the vjob and its resources consumption.

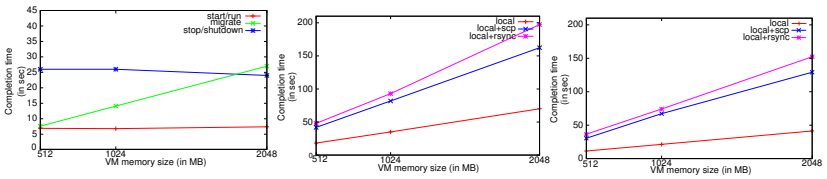
Relying on this model, implementing a dynamic cluster scheduler consists of the following steps, (i) determine the set of vjobs to run, (ii) associate a sufficient amount

of resource to each VM attached to the selected vjobs, and (iii) execute the mandatory actions to perform the vjobs transition.

The main contribution of our proposal has been to provide a generic system to deal with the two latest steps.

VM Elementary Operation Costs

Evaluating the cost of a cluster-wide context switch is mandatory since it significantly reduces the performance on the nodes involved in the action. Indeed migrating, suspending or resuming a VM requires some CPU resources, memory and network bandwidth. When the involved nodes host CPU intensive VMs, performing the action reduces their access to these resources for the whole duration of the action. In order to have an idea of the cost of each operation, in particular the completion time, we conducted a first series of experiments. Evaluations have been done on 11 homogeneous nodes from Grid'5000 including a 2.1 GHz Intel Core 2 Duo, 4 GB of RAM and interconnected through a giga ethernet network. Each node runs a GNU/Linux 2.6.26-amd64 with Xen 3.2 and 512 MB of RAM are allocated to the Domain-0. Three NFS storage servers provide the virtual disks for all the VMs.



(a) *Run/Migrate/Stop* (b) *Suspend* (c) *Resume*

Figure 4: *Completion time of VM Operations according to the amount of memory allocated to the VM*

Figure 4 shows the average duration of each action depending on the amount of memory allocated to the manipulated VM. As expected, the duration of a run or a stop action is independent from the amount of memory allocated to the VM: booting a VM takes around 6 seconds in our architecture whereas a clean shutdown takes approximately 25 seconds. This second duration is due to the different service timeouts and can be easily reduced to a second by using a "hard" shutdown of the VM. On the opposite, the duration of a migration on a VM clearly depends on the amount of memory allocated to it.

This series of experiments is one of the first link I want to highlight with the second major contribution I present in this manuscript. As discussed later on in Chapter 5, I conducted deeper investigations to understand the parameters that govern first the live migration process and second the VM boot operation. Here, I want to focus the reader's attention on the cost of manipulating VMs dynamically, especially on the start and migration time because we will see they can be really significant.

Moreover the duration of a suspend or a resume action depends on the involved nodes. We conducted several benchmarks to evaluate the cost implied by a local or a remote suspend/resume operation (*i.e.*, the suspend is done locally and the command `scp` or `rsync` pushes the file on the destination node and reciprocally for the resume). The difference between a local and a remote resume/suspend is quite significant (twice the duration). These results show the interests of preferring local suspend and resume actions instead of the remote ones to reduce the duration of a cluster-wide context switch.

2.2.2 Architecture Overview

As a generalization of one previous contribution of the ASCOLA Research group, the cluster-wide context switch has been integrated as an independend module of the Entropy consolidation manager (Hermenier, Lorca, et al., 2009). Entropy focused on hosting all the running VMs on the minimum number of nodes using migrations. Since it did not consider the whole life cycle of the VMs, it was not able to solve overloaded situations that require to suspend VMs. In addition, it manipulated each VM individually and was not able to ensure a consistent state for a set of interconnected VMs. After describing the changes we made to Entropy, we present the implementation of the cluster-wide context switch.

Global Design

We consider a cluster as a set of working nodes that can host VMs, a set of storage nodes to serve the virtual disks of the VMs and a set of service nodes that host services such as the head of the distributed monitoring system and the Entropy service. A *configuration* describes the CPU and the memory capacity of each node and the CPU, the memory requirement and the position of each VM in the cluster. A viable configuration denotes a configuration where each running VM has an access to a sufficient amount of memory and CPU resources to satisfy its requirements. Our modifications on Entropy make it able to implement the cluster-wide context switch.

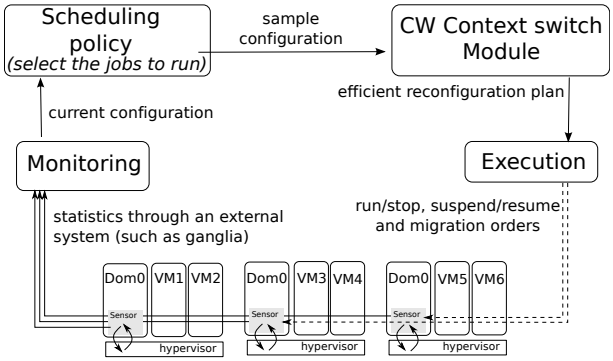


Figure 5: *The control loop of Entropy*

The system acts as a loop (see Figure 5) that (i) extracts the current configuration from a monitoring service, (ii) executes the scheduling strategy defined by administrators that indicates the state of the vjobs for the next iteration, (iii) determines from the current and the new configuration the actions to perform and plans them and (iiii) executes the cluster-wide context switch by performing the actions.

Implementation

Performing a cluster-wide context switch requires to execute several actions on VMs in a correct order and an efficient way. In this section, we first address the feasibility of these actions. Second we describe our solution to ensure the correctness of suspending and resuming a set of interconnected VMs. Finally, we present our approach to reduce the duration of its execution.

Plan the actions on VMs

Regarding to the life cycle of the vjobs, their current state and their state computed by the scheduling policy for the next round, we determine the actions to apply on VMs. These actions must be planned to ensure their feasibility. Some actions have to be executed before others (Grit, D. Irwin, Yumerefendi, et al., 2006; Wood et al., 2009) and additional migrations have to be considered to solve inter-dependency issues (Hermenier, Lorca, et al., 2009). Solving these dependencies consists in providing a *reconfiguration plan* that describes an execution protocol ensuring the feasibility of the actions. A reconfiguration plan is modeled as a sequence of *steps*, *i.e.*, a set of actions that are executed sequentially, while the actions composing them are executed in parallel. To reduce the duration of the cluster-wide context switch and to increase reactivity, it is mandatory to perform in parallel as many actions as possible so that each action takes place with the minimum possible delay.

The reconfiguration plan is created iteratively from a reconfiguration graph that describes the actions to perform to pass from one configuration to another. A reconfiguration graph is an oriented multigraph where each node denotes a machine and each edge denotes an action on a VM between two machines. First, all the feasible actions are grouped into a step. If there are no feasible actions, it is due to an inter-dependent issue. In this situation, the cycle is broken with an additional migration

on a temporary node to create at least one feasible action, added to the current step. Then the step is appended to the plan, and a new reconfiguration graph is created using the temporary resulting configuration of the plan and the expected configuration. This process is repeated until the resulting configuration of the plan equals the expected configuration.

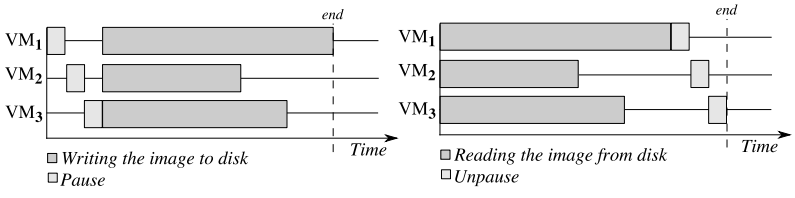
Suspending and resuming a vjob

When several VMs execute a distributed application, they exchange network packets through a transport protocol, commonly TCP. This protocol maintains a total ordering between IP packets and ensures their delivery. When a host does not receive an acknowledgment for a packet after multiple retransmissions or when the *keep-alive* timeout is reached, the protocol considers the receiver as unreachable and closes the connection.

During the cluster-wide context-switch, the states of VMs belonging to the same vjob may not be consistent for a moment since actions are based on a VM granularity and some VMs may not be reachable during this time. Thus it is necessary to coordinate the suspend and the resume actions when the cluster-wide context switch manipulates the state of several VMs belonging to the same vjob. This coordination ensures that the distributed application will not fail during the reconfiguration.

Suspending a VM consists in pausing it then writing the image of its memory to a disk. When the VM is paused, it does not execute any instructions. On the opposite, resuming a VM consists in restoring its memory by reading its image from the disk then unpausing it. While executing a pause (or an unpause) takes only a few milliseconds, reading (or writing) the image from (to) the disk takes a significant amount of times, led by the amount of memory allocated to the VM (see Section 2.2.1).

Once one of the VMs is paused while others are running, the state of the vjob is not consistent. To avoid network errors, the VMs will have to be in the same state (sleeping or running) before outreaching the *unreachable* delay of TCP. A solution to efficiently coordinate several suspend and resume actions without any clock synchronization is to pause each VM sequentially, in a deterministic order using their unique identifier. To coordinate the suspend actions, all the VMs are first paused sequentially then the content of their memory is written in parallel to the disk (Figure 6(a)). To coordinate several resumes, the images are read in parallel then all the VMs are unpaused sequentially in the same order they were suspended earlier (Figure 6 (b)). As the pause and the unpause actions take only a few milliseconds, this solution ensures that the delay between the first and the last pause (or unpause) is inferior to the *unreachability* delay while non-acknowledged packets will be retransmit.



(a) *Suspend* (b) *Resume*

Figure 6: *Several suspend and resume actions are coordinated by sequencing pause and unpause actions*

Our coordination protocol can be integrated into Entropy by altering the reconfiguration plan. It consists in grouping the resume and the suspend actions into the same step. The suspend actions do not have precondition so they are naturally grouped into the first step as they are always feasible. The conception of a plan ensures that if an action is feasible at a specific step, it is feasible for all the following steps. Thus, to satisfy the dependencies of the resume actions, each action is grouped into the step that initially contains the last resume action. Figure 7 shows an unmodified reconfiguration plan, it suspends a vjob j_1 composed of the VMs vm_1 and vm_2, resumes a vjob j_2 composed of the VMs vm_4 and vm_5, the VMs vm_3 and vm_6 belong to other

vjobs that stay in the running state. Figure 8 shows the modified version of this plan with the coordination of the actions related to vjobs j_1 and j_2.

$$
\begin{aligned}
s_1: &\quad suspend(vm_1) \,\&\, suspend(vm_2)\\
s_2: &\quad resume(vm_4) \,\&\, migrate(vm_3)\\
s_3: &\quad resume(vm_5) \,\&\, migrate(vm_6)
\end{aligned}
$$

Figure 7: *Unmodified reconfiguration plan with three steps s_1, s_2, and s_3. '&' indicated the parallelism.*

$$
\begin{aligned}
s_1: &\quad pause(vm_1)\\
s_2: &\quad pause(vm_2)\\
s_3: &\quad write(vm_1) \,\&\, write(vm_2)\\
s_4: &\quad migrate(vm_3)\\
s_5: &\quad migrate(vm_6) \,\&\, read(vm_4) \,\&\, read(vm_5)\\
s_6: &\quad unpause(vm_4)\\
s_7: &\quad unpause(vm_5)
\end{aligned}
$$

Figure 8: *Reconfiguration plan to coordinate the suspend and resume actions of the vjob j_1 and j_2.*

Reducing the duration of the execution

To reduce the duration of a cluster-wide context switch, Entropy computes several similar configurations where all the vjobs of each configuration have a state identical to the sample configuration provided by the scheduling algorithm. Indeed, according to the scheduler policy, the location of the VMs is not relevant, only the state of each vjob is significant. However from a cluster-wide context switch point of view, each transition has a duration. As a consequence, we should be able to compare reconfiguration plans with a cost function and select the one with the smallest estimated duration.

The cost of a plan is a positive value. Its has no unit but it allows to compare different plans when they reach equivalent configurations. The cost of a reconfiguration plan P composed of a sequence of n steps s and x actions a is defined through an objective function K described in the Equation (1). This model assumes the cost of an action is minimal when it is executed in the first step. Thus, delaying an action increases the cost of a plan.

$$
\begin{aligned}
K(p) &= \sum_{x=0}^{m} K(a_x)\\
a_x \in s_i, K(a_x) &= \sum_{j=0}^{i-1} K(s_j) + k(a_x)\\
K(s_i) &= max\left(k(a_x)\right), a_x \in s_i
\end{aligned}
\tag{1}
$$

The local cost k of an action a_x relies on the experiments described in Section 2.2.1. The different values for $k(a_x)$ are listed in Table 1. We showed that the duration of both the stop and the run actions is constant and depends on the software running in the involved VM. For this study, their cost has been set to 0. The duration of a suspend and a migrate actions is led by the memory requirement of the involved VM. We set this cost to the amount of memory allocated to it. Finally, the duration of the resume action depends on the memory demand of the VM and its destination location. Indeed, a *local* resume does not require to move the image file to the destination node. We choose to define the cost of a remote resume as twice the cost of a local one. By such a way, we favor local resume.

To compute the different viable configurations, Entropy uses the Choco solver (Jussien, Rochart, and Lorca, 2008) which is based on a Constraint Programming (Rossi, Van Beek, and Walsh, 2006) approach.

Table 1: *Local cost of an action a_i. $\mathcal{D}_m(vm_j)$ denotes the memory allocated to the VM vm_j*

Type of a_x	$k(a_x)$
migrate	$\mathcal{D}_m(vm_i)$
run	*constant*
stop	*constant*
suspend	$\mathcal{D}_m(vm_j)$
resume	$\mathcal{D}_m(vm_j)$ if *local*, $2 \times \mathcal{D}_m(vm_j)$ otherwise

2.3 Proof-of-Concept & Validations

As a proof of concept, we implemented a sample scheduler that only indicates at each round the states of the vjobs. According to the changes, the cluster-wide context switch system handles the transition and provides transparent dynamic resources allocation, live migration and vjob preemption. We discuss the execution of this algorithm relying on the cluster-wide context switch on a cluster composed of 11 working nodes running NAS Grid Benchmarks (Frumkin and Van der Wijngaart, 2002).

2.3.1 Proof-of-Concept

A common objective of schedulers is to execute jobs as soon as possible. The use of preemption, migration and dynamic allocation of resources improves the cluster usage by executing jobs, even temporarily, when there is a sufficient amount of resource to satisfy their requirements. The scheduling algorithm we implemented tends to satisfy those principles.

The algorithm computes a list of vjobs, initially empty, that specifies the vjobs that must be running for the next iteration. To avoid starvation, all the vjobs are stored in a queue with a FCFS priority. For each vjob in the queue, we check if it exists a viable configuration composed of the current vjob and the vjobs already in the list. This assumption is made using the First Fit Decrease algorithm that assigns each VM composing the vjob to the first node with a sufficient amount of free resources. If the algorithm succeeds in assigning all the VMs then the vjob is added to the list. After the last iteration over the queue, the list of vjobs that will be in the `running` state is defined, the other vjobs will be in the `ready` state.

2.3.2 Experiments

The experiment consists in scheduling 8 vjobs, each composed of 9 VMs, using our sample scheduling algorithm. Although vjobs are submitted at the same moment, they are enqueued in a deterministic order to ensure the reproducibility of the experiments. Each vjob is running an application composed of NASGrid Tasks. The application embedded in the vjob is launched when all the VMs are in the `running` state. When the application is terminated, it signals to Entropy to stop its vjob. Each VM requires a fixed amount of memory, from 512 MB to 2048 MB and requires an entire CPU when the NASGrid task is executing a computation on the VM. The monitoring sensor running on each working node takes at most 10 seconds to signal a change to the monitoring frontend (*i.e.*, a Ganglia monitoring system). Obviously, this reactivity has a negative impact on the ability of Entropy to quickly fix a non-viable configuration and a faster monitoring system would have improved this situation.

Computing the reconfiguration plan with the minimum cost may be time consuming. Choco has the property that it can be aborted at any time, in which case it returns the best result computed so far. This feature enabled us to impose a time limit on the solver, to ensure the reactivity of the approach. In previous experiments (Hermenier,

Lorca, et al., 2009), Entropy computes reconfiguration plans with 200 nodes and 400 VMs in one minute. For the current experiment, the total computation time is limited to 30 seconds and Choco computes each plan in less than 20 seconds.

Figure 9 shows the cost and the duration of the cluster-wide context switches performed during the experiment. Those with a small cost and duration only perform migrations, run or stop action. As an example, the five with a cost equal to 0 only perform run and stop actions and take at most 13 seconds. The cluster-wide context switch with a cost equal to 1024 performs 3 migrations in 19 seconds. Cluster-wide context switches with a higher cost perform in addition suspend and resume actions. This increases significantly the duration of their execution. As an example, the one with a cost equal to 4608 takes 5 min 15 seconds to execute 9 stop actions, 18 run actions, 9 resume actions and 9 migrations. In addition, the cost of the plan is not related to its duration when it implies different kind of actions. As mentioned in Section 2.2.2, the cost function only enables to determine the best reconfiguration plan when all the VMs in the destination configuration have the same state. At this moment, our cost function cannot be used to estimate the duration of one cluster-wide context switch. However it appears to be a viable solution to avoid to migrate VMs or perform remote resumes if possible: 21 over the 28 resume actions performed during the experiment were made on the nodes that perform the suspend earlier.

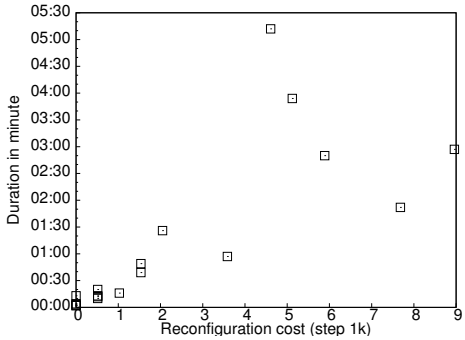


Figure 9: *Cost and duration of the 19 cluster-wide context switches performed during the experiment*

The second part of this evaluation analyses the benefit of the cluster-wide context switch with regards to to a static approach. In theory, the gain provided by dynamic scheduling policies is related to the resources requirements of the vjobs. In the best case, requirements evolve during the execution of the vjob, and the benefits are significant. In the worst case, requirements are constants and a static approach with backfilling is sufficient. As in many cases, the NASGrid benchmarks used in this experiment do not require an entire CPU all the time.

In this experiment, the static scheduling policy relies on a simulated *First Come, First Serve* (FCFS) algorithm. The FCFS scheduler selects the vjobs to execute by iterating over a queue. When there is a sufficient amount of free resources to execute all the VMs composing a vjob, the scheduler allocates a CPU and a sufficient amount of memory to each VM for the whole duration of the vjob. Each vjob is composed of 9 VMs, thus requiring 9 CPUs to run. As each vjob requires the same amount of CPU to run, a backfilling strategy would not improve scheduling. Figure 10 shows the execution diagram of the vjobs.

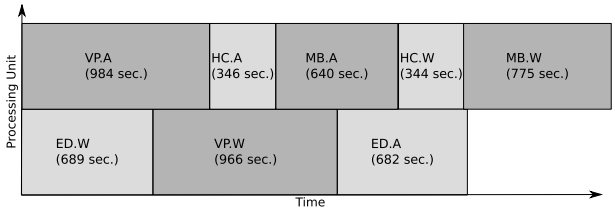
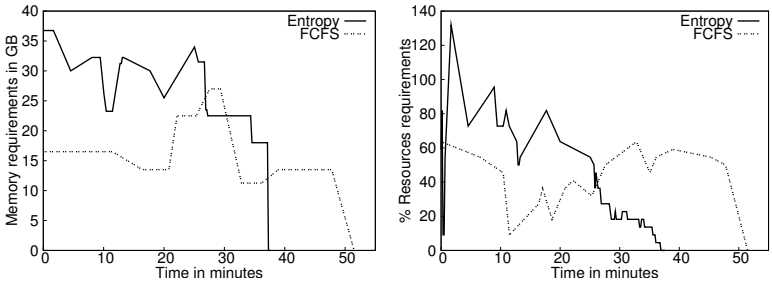


Figure 10: *Execution diagram for the vjobs with a FCFS Scheduler*

Figure 11a and 11b show the resources usage of the VMs with the two different schedulers. The average resources usage is more important with our module dur-



(a) *Memory requirements* (b) *CPU requirements*

Figure 11: *Resources utilization of the VMs*

ing the first 30 minutes. Afterwards, resources utilization with Entropy decreases as there are no more vjobs to run. At 2 minutes 10, the cluster is overloaded as the running vjobs demand 29 processing units while only 22 are available. In this situation, the scheduler indicates which of all the vjobs must be running to have a viable configuration and the cluster-wide context switch performs the transition by suspending the vjobs that must be in the `ready` state. This improvement of the resource usage reduces the cumulated completion time of the vjobs. The cumulated completion time for the 9 vjobs is equals to 250 minutes with the FCFS scheduler. It is reduced to 150 minutes with our sample scheduling policy.

To conclude, one can implement finer scheduling strategies by focusing on the scheduling policy and leaving the issues related to the vjob permutation to the cluster-wide context switch module.

2.4 Related Work

Back to 2010, Sotomayor *et al.* (Sotomayor, Keahey, and Foster, 2008; Sotomayor, Rubén Santiago Montero, et al., 2008) provided with Haizea the concept of lease as an abstraction for resource provisioning. Users negotiate an amount of resource for a specific duration by indicating (i) wether the lease is made in a best-effort mode or uses advanced reservations and (ii) if it is preemptible. Depending on its type, the lease may be migrated, or suspended to free resources for non-preemptible leases or leases with advanced reservations. This approach enables users to renew a period of execution for a new amount of time but does not provide a way to dynamically change the set of resources assigned to a lease with variable needs.

Grit *et al.* (D. E. Irwin et al., 2006; Grit, D. Irwin, Yumerefendi, et al., 2006; Grit, D. Irwin, Marupadi, et al., 2007) described in Shirako the necessity of separating the management policy of the VMs and the mechanisms to perform the changes as we argue in the present work. They considered the sequencing issues when migrating several VMs for resource management policies, However to solve inter-dependencies, they chose to suspend a VM to break the cycle of dependencies instead of using a bypass migration. Regarding to our experimental study, suspending a VM is much more time consuming than the use of a bypass migration. Moreover, they only considered VM migration in their VM manager. By ignoring the possibilities to manipulate the state of the VMs, it is not possible to implement advanced scheduling policies.

Fallenbeck *et al.* (Fallenbeck et al., 2006) provided an environment to dynamically adapt the number of slots available for specific scheduling policies with multiple queues. A VM is available on each working node for each queue. Depending on the size of each queue, the amount of corresponding activated VMs varies. This approach reduces the number of idle nodes in clusters as compared to clusters with a static partition scheme of the slots. The solution we proposed is different as we provide a single scheduling environment that acts on the jobs instead of acting on the number of slots per queue.

All of these works addressed the lack of flexibility in resource management systems and used VM mechanisms to improve it. Each solution addressed a particular use case.

All faced the same issues when manipulating VMs but all used ad-hoc mechanisms to solve them without considering a general approach as we described in this chapter.

Finally, several works addressed the interest of dynamic consolidation in datacenters to provide an efficient use of the resources. Khanna *et al.* (Khanna et al., 2006) and Bobroff *et al.* (Bobroff, Kochut, and Beaty, 2007) provided algorithms to minimize the unused portion of resources. However, they did not consider the sequencing and the cyclic issues when performing the migrations. Wood *et al.* (Wood et al., 2009) provideed a similar environment and exploit the page sharing between the VMs to improve the packing. They showed the interest of planning the changes to detect and avoid sequencing issues but did not consider inter-dependent migrations. In general, all of these solutions provide an algorithm to compute a viable configuration, specific to their objectives and use live migrations to perform the changes. However, their approaches are limited as they do not consider critical situations such as an overloaded cluster, with no viable assignment to satisfy all the resource requirements. Thanks to the suspend/resume mechanisms provided by the cluster-wide context switch, these situations become easily manageable and actions are performed more efficiently due to a finer plan.

2.5 Summary

For an efficient use of resources in clusters, advanced scheduling algorithms require preemption, migration and dynamic allocation of resources to the jobs. These mechanisms must be considered by developers in addition to the implementation of the algorithm that selects the jobs to run. This increases the complexity of developing advanced algorithms and prevents their deployment on clusters. To relieve developers of the burden of dealing with those operations, we defined the cluster-wide context switch building block. By leveraging this mechanism, developers can focus on the algorithm to select the vjobs to run. The cluster-wide context switch infers and plans the actions to perform on the VMs, ensures the feasibility of the process and selects, among the multiple satisfying solutions, the one implying the fastest reconfiguration between the current virtual machines/physical machines mapping and the expected one.

We validated our proposal by evaluating a sample strategy in charge of scheduling NASGrid jobs on top of Grid'5000 servers. As expected, the use of migration and preemption provided by the cluster-wide context switch block improves the whole resource usage compared to a static approach.

While this study brought important clues on how advanced scheduling strategies should be designed in cloud computing infrastructures, it also raised several questions.

The first question we identified concerns the importance of having a specific language that will enable developers to declare in details the composition of the vjob they submit. Keeping in mind that a vjob can be composed of several VMs, a scheduling strategy should not only take into account resource violations but also placement constraints of vjobs. This is critical as some placement decisions can be inefficient and even dramatic in some situations: For instance, a developer may expect to have high availability guarantees by replicating a VM several times. However, the scheduling strategy can relocate, in association with the cluster-wide context switch, those VMs on the same physical server, breaking the HA objective.

The second question is related to the scalability of the proposed mechanism and more generally about the scheduling challenge that is an NP-hard problem (the time needed to solve it grows exponentially with the number of nodes and VMs, which are considered). Because the time to compute a new scheduling placement as well as the time to apply the context-switch is important, the schedule may be outdated because workloads may have changed before it is eventually applied. Moreover, during these two phases, the scheduler manager does not enforce the QoS anymore, and thus can-

not react quickly to new QoS violations. It is therefore very important to propose some optimizations to reduce these two phases as much as possible.

The last question is linked to the previous one, it consists in improving the cost function that we use in the cluster wide context switch mechanism to identify the best reconfiguration plan (see Section 2.2.2). The objective is to estimate the duration of a cluster-wide context switch as accurate as possible. To this aim, additional investigations first on the cost of each action and second on the side effects of performing them simultaneously are required.

The first question has been addressed by my colleague Dr. Hermenier in (Hermenier, Lawall, and Muller, 2013). The second question is largely discussed in the next chapter. Finally, the last question is still an open issue. However, the work we did on modeling VM operations as described in Sections 5 and 6 has paved the way toward more advanced cost functions.

Distributed Virtual Machine Scheduler

3

As discussed in the previous chapter, encapsulating jobs into VMs allows developers to use preemption as well as relocation mechanisms to implement advanced scheduling strategies. These strategies enhance the optimization of systems utilization while guaranteeing Quality of Service (QoS) properties at the same time. While several studies promote the implementation of dynamic VM scheduling, they all face strong limitations regarding scalability, reactivity and fault-tolerance aspects. Our second contribution in this area has been to use the VM context switch building block, which has been previously introduced, and P2P mechanisms to address the aforementioned challenges.

In this chapter, we first describe DVMS (Distributed VM Scheduler), a framework that enables VMs to be scheduled cooperatively and dynamically in large-scale distributed systems. We describe, in particular, how several VM reconfigurations (a.k.a., *VM context switches) can be dynamically calculated in parallel and applied simultaneously. We evaluate our prototype by means of ad-hoc simulations and in-vivo experiments and compare it to a well-known centralized approach. The corresponding results show that our scheduler is able to manage hundreds of nodes and thousands of VMs in a few seconds.*

The second section deals with an optimization we proposed in DVMS. Although the first version improves the scalability criterion significantly, it does not consider the physical topology of the network. However, taking into account network considerations is critical for large-scale infrastructures, in particular when they are deployed throughout distinct geographical sites. The second contribution we present is a new building block designed on top of the Vivaldi overlay. Thanks to the Vivaldi coordinates and a shortest path based algorithm, DVMS is able to solve and perform each VM context switch by using the closest servers available in the neighborhood. Experiments confirm that without changing the scheduling decision algorithm, the number of WANWide reconfiguration as been considerably reduced.

The last section discusses related work.

3.1 Addressing Scalability, Reactivity and Fault-tolerant Aspects

In this section, we present how it can be possible to schedule VMs in order to maximize system utilization while ensuring the quality of service in large-scale distributed systems (*i.e.*, composed of thousands of physical servers).

After presenting in Section 3.1.1 the limitations of centralized-based scheduling approach, we give an overview of the DVMS proposal in Section 3.1.2. In Section 3.1.3, we discuss a few experiments we performed on top of Grid'5000 to validate our Proof-of-Concept. Finally, we conclude by giving a summary of the major results we obtained and higlight two research directions this work opened in Section 3.1.4.

3.1.1 Challenge Description

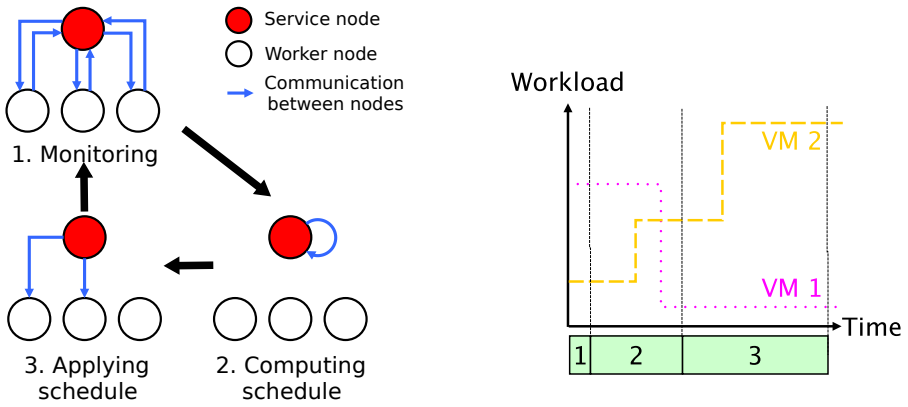
Although VM scheduling is an NP-hard problem [1], most of the proposals that promote the use of VMs to implement dynamic VM scheduling (Hermenier, Lorca, et al.,

1 The time needed to solve it grows exponentially with the number of nodes and VMs considered

2009; Sotomayor, Rubén S Montero, et al., 2009; Lowe, 2010) rely on a master/worker design (see Figure 12a). In this model, a single daemon is in charge of performing the different phases: monitoring all resources, computing the schedule and applying the reconfiguration.

The choice of a single master node leads to several problems. First, during the computation and the application of a schedule, these managers do not enforce QoS properties anymore, and thus cannot react quickly to QoS violations.

Second, since the manipulation of VMs is costly as discussed in the previous chapter, the time needed to apply a new schedule is particularly important: the longer the reconfiguration process, the higher the risk that the schedule may be outdated, due to the workload fluctuations, when it is eventually applied (see Figure 12b). Finally, a single master node can lead to well-known fault-tolerance issues: a group of VMs may be temporarily isolated from the master node in case of a network disconnection; moreover, QoS properties may not be ensured anymore if the master node crashes.



(a) *Scheduling steps*

(b) *Workload fluctuations during scheduling*

Figure 12: *VMs scheduling in a master/workers architecture*

Even if a better design of the master can help to separate each phase in order to resolve some of these issues (a multi-threaded model, for instance, can help to track workload changes so that a scheduling process can be stopped if need be), a centralized approach will always be subject to scalability, reactivity and fault tolerance issues. Considering the increasing size of data centers, these concerns are predominant (Kotsovinos, 2011; Vaquero, Rodero-Merino, and Buyya, 2011).

To address this challenge, we investigated whether a more decentralized approach can tackle the aforementioned limitations. Indeed, scheduling takes less time if the work is distributed among several nodes, and the failure of a node does not impede scheduling strongly anymore. We introduced DVMS (*Distributed VM Scheduler*), a manager that schedules and supervises VMs cooperatively and dynamically in distributed systems (Quesnel and Lebre, 2011; Quesnel, Lebre, and Südholt, 2013; Quesnel, Lebre, Pastor, et al., 2013; Balouek, Lebre, and Quesnel, 2013). Hierarchical solutions (Feller, Rilling, and Morin, 2012) that can be seen as good candidates face important limitations: First, finding an efficient partitioning of resources is a tedious task as matching a hierarchical overlay network on top of a distributed infrastructure is often not natural. Secondly, in addition to requiring complex failover mechanisms to face crashed leaders/super peers and network disconnections, hierarchical structures have not been designed to react swiftly to physical topology changes such as node apparitions/removals and network performance degradations.

P2P algorithms allow to address both concerns, *i.e.*, scalability as well as resiliency of infrastructures. Few proposals have been made precisely to distribute the dynamic management of VMs (Rouzaud-Cornabas, 2010; Yazir et al., 2010). However, the resulting prototypes are still partially centralized. Firstly, at least one node has access to a global view of the system. Secondly, most solutions consider all nodes for schedul-

ing, which limits scalability. Thirdly, they still rely on service nodes, that are potential single points of failure (SPOF).

Our proposal has been designed to work with partial views of the system, without any SPOF, and to be non-predictive and event-driven. This design rules enabled us to propose a system that is more reactive, more scalable and more tolerant to nodes crashes or network disconnections in comparison with available approaches.

3.1.2 Our Proposal: DVMS

In this section, we describe the foundations of the DVMS proposal. Then we present its main characteristics and focus on on two major parts: (i) its iterative scheduling procedure and (ii) the deadlock detection/resolution mechanism.

In DVMS, when a node cannot guarantee the QoS for its hosted VMs or when it is under-utilized, it starts an iterative scheduling procedure (ISP) by querying its neighbor to find a better placement. If the request cannot be satisfied by the neighbor, it is forwarded to the following free one until the ISP succeeds. This approach allows each ISP to send requests only to a minimal number of nodes, thus decreasing the scheduling time without requiring a central point. In addition, this approach allows several ISPs to occur independently at the same moment throughout the infrastructure. In other words, scheduling is performed on partitions of the system that are created dynamically, which significantly improves the reactivity of the system. It should be noted that nodes are reserved for exclusive use by a single ISP, in order to prevent conflicts that could occur if several ISPs performed concurrent operations on the same nodes or VMs. Even if such a strategy allows to maintain a consistent state of the infrastructure, it may lead to deadlocks when all nodes are reserved by active ISPs. To tackle this issue, we proposed a (distributed) deadlock detection and resolution mechanism. Moreover, communication between nodes is done through a fault-tolerant overlay network, which relies on distributed hash table (DHT) mechanisms to mitigate the impact of a node apparitions/removals (Ratnasamy et al., 2001; Rowstron and Druschel, 2001; I. Stoica et al., 2003).

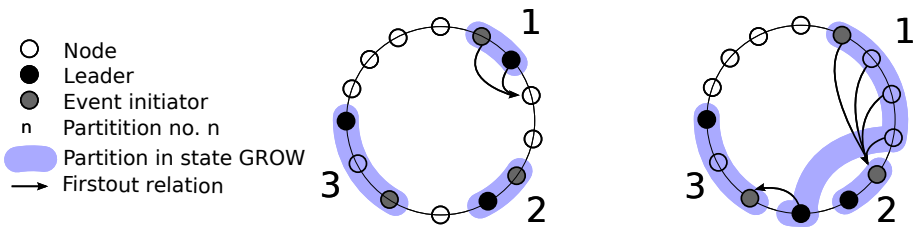
Architecture Overview

In order to develop a scheduling mechanism that is reactive, scalable and reliable, we designed it on the basis of a peer-to-peer and event-based approach. A scheduling activity is started on demand, *i.e.*, only if a (scheduling request) event is received. Since such events may occur simultaneously throughout the infrastructure, we propose to manage each event independently so that several events can be tackled in parallel, thus leading to better reactivity. This approach contrasts with more traditional solutions, where scheduling is started periodically. It also differs from a predictive strategy that computes schedules in advance to anticipate workload fluctuations. Although predictive approaches may help to prevent the start of scheduling processes that finally prove to be superfluous, prediction requires an accurate knowledge of workload profiles, which is not always available. In our proposal, an event may be generated each time a virtualized job (vjob, see Chapter 2) is submitted or terminated, when a node is overloaded or underloaded, or when a system administrator wants to put a node into maintenance mode.

Regarding the peer-to-peer aspects of our approach, our system does not distinguish any node: all nodes behave equivalently as far as scheduling is concerned. Each node can (i) submit vjobs, (ii) generate events and (iii) handle events generated by other nodes. A node monitors only its local resources. However, it can get access on demand to a partial view of the system, by communicating with its neighbors by means of an overlay network which is similar to those used to implement distributed hash tables (Ratnasamy et al., 2001; Rowstron and Druschel, 2001; I. Stoica et al., 2003). To facilitate understanding, we consider here that the overlay provides a ring topology. Accessing a partial view of the system improves scalability (computing and

applying a schedule on a small subset is faster) while the DHT mechanism enhances fault-tolerance (the nodes can continue to communicate transparently even if several of them crash).

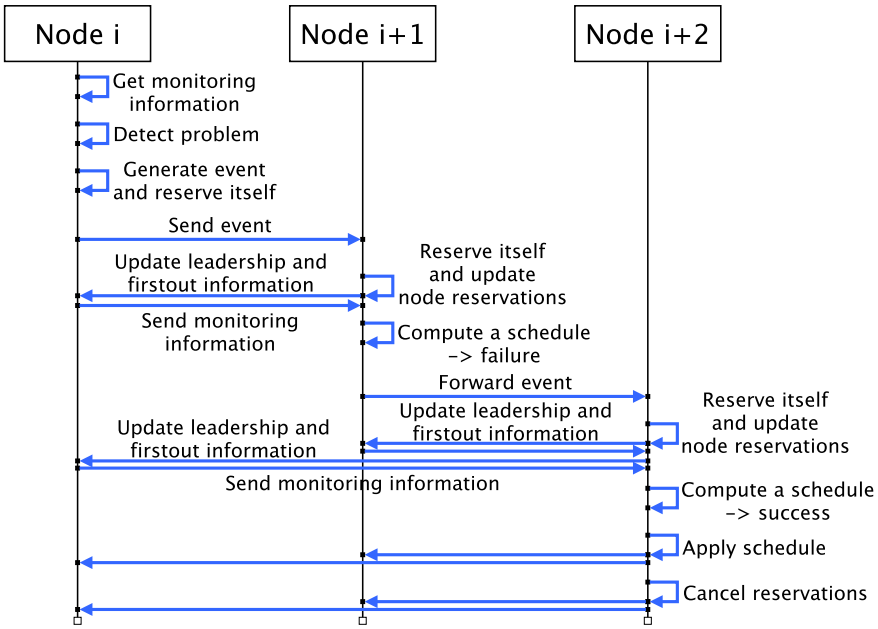
Figure 13: *Iterative scheduling: example*



The Iterative Scheduling Procedure

The ISP defines the different actions that should be applied when a (scheduling request) event occurs on a node; the goal of the ISP is the reservation of a subset of nodes, a so-called *partition*, that is sufficient in order to solve the (scheduling) event that motivated the start of the ISP. Overall, the ISP proceeds by successively reserving free nodes (and leapfrog sets of already reserved nodes) until an appropriate set of nodes is reserved or a deadlock is reached. In the latter case, deadlocks are handled as shown in the following section.

Figure 14: *Iterative scheduling: algorithm*



In order to maintain a consistent state in an environment in which several ISPs can occur simultaneously, each node can take part only in one ISP at a time. Each node monitors its local resources.

When a node N_i detects a problem (e.g., it is overloaded), it starts a new ISP by generating a scheduling event (we refer to the original node that initiated the ISP as the *initiator*). The ISP reserves the current node and sends the event to its neighbor, the node N_{i+1}. If the node N_{i+1} is not already involved in another ISP, the node N_{i+1} reserves itself, i.e., joins the *growing partition*, and becomes the new *leader* of the partition. In Figure 13, two steps during the execution of three on-going ISPs are shown: partitions are colored; leaders are marked as black, initiators as gray nodes (we explain the meaning of the arrows later on). From the first to the second execution state, the first partition has grown from two to five nodes.

The *leader* is the node that is actively trying to solve the event. First, the *leader* notifies the nodes involved in the ISP that it has taken the leadership. Then, it retrieves up-to-date monitoring information from these nodes. Finally, it computes a new schedule based on this information. If this schedule does not meet the resource requirements of the initial scheduling event, the *leader* forwards the event to its neighbor, node N_{i+2} (see the corresponding sequence diagram shown in Figure 14). Similarly, if the node N_{i+2} is not involved in an another ISP, the node joins the partition, becomes the new *leader* and performs the same operations as previously described. If the computation of the new schedule succeeds, node N_{i+2} applies it (e.g., by performing appropriate VM migrations) and finally releases all members of the partition, so that nodes N_i, N_{i+1} and N_{i+2} are free to take part in another ISP.

To avoid useless communications and to determine the next available node for an ISP (i.e, the next *leader*), each node maintains a 'first out' relation. This value indicates the next node that does not belong to the current partition. Thus, when an already reserved node receives an event, it forwards the request directly to its 'first out' node. This approach allows us to efficiently handle the transience of ISPs, that appear and disappear according to events and their resolutions, as well as to efficiently contact the following potentially free node and thus limit the communication with nodes that are already taking part in another partition.

Finally, note that if a node detects a problem while it is already involved in an ISP, the event is not forwarded to its neighbor; the update process performed by the *leader* enables its integration into the following scheduling computation.

Deadlock Prevention

Resource requests may lead to deadlock situations in which no partition can satisfy its corresponding scheduling request since there are no free node left. It is therefore necessary to prevent deadlocks. Figure 15 shows an example of two deadlock-related execution states: a situation just before a deadlock on the left, another one just after deadlock resolution on the right.

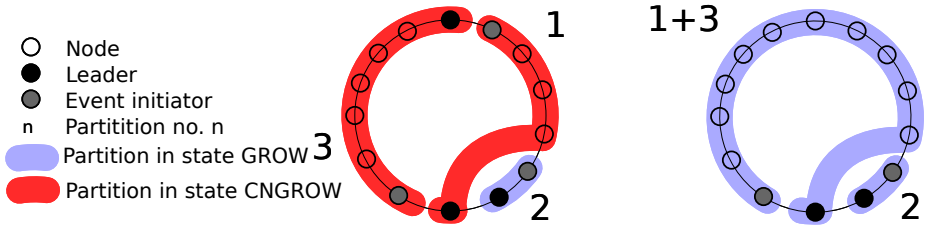


Figure 15: *Handling deadlocks*

Conceptually, we prevent deadlocks as follows. Partitions are used to handle resource requests by reserving free nodes. They are therefore in state `GROW` (indicated by the set of nodes colored in light blue in figures 13 and 15). In a potential deadlock situation, partitions become aware that they cannot grow anymore and switch to state `CNGROW`, that stands for 'cannot grow' (the corresponding partitions are colored in dark red on Figure 15). Note that the partition no. 2 on the left hand side has not yet switched to that state. Once a deadlock has been identified, two partitions (or several pairs) in state `CNGROW` will merge and thus provide more nodes to try to solve the two ISPs simultaneously. This is shown on the right hand side, where partitions no. 1 and 3 have been merged and reached state `GROW` once again. In this example, the merged partition may now be sufficient after which the corresponding computations are started and the partition is dissolved.

Note that merging partitions does not guarantee that a solution will be found, it only gives better chances to solve the corresponding events by providing more possibilities to move the VMs. It is especially true if complementary events are combined (e.g., an 'overloaded node' event and an 'underloaded node' one).

From the algorithm viewpoint, deadlock prevention is performed in a decentralized manner during event processing as follows (see Figure 16). The ISP of a growing partition sends a (scheduling) event around the ring in order to find free nodes. If the

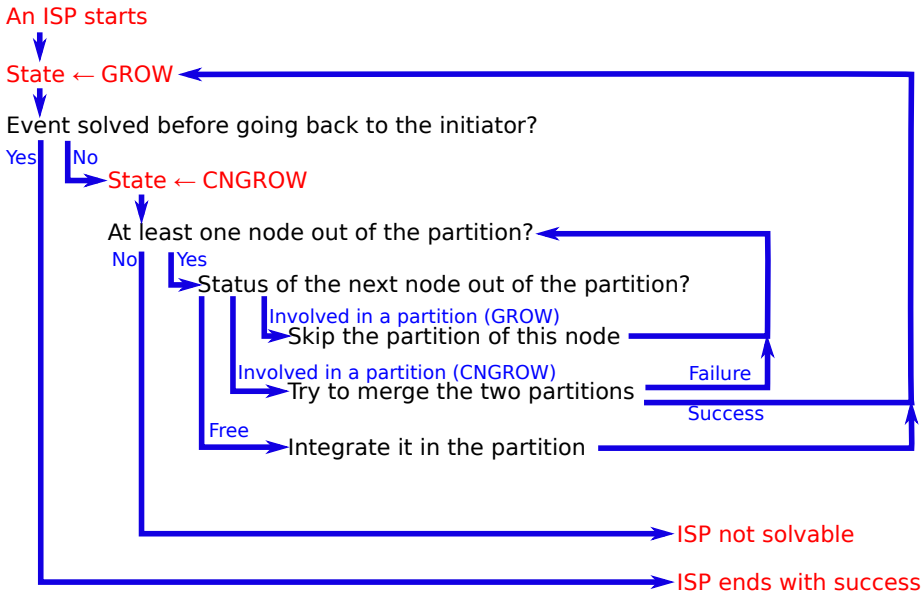


Figure 16: *Per-node event processing algorithm*

event comes back to the *initiator*, the partition switches to state `CNGROW`. After that, this partition will switch back to state `GROW` if:

- The ISP finds free nodes (e.g., if another partition was dissolved).
- The ISP finds another partition in state `CNGROW` and is able to merge its partition with it.

This procedure is correct in that it permits all deadlock situations to be prevented. It may indicate deadlock situations even if there are some free nodes, because a partition may be freed after the deadlock prevention algorithm has visited that partition. This is, however, unavoidable in a dynamic, asynchronous setting as ours, and has a negligible impact on the performance of our algorithm.

Although we do not discuss it in this manuscript, We highlight that we have proven the correctness, including deadlock freedom and absence of starvation, of our algorithm by means of a (manual) formal development using the formal framework of first-order linear temporal logic(see (Quesnel, Lebre, Pastor, et al., 2013) for further details).

3.1.3 Proof-Of-Concept & Validation

To validate our proposal, we implemented a Proof-of-Concept in Java. This prototype enabled us to conduct ad-hoc simulations as well as *in-vivo* experiments that we present in this section

Proof-Of-Concept

The prototype processes 'overloaded node' and 'underloaded node' events (defined by means of CPU and memory thresholds by the system administrator). The ISP relies on a a simple (unidirectional) ring (see Figure 17). This enabled us to evaluate the benefits in scalability and reactivity of our proposal as discussed in the following section.

From the software architecture viewpoint, our system is composed of a node agent (NA) per node. Each NA is made of a knowledge base, a resource monitor, a client, a server and a scheduler (see Figure 17).

The knowledge base contains various types of information. Some information is available permanently: monitoring information about the local node (resources consumed and VMs hosted), a stub to contact the neighbor, and a list of events generated

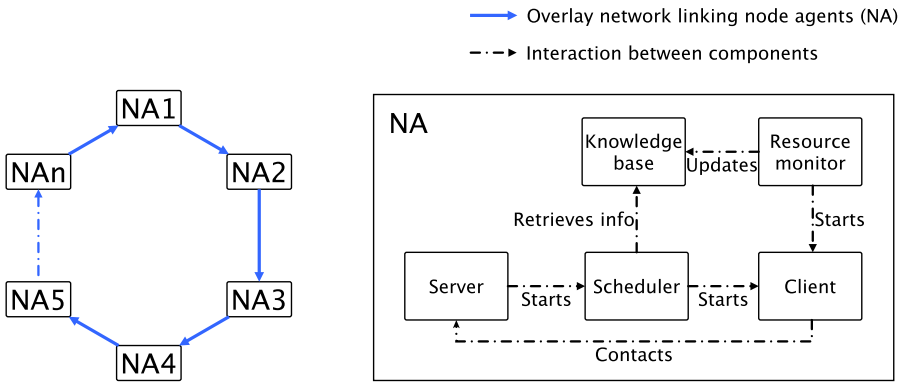


Figure 17: *Implementation overview*

by the node. Other information is accessible only during an iterative scheduling procedure: monitoring information about the reserved nodes (if a scheduler is running on the node) and a stub to contact the scheduler that tries to solve the event.

The resource monitor is in charge of updating the monitoring information every 2 seconds. If it detects a problem while it is not already involved in another ISP, it triggers a new ISP by generating the appropriate event (e.g., to indicate that the node is overloaded), reserving the node and sending the event to the neighbor by means of a client (the latter being instantiated on-demand to send a request or a message to a server).

The server processes requests and receives messages from other nodes. It launches, in particular, a scheduler when it receives an event.

The scheduler first updates the leadership information of the other members of the ISP and retrieves the monitoring information from each member as part of the acknowledgment. Based on the information it received, it tries to solve the corresponding event by computing a new schedule and applying it, if possible. If the schedule is applied successfully, the scheduler finally cancels node reservations.

Last but not the least, the prototype has been designed so that any dynamic VM scheduler may be used to compute and apply a new schedule. However, we underline that DVMS relies by default on the latest proposal of the Entropy system (Hermenier, Demassey, and Lorca, 2011).

Validation

DVMS has been evaluated through ad-hoc simulations and *in-vivo* experiments. We do not discuss in details all experiments in this manuscript and invite the reader to refer to (Quesnel, Lebre, and Südholt, 2013; Quesnel, Lebre, Pastor, et al., 2013) for further information. We chose to only discuss large-scale evaluations involving up to 4.7K VMs distributed over 467 nodes of the Grid'5000 testbed. As far as we know, these experiments constitute the largest *in vivo* validation that has been performed so far with decentralized VM schedulers.

The objective of our experiments was to validate the DVMS relevance to improve scalability as well as reactivity of a centralized scheduling policy on real experiments. The scheduling algorithm we used in all experiments was the latest one that has been developed as part of the Entropy framework as mentioned previously. Giving up consolidation optimum in favor of scalability, this algorithm provides a 'repair mode' that enables overload problems to be corrected according to the requirements of VMs and placement constraints (Hermenier, Lawall, and Muller, 2013). At each invocation, the algorithm looks for the optimal solution until it reaches a predefined timeout. The optimal solution is the one that implies the cheapest reconfiguration plan while satisfying resource and placement constraints during relocation operations. Once the timeout is triggered, the algorithm returns the best solution among the ones it has found so far.

Experimental Parameters:

The number and the characteristics of the worker nodes used during the three experiments are shown respectively on Figure 18 and on Table 2.

We developed a set of scripts to make the control of the experiments easier. These scripts have been capitalized in a framework entitled vm5k [2] (Balouek, Lebre, and Quesnel, 2013; Imbert et al., 2013) in charge of:

- Installing a software stack on each Grid'5000 node, including the KVM hypervisor and the Libvirt virtualization library.
- Deploying KVM virtual machines, so that Entropy and DVMS are evaluated on the same configurations.
- Iteratively injecting CPU workloads into the different VMs by invoking the `stress` command; it is worth noting that the framework turns to a new iteration when all 'overloaded node' events have been solved.

Figure 18: *Number of nodes used during the experiments on Grid'5000*

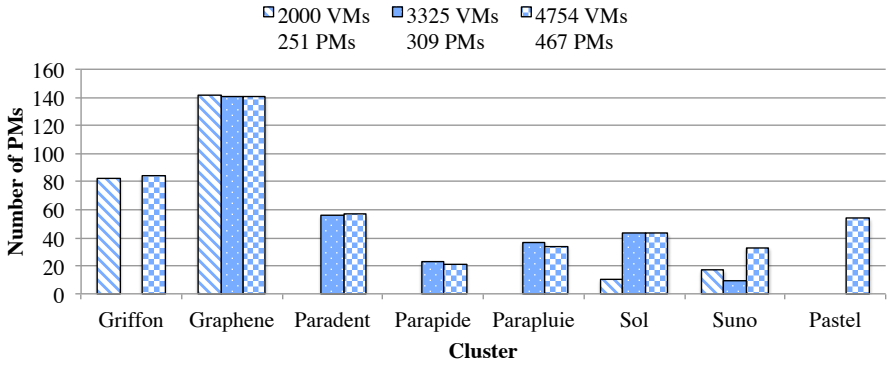


Table 2: *Characteristics of the nodes used during the experiments*

Site	Cluster	Processor	RAM (GB)
Nancy	Griffon	Intel; 2*4 cores; 2.50Ghz	16
	Graphene	Intel; 1*4 cores; 2.60Ghz	16
Rennes	Paradent	Intel; 2*4 cores; 2.50Ghz	32
	Parapide	Intel; 2*4 cores; 2.93Ghz	24
	Parapluie	AMD; 2*12 cores; 1.70Ghz	48
Sophia	Sol	AMD; 2*2 cores; 2.60Ghz	4
	Suno	Intel; 2*4 cores; 2.26Ghz	32
Toulouse	Pastel	AMD; 2*2 cores; 2.61Ghz	8

vm5k was running on a dedicated node on the suno cluster. The software stack of each node is as follows: Linux Debian 6 (Squeeze) 64 bits, KVM 1.1.2, virt-install 0.600.1, virsh 0.9.12, vm5k, OpenJDK JRE 6, Entropy 2.1 and DVMS.

Each VM had 1 virtual CPU and 1 GB of RAM and is attached to a 1 GB COW disk image leveraging a common backing file previously deployed on each node. The CPU consumption of each VM was set randomly (with a seed for reproducibility) to 0% or 100% of its virtual CPU. During each iteration, the percentage of VMs that consumed 100% of their virtual CPU was between 40% and 70%.

The overloaded threshold was configured at 100% (i.e., a PM was considered ad overloaded if the VMs it hosted tried to consume more than 100% of its CPU resources). The number of overload problems injected per iteration was on average:

- 26 (standard deviation: 22) for the experiment with 251 PMs and 2000 VMs.

2 https://github.com/lpouillo/vm5k

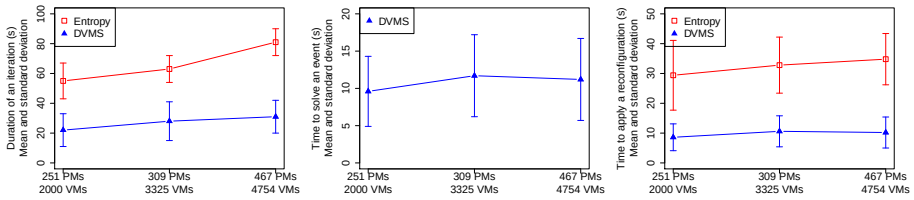
- 28 (standard deviation: 21) for the experiment with 309 PMs and 3325 VMs.
- 40 (standard deviation: 35) for the experiment with 467 PMs and 4754 VMs.

Each experiment was performed on 10 iterations. Migrations have been performed using the `virsh` command with the `copy-storage-inc` option.

Finally, the Entropy timeout was determined empirically and configured so that the time (in seconds) to compute a reconfiguration did not exceed one tenth of the number of nodes considered for scheduling.

Results:

Figure 19 shows the results. The mean value and the standard deviation are given for each experiment.



(a) *Duration of an iteration* (b) *Time to solve an event with DVMS* (c) *Time to apply a reconfiguration*

Figure 19: *Experiments in vivo on the Grid'5000 testbed*

We observe that, on average, DVMS improves the reactivity of Entropy by a factor two (cf. Figure 19a). This can be explained by the fact that DVMS considers each problem independently and hence can process each of them simultaneously. On average, each problem is solved in less than 12 seconds, cf. Figure 19b)

We can see that the time required to apply a reconfiguration cannot be neglected (cf. Figure 19c). Entropy spends approximately as much time to compute a reconfiguration as to apply it (cf. Figures 19c and 19a). However, on the DVMS side, the time needed to solve an overload problem is strongly dominated by the time required to apply the corresponding reconfiguration (cf. Figures 19c and 19b). Indeed, since each problem affects only few nodes, the time needed to compute a reconfiguration is very small (less than 1 second on average) comparing with the time to perform a migration including the COW file.

Regarding the number of migrations more precisely, DVMS performed as many migrations as Entropy for each iteration since these two frameworks were evaluated with the same configurations and had to solve exactly the same overload problems.

3.1.4 Summary

To address scalability, reactivity and fault tolerance aspects of virtualized resources managers, we proposed DVMS, a new P2P-based VM manager. Keeping in mind the following objective: maximizing system utilization while ensuring the quality of service., DVMS schedules VMs dynamically and cooperatively in distributed systems, To validate the behavior of DVMS as well as the resulting overhead on the nodes composing the infrastructure, we conducted several experiments through ad-hoc simulations and *in vivo* experiments. In particular, the experiments, which have been presented in this manuscript, are as far as we know, the only ones conducted at such as scale(i.e. up to 4.7K VMs on 467 nodes). Results showed that a cooperative approach like DVMS is appropriate to solve resource violations in an highly reactive and scalable manner.

To consolidate our study, it would have been interesting to perform additional experiments on the resilience aspects, when nodes join or leave the system. Indeed, a key element of the DVMS proposal is the network overlay that enables peers to communicate throughout the ring. Although, we propose to leverage parts of the Chord

algorithm (I. Stoica et al., 2003) in order to manage node additions and removals in an autonomous way (that is, node predecessors as well as node successors are notified at each modification of the ring structure), it would have been interesting to measure the impact of the ring's modifications on the ISP performance. These experiments would have enabled us to identify whether DVMS can cope with a high-churn rate.

Another limitation regarding DVMS is related to the "simple" ring approach it leverages. Even though DVMS has significantly improved the scalability, it does not consider the phyiscal topology of the network and thus does not favour cooperations between nodes that are close to each other. While this is acceptable within an infrastructure where nodes are interconnected through low latency and high bandwitdh networks, it becomes critical when the scheduler has to perform reconfigurations between nodes of different racks/sites. To address this issue, we proposed a revision of our proposal as discussed in the next section.

3.2 Locality Aware-Scheduling Strategy

The ever-increasing size of modern data-centers as well as the promotion of distributed Cloud Computing infrastructures like the next platform to deliver the Utility Computing paradigm, had favor the investigations on distributed VM scheduling algorithms (Feller, Morin, and Esnault, 2012; Quesnel, Lebre, and Südholt, 2013). Although these proposals considerably improve the scalability, leading to the management of hundreds of thousands of VMs over thousands of physical machines (PMs), they do not consider the network overhead introduced by multi-site infrastructures. This over- head can have a dramatic impact on the performance if there is no mechanism favoring intra-site *vs.* inter-site manipulations.

In this section, we introduce a new building block designed on top of a Vivaldi overlay network in order to maximize the locality criterion (*i.e.*, efficient collaborations between PMs. We combined this mechanism with DVMS and showed its benefit by discussing several experiments performed on four distinct sites of Grid'5000. In details, Section 3.2.1 describes the challenge we addressed. Section 3.2.2 gives an overview of our proposal by introducing the short path algorithm on top of Vivaldi and the way we integrate it into DVMS. In Section 3.2.3, we validate the proposal by analyzing its benefits with respect to the previous version of DVMS. Finally, Section 3.2.4 concludes and raises research questions.

3.2.1 Challenge Description

System mechanisms in charge of operating cloud computing infrastructures rely more and more on P2P buidling blocks. Although it has been used first on storage services such as key/value stores (DeCandia et al., 2007), several academic studies have investigated the relevance of P2P algorithms in other mechanisms. The design of new scheduler managers such as DVMS is one example among others. However, P2P proposals still face limitations coming from the overlay network they rely on. DVMS, for instance, maps a ring overlay network on a distributed infrastructure. Such an overlay prevents DVMS to make any distinction between close nodes and distant ones, limiting the optimization that can be performed. Similarly, the approach described in (Feller, Morin, and Esnault, 2012), while adopting an orthogonal, gossip-based approach, still suffers from building a randomized overlay network, thus breaking the physical topology.

Considering that both the network latency and the bandwidth between peers have a strong impact on the reactivity criterion of the scheduling problem (see Section 2.2.1 and Chapter 5, *locality* properties of peers should be considered to favor efficient VM operations. In other words, to reduce as much as possible the time to perform a VM context switch (*i.e.*, to switch from a mapping between VMs and PMs running in the infrastructure to another one), it is crucial to make cooperation first between peers in

the closest neighborhoods before contacting peers belonging to other sites. Moreover, it is noteworthy that this notion of locality is dynamic, and varies over time according to the network bandwidth/latency and disconnections.

We illustrate the problem in Figure 20. In this example, we have three partitions and we can see the growth of partition 1 between two steps [3].

While the ISP strategy enables DVMS to limit the size of one partition to a minimal number of nodes, these nodes are selected without considering the network conditions at the time the ISP starts. This leads to inefficient situations where VM migrations occur between two nodes that are far from each other, which lasts longer than a migration between two close nodes. Obviously the ring can be built to limit the distance between peers globally (*i.e.*, peers of the same region/area would be grouped together as illustrated in Figure 20). However, in such a case, at least two nodes of each group are directly connected to two far nodes. Note that an approach such as the one proposed in (Garces-Erice et al., 2003), which consists in deploying one ring per site and relying on a *super-ring* to interconnect few representatives of each local ring, would not solve many problems. Besides problems inherent to hierarchical and structured overlay networks, this solution would not provide a good answer to locality: When going out of the local ring, it would still not be possible to find the next closest ring.

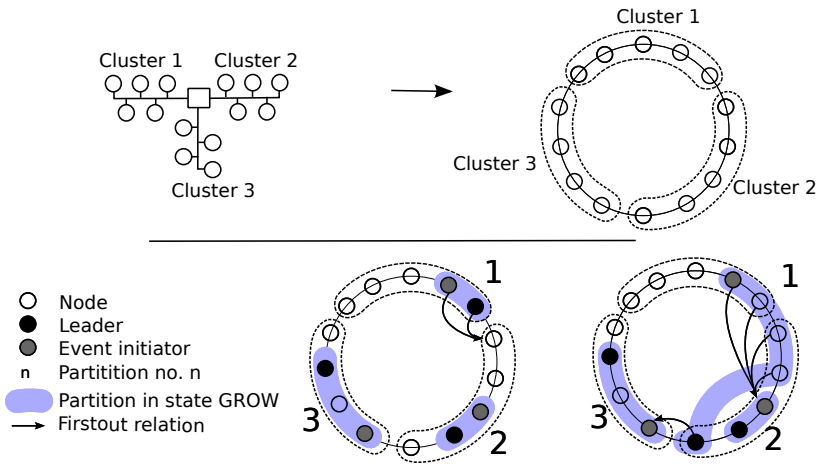


Figure 20: *Solving three problems simultaneously and independently with DVMS. The ring has been matched on top of three distinct clusters.*

3.2.2 Our Proposal: A Locality-aware Overlay Network

As illustrated in the previous paragraph, one of the primary downsides of overlay networks lies in that they break the physical topology by connecting nodes that have no physical proximity. After presenting previous initiatives that dealt with this notion of peers locality, we present our proposal.

Overlay Networks and Locality

Besides hierarchical attempts in building locality-aware overlay networks (Garces-Erice et al., 2003; Z. Xu, Mahalingam, and Karlsson, 2003; Z. Xu and Z. Zhang, 2002), we can first mention the locality improvement mechanisms of the Pastry structured overlay network (Rowstron and Druschel, 2001). In order to reduce the latency of the routing process, each node is given the opportunity to choose the closest nodes to fill its routing table. Learning the existence of new nodes relies on a periodic exchange of parts of routing tables.

3 For further information regarding the notion of "first out", see Section 6.3. In this example, readers can consider that the "first out" relation enables DVMS to handle communications efficiently, as each node involved in a partition can forward a request directly to the first node on the outside of its partition.

Similar mechanisms have been adopted within unstructured overlay networks to make their logical connections reflect the physical proximity of nodes, each node discovering its closest nodes through gossiping. Note that the proximity between two nodes can be estimated through any transitive metric, in particular the latency between the nodes (Jelasity and Babaoglu, 2005).

These approaches need to constantly maintain the knowledge of close nodes in order to provide the *best* node possible at the cost of periodic communications (uncorrelated to the actual amount of requests to be processed by the overlay network).

The overlay network we proposed differs in that it adopts a lazy approach consisting in searching close nodes only upon receipt of requests. This way, the quality of the response is proportional to the frequency of requests.

Our protocol relies on the Vivaldi protocol (Dabek et al., 2004) to detect close nodes. Vivaldi places nodes in a multi-dimensional space. Each node is given coordinates inside this space reflecting its physical location. The protocol is based on simple message exchanges. Initially, each node is given a random position in the space and chooses (possibly arbitrarily) a small subset of nodes, composing its *view*. Then, each node starts estimating the round trip time between itself and another node chosen randomly in its view, and adapts its distance with this node in the space accordingly, coming closer to it or moving away from it. The nodes can repeat this step independently (each with another node from its view), to improve the accuracy of the positioning. A globally accurate positioning of nodes can be obtained very quickly (in a small number of such steps) if nodes have a few long-distance nodes in their view and if the network is not excessively dynamic. These long distance links can be easily maintained.

Recall that Vivaldi does not allow to directly know the nodes that are close in the network, but to be able to recognize them through their coordinates. Our overlay relies on the examination of Vivaldi coordinates of nodes discovered during the processing of requests sent to it.

Locality-aware Overlay Network

The overlay network we proposed is made of two layers.

- The lower layer is mainly an implementation of the Vivaldi protocol, which allow nodes to be aware of their position in the infrastructure. We underline that nodes are initially interconnected arbitrarily.
- Based on these coordinates, the upper layer is responsible for building a locality-aware overlay dynamically. This layer takes its roots in the classic Dijkstra's shortest path algorithm to collect a set of close nodes starting from a given position.

Searching for Close Nodes:

Once the Vivaldi map is achieved, and each node knows its coordinates, we are able to estimate how *close* two given nodes are by calculating their distance in the map. However, recall that the view of each node does not *a priori* contain its closest nodes [4]. Therefore, we need additional mechanisms to locate a set of nodes that are close to a given initial node. Vivaldi gives a *location* to each node, not a neighborhood.

We use a modified, distributed version of the classic Dijkstra's shortest path algorithm that leverages the Vivaldi map to build such a neighborhood. More specifically, its goal is to build a **spiral**[5] interconnecting the nodes in the plane that are the closest ones from a given initial node.

4 In the following, we call this view the **network view**, to distinguish it from the **spiral view** to be introduced later.

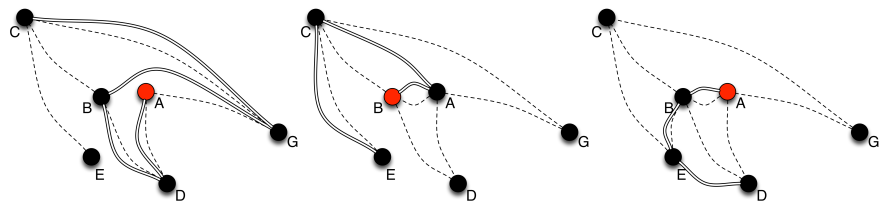
5 Our use of the term *spiral* is actually a misuse of language, since the graph drawn in the plane might contain crossing edges.

Let us consider that our initial (or root) point is the node n_R. The first step is to find a node to build a two-node spiral starting with n_R. This is done by selecting the node from n_R's network view, say n_i, which exhibits the smallest distance with n_R. n_i becomes the second node in the spiral. From this point on, n_R remembers n_i as its successor and n_i remembers n_R as its predecessor. n_R also sends its network view to n_i, which, on receipt, creates its **spiral view** that contains the N nodes closest to n_R taken from both n_R and n_i network views. It will allow n_i to find the next node to build the spiral. Assuming this closest node from n_R in n_i's spiral view is n_j, n_j will be added in the spiral by becoming the successor of n_i. n_j receives n_i's spiral view and creates and fills its own spiral view with nodes closest to n_R contained in both n_i's spiral view and n_j's network view. This algorithm is repeated until the amount of nodes requested by the application have been interconnected in the spiral.

Note that there is a risk to be blocked at some point, having a spiral view containing only nodes that are already in the spiral, hindering from extending it further. However, this problem can be easily addressed by introducing few long-distance nodes when the spiral view is created/updated.

Learning:

Applying the protocol described above, the quality of the spiral is questionable in the sense that the nodes that are actually close to the root node n_R may not be included.To improve the *quality* of the spiral, *i.e.*, to reduce the average distance from each of its nodes to the initial node, we rely on a learning mechanism coming with no extra communication cost: When a node is contacted to become the next node in one spiral, and when it receives the associated spiral view, it can also keep in its network view the nodes that are closer to itself, thus potentially increasing the quality of a future spiral construction. Such an improvement through learning is illustrated in Figure 21. Note that learning may also be used to constantly improve already built spirals. While providing obvious advantages, allowing it comes at the cost of changing links in the spirals dynamically, which may not match all applications' constraints.



(a) *Initial View* (b) *B creates a "spiral"* (c) *A creates a "spiral"*

Figure 21: *Learning Mechanism (a) The initial view of each node is materialized by the dashed lines. Given these views, the spiral obtained from node A is represented by the double thick lines. In particular, this spiral allowed A and B to discover each other. (b) If B starts building a spiral, it will start by contacting A. This spiral construction allows also E and B to discover each other. (c) If A is requested to start another spiral, it will exhibit an increased locality awareness.*

PeerActor: A Building Block to Abstract Overlay Networks

As a P2P scheduling algorithm, the DVMS proposal can be divided in two major components: (i) The ring overlay network and (ii) the protocol in charge of detecting and resolving scheduling issues. As our goal consists in taking into account locality criteria without changing the DVMS protocol, we designed a building block, *i.e., the Peer actor*, which enables us to revisit DVMS by abstracting the overlay network it relies on. At a coarse-grain level, the Peer actor can be seen as a generic layer for high level distributed services, providing network abstractions and robust communications between agents deployed on each node. By leveraging the Peer actor API, developers can focus on the service itself without dealing with node apparitions/removals and network disconnections.

From the software point of view, the Peer actor relies on modern software frameworks (Scala and Akka) following the actor model rules. In such a model, each in-

stance will collaborate by exclusively exchanging messages, and priority will be given to collaboration between close instances when using the locality-based overlay (LBO).

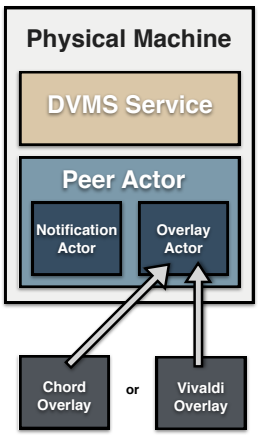


Figure 22: *DVMS on top of the Peer actor.*

As illustrated in Figure 22, the Peer actor contains two sub actors: The *Notification actor* and the *Overlay network actor*. The Notification actor enables services to subscribe to events that will be triggered by other services, as for detecting overloading of nodes or for handling crash of neighbours. The Overlay network actor is in charge of sending/receiving messages through the network. In order to compare both approaches, ring-based *vs.* locality-aware, we developed two different Overlay network actors: The first one provides a Chord-like overlay (I. Stoica et al., 2003), while the second one delivers the locality-aware overlay described in Section 3.2.2.

3.2.3 Proof-of-Concept & Validation

The main objective of the experiments we conducted was to estimate the impact of locality on the performance of a distributed scheduling algorithm. A significant portion of the reconfiguration time is spent in VM live migration operations, which depends of network parameters such as latency and bandwidth. One way to improve the performance of distributed scheduling algorithms is to promote collaborations between close resources, which can be reached by maximizing the ratio:

$$nb\ intrasite\ migrations / nb\ migrations.$$

Proof-of-Concept

A prototype of DVMS leveraging the *peer actor* abstraction has been developed. In addition we built two versions of the network overlay actor: one working with Chord, and one working with the Vivaldi based overlay. This mean that now DVMS is network overlay agnostic to, and thus can be used with either of the network overlay without requiring any modification in it's source code.

The implementation is based on modern programming language and framework such as *Scala* and *Akka framework*. Scala is a language that mixes object oriented programming with functional programming, it's compiler produces *Java bytecode* which can be run in any JVM environment. Combining Scala with Akka enabled us to take advantage of advanced techniques for concurrent programming such as *future/promise* and *actor model*, and to benefit from Java ease of deployment.

Experimental Protocol

To compare our experiments, we implemented a dedicated injector that makes load changes of VMs during a predefined time. VMs are launched on nodes in a round-

robin manner, *i.e.*, each node hosts roughly the same number of VMs at the beginning. The experiment consists in repeatedly changing target CPU loads of VMs. Every t seconds, the injector that is deployed on a dedicated node selects one VM and changes its CPU load according to a Gaussian distribution. t is a random variable that follows an exponential distribution with rate parameter λ. The Gaussian distribution is defined by a mean (μ) as well as a standard deviation (σ) that are given at the beginning of the experiment. The parameters are $\lambda = Nb\_VMs/300$ and $\mu = 70, \sigma = 30$. Concretely, the load of each VM starts from 0% and varies on average every 5 minutes in steps of 10 (with a significant part between 40% and 100% of CPU usage). The duration of each experiment was set to 3600 seconds.

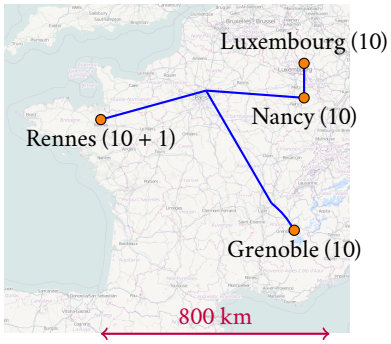


Figure 23: *Testbed.*

Figure 23 depicts our testbed. For each experiment, we booked 40 compute servers spread over 4 geographical sites (10 PMs per site) and 1 service server from the Grid'5000 testbed. The compute servers were used to run VMs and DVMS while the service node runs the aforementioned injector. Each compute node was equipped with 8 cores and hosted a number of VMs proportional to its number of CPU cores ($nbVM = 1.3 \times nb\ cores$), leading to a global number of 416 VMs. Although such a number is rather small regarding the latest experiments that have been performed on DVMS (Quesnel, Lebre, Pastor, et al., 2013), our goal is not to validate once again the scalability criteria but to focus on the locality aspect of such an algorithm.

Results

Maximization of Intra-Site Migrations:

Table 3 compares the ratio between intra-site migrations and the total number of migrations, using Chord or our LBO network. The results show that the impact of locality is significant: Using LBO leads to an average number of 86.3% of intra-site migrations while using a Chord-based DVMS decreases this ratio to 49.6%.

	Chord	LBO
Average	0.496	0.863
Minimum	0.378	0.798
Maximum	0.629	0.935

Table 3: *Comparison of intra-site migrations ratio (DVMS/-Chord vs. DVMS/LBO)*

Dynamic Clustering:

During our investigation of the results brought about LBO, we noticed that many of the inter-site migrations were performed between Luxembourg and Nancy sites. In Table 4, it is noticeable that Luxembourg and Nancy have a latency that is significantly below usual inter-site latencies (Nancy and Luxembourg are separated by only

100 kilometers), while Rennes and Grenoble have almost the same latency with all their respective remote sites. Indeed, servers located in Luxembourg and Nancy are more likely to collaborate with each other, while those located on Rennes and Grenoble will find collaborators regardless of their location. This explains why many of the inter-site migrations were performed between Luxembourg and Nancy. This means that LBO enabled DVMS to learn which site is more interesting to perform VM migration. Promoting low latency inter-site collaboration made many inter-site migrations acceptable compared to those executed by the Chord version.

Table 4: *Latency measured between sites.*

	Grenoble	Luxembourg	Nancy	Rennes
Grenoble	0.09 ms	16.55 ms	14.24 ms	15.92 ms
Luxembourg		0.17 ms	2.70 ms	13.82 ms
Nancy			0.27 ms	11.42 ms
Rennes				0.23 ms

Reactivity:

Table 5: *Comparison of partitions metrics using DVMS/Chord and DVMS/LBO.*

	Chord	LBO
Average number of sites involved	1.645	1.082
Average duration to detect a valid configuration (msec)	154.63	98.50

Table 5 depicts metrics that allow for an objective comparison of the efficiency of both overlay networks. In addition to reducing the number of inter-site migrations, the side effect of using the LBO is to reduce the solving time: The partition duration is 46% lower than that encountered with Chord. This result is consistent with the fact that with our locality-aware overlay, the number of sites that are involved in partitions becomes very close to one. Indeed collaborating with closer nodes allows exchanging information between nodes of the partition much faster, thus increasing once again the reactivity of the system.

3.2.4 Summary

In this section, we introduced a lazy locality-aware overlay network that acts as a generic building block to reify the *locality* aspects in distributed system mechanisms. In our study, the locality has been estimated through a cost function of the latency between peers in the network, thus enabling each peer to select its closest neighbors. We rely on Vivaldi (Dabek et al., 2004), a simple decentralized protocol allowing to map a network topology onto a logical space while preserving locality. On top of Vivaldi, a shortest path construction, similar to the well-known Dijkstra algorithm, is performed each time there is a need for cooperation between two nodes. We illustrated the advantage of this new building block by changing the overlay network in the DVMS proposal that has been previously discussed (see Section **??**).

Our experiments over Grid'5000 showed that, connecting 4 different sites and scheduling VMs over them, we could gain up to 72% of inter-site operations. It is worth noting that one experimental observation we had during this work was that the proposed overlay network was actually able to reflect the underlying topology, and in particular to build a hierarchical overlay dynamically if the underlying topology is hierarchical.

It is noteworthy that this study also opened two additional scientific challenges. The first question is related to the VM context switch operation and the decision

model used in the scheduling mechanisms. Indeed, it would be interesting to refine both of them in order to consider the cost difference between intra-site and inter-site migrations. By favoring intra-site migrations in multi-site partitions, it should be possible to significantly reduced the time to complete the VM context switch operation. The second question concerns the way of estimating the locality criterion. While we considered the latency criterion to identify the location of each node, it would have been interesting to consider the latency/bandwidth tulple. While the latency is rather stable, the bandwidth may significantly vary according to the on-going network exchanges that occur on the physical backbone: Two nodes that are geographically remote, can provide better performances than two nodes belonging to the same site. It would be interesting to perform additional experiments to see how our lazy-overlay is reactive with respect to such fluctuations.

3.3 Related Work

In this section, we present major work on distributed resource management, and especially notable approaches related to the dynamic scheduling of VMs. Contrary to previous solutions that performed scheduling periodically, latest proposals have tended to rely on an event-based approach: scheduling is started only if a scheduling event occurs in the system, for example if a node is overloaded.

In the Snooze project (Feller, Rilling, Morin, et al., 2010; Feller, Rilling, and Morin, 2012), nodes are distributed among managers and a front-end oversees the managers; when an event occurs, it is processed by a manager that takes into account all nodes it is in charge of. This approach considers both scalability and fault-tolerance concerns. However, it still relies on a single front-end to submit new VMs, which can limit its scalability. Moreover, the infrastructure is partitioned statically, which may prevent Snooze to handle events in an optimal way.

Most of the recent approaches try to delegate as much scheduling work as possible to the worker nodes.

With some distributed approaches, the worker nodes rely on a single service node to schedule VMs (Yazir et al., 2010; Mastroianni, Meo, and Papuzzo, n.d.), which may lead to scalability and fault-tolerance problems. (Yazir et al., 2010) uses a service node to collect monitoring information on all worker nodes; when an event occurs on a worker node, this node retrieves information from the service node, computes a new schedule and performs appropriate migrations. (Mastroianni, Meo, and Papuzzo, n.d.) uses a front-end to submit new VMs to the system; this front-end broadcasts the arrival of a new VM to all nodes; each node answers only if it accepts to host the VM; the front-end then chooses a node randomly among those which have responded; a migration is handled in the same way as a new VM submission.

With some other proposals, the scheduling is fully handled by the worker nodes, but they do require a global view of the system to schedule VMs (J. Xu, M. Zhao, and Fortes, 2009; Rouzaud-Cornabas, 2010). This global view may be kept up-to-date continually (J. Xu, M. Zhao, and Fortes, 2009) or on demand (Rouzaud-Cornabas, 2010), which may be a time-consuming operation if the infrastructure is big.

Finally, some proposals work with a partial view of the system (Barbagallo et al., 2010; Marzolla, Babaoglu, and Panzieri, 2011; Feller, Morin, and Esnault, 2012). (Marzolla, Babaoglu, and Panzieri, 2011) and (Feller, Morin, and Esnault, 2012) rely on a periodic approach; with (Marzolla, Babaoglu, and Panzieri, 2011), each node tries to exchange VMs with its neighbors, while with (Feller, Morin, and Esnault, 2012), a node takes the leadership and performs scheduling among its neighborhood. In (Barbagallo et al., 2010), each node sends scouts to other nodes; a scout is a software agent that is in charge of (i) travelling in the infrastructure to collect monitoring information and (ii) sharing this information with its origin node when it comes back, to help the origin node schedule its VMs. These proposals focused on long-term scheduling policies, like consolidation, and were not designed to handle QoS viola-

tions quickly. Moreover, they may lead to a high number of migrations (Barbagallo et al., 2010; Marzolla, Babaoglu, and Panzieri, 2011).

Last but not the least, none of these approaches have been designed to take account of the network topology and therefore cannot manage VMs efficiently in a multi-site deployment.

As claimed in this chapter, an ideal decentralized approach should (i) handle QoS violations quickly, (ii) compute reconfigurations that are close to optimality, (iii) limit the number of migrations and (iv) consider the physical topology and the resilience aspects of the infrastructure.

Conclusion & Open Research Issues

4

The holy grail for Infrastructure-as-a-Service providers is to maximize the utilization of their infrastructure while ensuring the quality of service for the applications they host. Depiste the flexibility brought by system virtualization and all progress in the design of the frameworks in charge of operating VM on pools of physical machines (Lowe, 2010; *OpenNebula* website; *CloudStack* website; *OpenStack* website), most IaaS managers do not efficiently handle the aforementioned objective. Even worse, most production cloud computing infrastructures still leverage static and greedy policies. There are two reasons for this situation. First, manipulating a large number of VMs in an efficient and consistent manner can rapidely become tedious. Second, the implementation of advanced scheduling policies are subject to hard scalability problems, in part due to their centralized design (*i.e.*, the management tasks are performed by a restricted set of dedicated nodes).

In this first part, we studied how to mitigate these concerns.

- In Chapter 2, we described a generalization of the concepts proposed in the Entropy framework (Hermenier, Lorca, et al., 2009). Entropy was an academic proposal developped by the ASCOLA Research group at my arrival. The system acted as an infinite control loop, which performs a globally optimized placement according to cluster resource usage and scheduler objectives. [1] Our work has consisted in proposing the context switch module that transparently handle all VMs manipulations (Hermenier, Lebre, and Menaud, 2010). Given the initial placement and the expected one, the context switch module computes an optimized reconfiguration plan. This plan describes sequences of transitions to perform (i.e. the run, migrate, suspend/resume, stop VM operations) in order to move from the current situation to the new one. As the cost of each action and the dependencies between them is considered, the module reduces, the duration of each VM context switch by performing a minimum number of actions, in the most efficient way. Thanks to this module, developers can focus on the scheduling algorithm in charge of identifying which VMs should be executed. This contribution has tackled the difficulty of manipulating in an efficient and consistent manner a large number of VMs.

- in Chapter 3, we explained how we revised the context switch module in order to address the aforementioned scalability limitation. Our first contribution has consisted in decentralizing the cluster-wide context switch framework, resulting in the DVMS (Distributed Virtual Machine Scheduler) proposal (Quesnel, Lebre, and Südholt, 2013). DVMS is deployed as a set of agents that are organized following a ring topology. Agents cooperate with one another to guarantee that demands of the VMs are satisfied during their executions. A Proof-of-Concept has been implemented and validated by means of ad-hoc simulations (not presented in this manuscript) and with experiments on the Grid'5000 testbed (Quesnel, Lebre, Pastor, et al., 2013). Although, DVMS was very reactive to schedule tens of thousands of VM distributed over thousands of nodes, we observed that the duration of some reconfigurations was much longer than others. Deeper investigations revealed that considering the physical topology of large-scale Cloud Computing platforms is important to handle VM context switches in an efficient manner (it is faster to reconfigure VMs hosted on the

1 http://entropy.gforge.inria.fr.

same rack than between remote ones). This issue becomes even critical in multi-sites infrastructures, which become the norm as discussed in Part IV. In this case, VM context switches can occur WANWide (*i.e.*, at a Wide Area Network scale), degrading significantly performance as well as quality of the service of the whole system. In the second section of this chapter, we have described how it has been possible to reify the physical topology information that were previously hidden by the DVMS overlay. Concretely, we have proposed a new overlay built on top of the Vivaldi coordinates system and a shortest path algorithm. We have illustrated the advantage of this new building block by changing the overlay network in DVMS. Experiments performed on top of Grid'5000 have shown the advantages of this new overlay in the case of DVMS.

In addition to a few perspectives that have been presented for each contribution, these studies has raised two important questions.

- The first question is related to the low adoption of advanced scheduling strategies. Although the scientific community has investigated for almost one decade the use of dynamic and efficient VM scheduling such as the ones discussed in the previous chapters, most production infrastructures continue to leverage static policies.

  One of the important reasons is the contrast between the number of VMs considered to validate academic proposals (up to one thousand in the best case) and the number actually hosted by Infrastructure-as-a-Service platforms (up to tens and hundreds of thousands instances). Such a gap does not enable academic/industry experts to corroborate the relevance of managing VMs dynamically. The issue is that valid assumptions on small infrastructures might become completely erroneous on much larger ones. However, evaluating robustness and performance of advanced mechanisms through large-scale "in vivo" (*i.e.*, real-world) experiments is not straightforward. In addition to disposing of an appropriate testbed, it implies expensive and tedious tasks to build a complete system that handles control, monitoring and reproducibility of the experiments. When investigated, the correctness of most contributions in the management of large-scale IaaS platforms has been proved by means of ad-hoc simulators. Although such analyses contribute to the state of the art, the validation methodology is unsatisfactory to compare, validate and promote new proposals. Like for Grids (Casanova, Legrand, and Quinson, 2008), P2P (Montresor and Jelasity, 2009) and more recently highest levels of cloud systems (Calheiros et al., 2011), the IaaS community needs an open simulation framework for evaluating concerns related to the dynamic management of VMs. Besides allowing to investigate large-scale instances in an efficient and accurate way, this framework should provide adequate analysis tools to make the discussions of results as clear as possible.

- The second question concerns the scalability, reliability and reactivity challenges of all mechanisms that are necessary to operate and use Cloud Computing infrastructures (*OpenNebula* website; *CloudStack* website; *OpenStack* website). Indeed, if we consider current trends about Cloud Computing infrastructures in terms of size (larger and larger) and in terms of usage (cross-federation), every large-scale issues must be addressed as soon as possible to efficiently manage next generation of Cloud Computing platforms. Leveraging a preliminary study we conducted in 2011 (Lebre, Anedda, et al., 2011) and the results presented in Chapter 3, revising system mechanisms with P2P building blocks looks to be an interesting research direction to investigate. The challenges will be related to two aspects: First, P2P algorithms need to be redrafted to take into account the physical topology that is specific to multi-sites infrastructures. This is mandatory, as illustrated with DVMS, to favor intra-site collaborations but should be extended to mitigate expensive inter-sites communications and

to deal with network split brain in an efficient and robust manner. Second, most of the Cloud Computing software stack should be revised to cope with scalability as well as distribution of resources of next generation Cloud Computing platforms.

The first question has been addressed to a certain extent through the works that are presented in Part III. The second one is currently under investigation through several actions that are performed within the framework of the DISCOVERY initiative [2] as described in Part IV.

2 http://beyondtheclouds.github.io.

Part III

VIRTUALIZATION AND SIMULATION TOOLKITS

This part presents a compilation of the following publications (Hirofuchi, Pouilloux, and Lebre, 2015) and (Lebre, Pastor, and Südholt, 2015). The ongoing work that is presented in the conclusion has been recently presented in (NGuyen and Lebre, 2017).

These results have been possible thanks to collaboration with:

- Flavien Quesnel, Phd in the Ascola Research Team (0ct 2010- Feb 2013), Nantes, now Research Engineer at System-X Institute ;
- Lionel Eyraud-Dubois, Researcher in the RealOpt Research Team, Inria, Bordeaux ;
- Jonathan Pastor, Phd in the Ascola Resarch Team (Oct 2012 - Oct 2016), Nantes, now PostDoc at Chicago University ;
- Laurent Pouilloux, Research Engineer Hemera Inria Large Scale Initiative, (Jan 2013 - Sep 2015), Lyon ;
- Takahiro Hirofuchi, Invited Researcher in the ASCOLA Research Team (Jan 2013 - Dec 2013), AIST Japan ;
- Mario Südholt, Professor and Head of the ASCOLA Research Group, IMT Atlantique, Nantes ;
- Frédéric Desprez, Senior researcher and Deputy Scientific Director at Inria, Grenoble ;
- Anthony Simonet, PostDoc Researcher in the Ascola Research Team (Oct 2015-2017), Nantes ;
- Linh Thuy Nguyen, Phd in the Ascola Resarch Team (Dec 2015, defense expected Dec 2018)

I should complete this list with all folks I met during the different SimGrid user-days. In particular, the results presented here could not have been achieved without the support of the project founders and active maintainers of this crazy piece of software :

- Arnaud Legrand, CNRS Researcher and Head of the POLARIS Research Team, Grenoble ;
- Martin Quinson, Professor at the Ecole Normale Supérieur, Rennes ;
- Frédéric Suter, CNRS Researcher at the IN2P3 Computing Center, Lyon.

Between 2013 and 2016, the activities have been mainly supported by the French ANR project SONGS (11-INFRA-13) and the Japaneese JSPS KAKENHI (Grant 25330097). The ongoing activies are supported by the H2020 BigStorage European Training Network (MSCA-ITN-2014-ETN-642963) and the DISCOVERY Inria Project Lab. I underline that I am the principal investigator of the different actions that have been done and that I have been (co)-supervising all the aformentioned PhDs candidates.

Major Results as well as software code are available at: `http://simgrid.gforge.inria.fr/contrib/clouds-sg-doc.php` and `http://beyondtheclouds.github.io/VMPlaceS/`.

Adding virtualization capabilities to SimGrid

5

As real systems become larger and more complex, the use of simulator frameworks grows in our research community. By leveraging them, users can focus on the major aspects of their algorithm, run in-siclo experiments (i.e., *simulations), and thoroughly analyze results without facing the complexity of conducting in-vivo studies* (i.e., *on real testbeds). Nowadays the virtual machine (VM) technology has become a fundamental building block of distributed computing environments, in particular in cloud infrastructures, thus our community needs a full-fledged simulation framework that enables us to investigate large-scale virtualized environments through accurate simulations. To be adopted, such a framework should provides easy-to-use APIs, close to the real ones and preferably fully compatible with those of an existing popular simulation framework.*

In this chapter, we present how we have extended SimGrid, a widely-used open-source simulation toolkit, in order to deliver a highly-scalable and versatile simulation framework supporting VM environments. Our proposal allows users to launch hundreds of thousands of VMs on their simulation programs and control VMs in the same manner as in the real world (e.g., suspend/resume and migrate). Users can execute computation and communication tasks on physical machines (PMs) and VMs through the same SimGrid API, which will provide a seamless migration path to IaaS simulations for hundreds of SimGrid users. Moreover, SimGrid VM includes a live migration model implementing the precopy migration algorithm. This model correctly calculates the migration time as well as the migration traffic, taking into account resource contention caused by other computations and data exchanges within the whole system. This allows user to obtain accurate results of dynamic virtualized systems. We confirmed accuracy of both the VM and the live migration models by conducting several micro-benchmarks under various conditions.

In addition to enabling the development of VMPlaceS as explained in the last section, this work has been important for more recent activities focusing on the estimate of the VM context switch operation (Kherbache, Madelaine, and Hermenier, 2015).

5.1 Challenge Description

Nowadays, VM technology plays one of the key roles in cloud computing environments. Large-scale data centers can manipulate up to one million of VMs, each of them being dynamically created and destroyed according to user requests. Numerous studies on large-scale virtualized environments are being conducted by both academics and industries in order to improve performance and reliability of such systems. However, these studies sometimes involve a potential pitfall; the number of virtual machine (VMs) considered to validate research proposals (up to one thousand in the best case) is far less than the number actually hosted by real Infrastructure-as-a-Service (IaaS) platforms. Such a gap prevents researchers from corroborating the relevance of managing VMs in a more dynamic fashion, since valid assumptions on small infrastructures sometimes become completely erroneous on much larger ones. The reason behind this pitfall is that it is not always possible for researchers to evaluate robustness and performance of cloud computing platforms through large-scale *in vivo* experiments. In addition to the difficulty in obtaining an appropriate testbed, it implies expensive and tedious tasks such as controlling, monitoring and ensuring the reproducibility of the experiments. Hence, correctness of most contributions in the

management of IaaS platforms has been proved by means of ad-hoc simulators and confirmed, when available, with small-scale *in vivo* experiments. Even though such approaches enable our community to make a certain degree of progress, we advocate that leveraging ad-hoc simulators with not so representative *in vivo* experiments is not rigorous enough to compare and validate new proposals

Like for Grids (Casanova, Legrand, and Quinson, 2008), P2P (Montresor and Jelasity, 2009) and more recently highest levels of cloud systems (Calheiros et al., 2011), the IaaS community needs an open simulation framework for evaluating concerns related to the management of VMs. Such a framework should allow investigating very large-scale simulations in an efficient and accurate way as well as it should provide adequate analysis tools to make the discussions of results as clear as possible: By leveraging simulation results, researchers will be able to limit *in vivo* experiments only to the most relevant ones.

Our contribution is SimGrid VM, the first highly-scalable and versatile simulation framework supporting VM environments. We built it upon SimGrid (Casanova, Legrand, and Quinson, 2008) since its relevance in terms of performance and validity has already been demonstrated for many distributed systems. [1]

Our simulation framework allows users to launch hundreds of thousands of VMs on their simulation programs and accurately control VMs in the same manner as in the real world. The framework correctly calculates resource allocation to each computation/communication task, considering VM placement on a simulated system.

In addition to the virtual workstation model, we also integrated a live migration model implementing the precopy migration algorithm. This model correctly calculates the migration time as well as the migration traffic, taking account of resource contention caused by other computations and data exchanges within the whole system. This allows user to obtain accurate results of systems where migrations play a major role. This is an important contribution as several people might erroneously consider that live migrations can be simulated by simply leveraging data transfer models. As discussed in this chapter, several parameters such as the memory update speed of a VM govern live migration operations and should be considered in the model if we want to deliver correct values.

The remaining of the chapter is organized as follow Section 5.2 gives an overview of SimGrid. After summarizing the requirements for the VM support in a simulation framework, Section 5.3 introduces SimGrid VM and explains in details how the virtual workstation and the live migration models have been designed. Evaluation of both models is discussed in Section 5.4. Section 5.5 deals with related work and finally, Section 5.6 concludes and gives some perspectives for SimGrid VM.

5.2 SimGrid Overview

SimGrid is a simulation framework to study the behavior of large-scale distributed systems such as Grids, HPC and P2P systems. There is a large, world-wide user community of SimGrid. The design overview of SimGrid was described in (Casanova, Legrand, and Quinson, 2008). SimGrid is carefully designed to be scalable and extensible. It is possible to run a simulation composed of 2,000,000 processors on a computer with 16GB of memory. It allows running a simulation on arbitrary network topology under dynamic compute and network resource availabilities. It allows users to quickly develop a simulating program through easy-to-use APIs in C and Java.

Users can dynamically create two types of tasks, *i.e.*, computation tasks and communication tasks, in a simulation world. As the simulation clock is going forward, these tasks are being executed, consuming CPU and network resource in the simulation world. Behind the scene, SimGrid formalizes constraint problems (Rossi, Van Beek, and Walsh, 2006) to get a resource share to each task. Even though there is complex

1 http://simgrid.gforge.inria.fr/Publications.php.

resource contention, it can formulate constraint problems expressing such a situation. Until the simulation ends, it repeats formalizing constraint problems, solving them, and then continuing with the next simulation step. As input parameters, users will prepare *platform* files that describe how their simulation environments are organized; for example, they will include the CPU capacity of each host and the network topology of their environments.

Although SimGrid has many features such as model checking, the simulation of MPI applications, and the task scheduling simulation of DAGs (Direct Acyclic Graphs), we limit our description to the fundamental parts, which are directly related to the VM support.

SimGrid is composed of three different layers:

- The MSG layer provides programming APIs for users. In most cases, users develop their simulation programs only using the APIs in this layer. It allows users to create a process on a host and to execute a computation/communication task on it.
- The SIMIX layer is located in the middle of the components. This layer works for the synchronization and scheduling of process executions on a simulation. Roughly, it provides a functionality similar to system calls in the real world, *i.e.,simcall* in the SimGrid terminology. When a process calls a simcall to do an operation (e.g., compute, or send/receive data) in the simulation world, the SIMIX layer converts it to an action in a corresponding simulation model (explained later). Then, the SIMIX layer blocks the execution of the process until the operation is completed in the simulation world. If there is another process that is not yet blocked, the SIMIX layer performs the context switch to another process, converts its operation to another action, and blocks the execution of that process. After all processes are blocked (*i.e.,* the SIMIX layer has converted the operations of all the currently-running processes to actions), the SIMIX layer requests each model to solve constraint problems, which determines the resource share of each action. After all constraint problems are solved, the SIMIX layer sets the simulation clock ahead until at least one action is completed under the determined resource shares. Then, it restarts the execution of the processes of the completed actions. These steps are repeated until a simulation is over.
- The SURF layer is the kernel of SimGrid, where a simulation model of each resource is implemented. A model formulates constraint problems according to requests from the SIMIX layer, and then actually solves them to get the resource share of each action. There are a CPU model and a network model, used for computation and communication tasks, respectively.

In addition to the MSG API, SimGrid provides several abstractions such as the TRACE mechanisms that enable to perform fine post-mortem analysis of the users' simulations with some visualization tools like the Triva/Viva/PajeNG software suite (). SimGrid will produce rich simulation outputs allowing users to perform statistical analysis on simulation results.

5.3 Our proposal: Simgrid VM

The extension of SimGrid to support VM abstractions has been driven by the following requirements:

- Produce accurate results. Our simulation framework requires the capability to accurately determine resource shares on both virtualized and non-virtualized systems, which must take account of VM placement on physical resources as well as the mechanism of virtualization. In other words, if VMs are co-located

on a PM, the system should calculate a correct CPU time to each VM. If a network link is shared with multiple data transfers from/to VMs or PMs, the system needs to assign a correct bandwidth to each transfer.

- Achieve high scalability. SimGrid has been carefully designed to be able to perform large-scale simulations. The VM support on SimGrid must also achieve high scalability for large simulation environments comprising a mix of thousands of VMs and PMs.

After giving an overview of how we introduced the concept of virtual and physical machine layers in its solving engine of constraint problems, we present the live migration model we implemented. Thanks to these extensions, users can now simulate the computation and the communication of virtualized environments as well as investigating mechanisms involving VM live migration operations.

5.3.1 Adding a VM Workstation Model

In the extension for the VM support, we introduced the concept of an abstraction level in the core of SimGrid, *i.e.*, the PM level and VM level. This design enables us to leverage the constraint problem solver of SimGrid also for the VM support. No modification to the solver engine has been required.

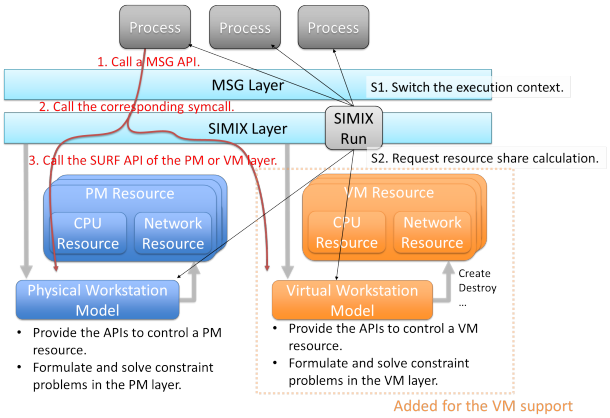


Figure 24: *Design Overview of the VM Support in SimGrid*

During a simulation, SimGrid repeats the following steps: (S1) The SIMIX Run function switches the execution context to each process. (1-2-3) Each process calls a MSG API function, and the corresponding workstation model formulates constraint problems. (S2) The SIMIX Run function requests to solve constraint problems. Then, it updates the states of processes with solved values and sets the simulation clock ahead.

Figure 24 illustrates the design overview of SimGrid with the VM support. We added the virtual workstation model to the SURF layer, and also modified the SIMIX layer to be independent of the physical and virtual workstations. A workstation model is responsible for managing resources in the PM or VM layer. The virtual workstation model inherits most callbacks of the physical one, but implementing VM-specific callbacks. When a process requests to execute a new computation task, the SIMIX layer calls the SURF API of the corresponding workstation model (i.e. depending on where the task is running, the PM or a VM workstation model is used). Then, the target workstation model creates a computation action, and adds a new constraint problem into that layer.

In the SURF layer, the physical workstation model creates PM resource objects for each simulated PM. A PM resource object is composed of a CPU resource object and a network resource object. A CPU resource object has the capability (flop/s, floating operation per second) of the PM. A network resource object corresponds to the network interface of the PM. A VM resource object basically has the same structure as a PM one, including a CPU resource object and a network object. However, a VM resource object has the pointer to the PM resource object where the VM is running. It also has a dummy computation action, which represents the CPU share of the VM in the PM layer (*i.e.*, a variable object X_i in the following). Currently, we mainly focus

on the simulation of CPU and network resources. Disk resource simulation will be integrated in the near future.

As explained in Section 5.2, the SIMIX RUN function executes processes in a simulation in a one-by-one fashion, and then requests each workstation model to calculate resource shares in each machine layer. We modified the SIMIX RUN function to be aware of the machine layers on the simulation system. In theory, it is possible to support the simulation of nested virtualization (*i.e.*, execute a VM at the inside of another VM) by adding another machine layer to the code.

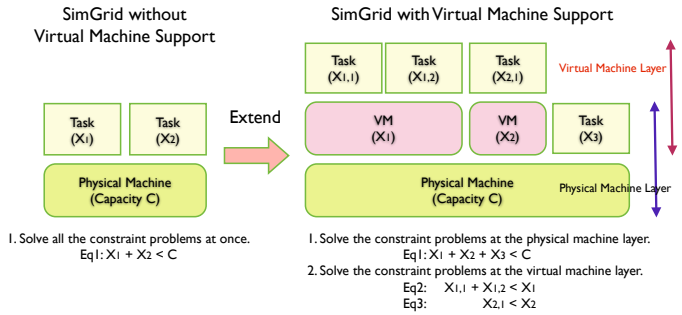


Figure 25: *Resource Share Calculation with VM Support*

C is the CPU capacity (i.e., *the processing speed in flop/s) of a PM. X_i are CPU resource shares assigned to tasks or VMs at the PM layer (from the Host OS viewpoint, a VM is regarded as a task). $X_{i,j}$ are those of the tasks running inside the VM_i.*

The extension of the VM support solves constraint problems with 2 steps. First, the system solves the constraint problems in the PM layer, and obtains the values of how much resource is assigned to each VM (using the corresponding dummy action of each VM). Then, the system solves the constraint problems in the VM layer. From the viewpoint of a PM, a VM is considered as an ordinary task on the PM. From the viewpoint of a task inside a VM, a VM is considered as an ordinary host below the task.

Without the VM support, the solver engine solves all constraint problems on a simulation at once. The left side of Figure 25 shows a simple example where 2 computation tasks are executed on a PM. The PM has a CPU of the capacity C (flop/s). Then, the system formulates a constraint problem,

$$X_1 + X_2 < C \tag{2}$$

where X_1 and X_2 are the CPU shares of each task, respectively. If there are no other conditions, the solver engine assigns 50% of the computation capacity of the PM to each task.

The right side of Figure 25 shows an example with the VM support. A PM has two VMs (VM1 and VM2) and a computation task. VM1 has two computation tasks, and VM2 has a computation task. First, the system formulates a constraint problem at the PM layer.

$$X_1 + X_2 + X_3 < C \tag{3}$$

where X_1, X_2, and X_3 are the CPU shares of VM1, VM2, and the task on the PM. If there are no other conditions, the solver engine assigns 33.3% of the computation capacity of the PM to VM1, VM2 and the task on the PM. Second, the system formulates a constraint problem at the VM layer. Regarding VM1 executing 2 computation tasks, the solver engine makes

$$X_{1,1} + X_{1,2} < X_1 \tag{4}$$

where $X_{1,1}$, $X_{1,2}$ are the CPU shares of the tasks on VM1. In the same manner, for VM2 executing 1 computation task, the solver engine makes

$$X_{2,1} < X_2 \tag{5}$$

where $X_{2,1}$ is the CPU shares of the task on VM2. Thus, if there are no other conditions, each task on VM1 obtains 16.7% of the CPU capacity while the VM2 one obtains 33.3%.

SimGrid allows end-users to set priority to each task. This capability also works for the VM support. As for the above example, if we set 2x larger priority to VM1, VM1 obtains 50% of the computation capacity, and VM2 and the task on the PM get only 25%, respectively. Additionally, we added the capping mechanism of the maximum CPU utilization of each task and VM. We can set the maximum CPU utilization of VM1 to 10% of the capacity of the PM. Even if we remove VM2 and the task on the PM, VM1 cannot obtain more than 10%. These features are useful to take account of virtualization overheads in simulations. In the real world, we can sometimes observe that the performance of a workload is degraded at the inside of a VM. It is possible to simulate this kind of overhead by means of setting priority and capping of tasks and VMs appropriately.

The network resource calculation mechanism with the VM support is implemented in the same manner as the CPU mechanism. The network mechanism considers the resource contention on the PM and also that of shared network links.

5.3.2 Adding a Live Migration Model

Virtual machine monitors (VMMs) supporting live migration of VMs usually implement the precopy algorithm (Clark et al., 2005). At coarse grained, this algorithm transfers all memory pages to the destination PM, before switching the execution host of a VM. Thus, one can erroneously envision that live migration operations can be simulated simply by one network exchange between the source and the destination nodes. However, the pre-copy algorithm is a bit more complex and it is crucial to consider several parameters that clearly govern the time that is mandatory to migrate one VM from one node to another. In this section, first, we describe the successive steps of the precopy algorithm and show that the memory update speed of the VM governs this algorithm by discussing several micro-benchmarks. The design of our live migration model that relies on this preliminary study is finally introduced.

Live Migration Fundamentals

When a live migration is invoked for a particular VM, the VMM performs the following stages:

- Stage 1: Transfer all memory pages of the VM. Note that the guest operating system is still running at the source. Hence, some memory pages can be updated during this first transfer.
- Stage 2: Transfer the memory pages that have been updated during the previous copy phase. Similar to Stage 1, some memory pages will be updated during this second transfer. Hence Stage 2 is iteratively performed until the number of updated memory pages becomes sufficiently small.
- Stage 3: Stop the VM. Transfer the rest of memory pages and other negligibly small states (*e.g.*, those of virtual CPU and devices). Finally, restart the VM on the destination.

Since Stage 3 involves a temporal pause of the VM, the VMM tries to minimize this downtime as much as possible, making it unnoticeable from users and applications. For example, the default maximum downtime of Qemu/KVM, the de-fact Linux VMM (Kivity et al., 2007), is 30 ms. During Stage 2, Qemu/KVM iteratively copies updated memory pages to destination, until the size of remaining memory pages becomes smaller than the threshold value that will achieve the 30 ms downtime. Consequently, a migration time mainly depends on the memory update speed

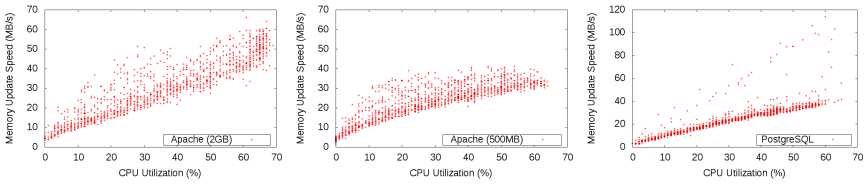
of the VM and the network speed of the migration traffic. A VM intensively updating memory pages will require a longer migration time and in the worst case (*i.e.*, the memory update speed is higher than the network bandwidth), a migration will not finish (*i.e.*„ not *converge* in technical terms). Although libvirt (*libvirt: The virtualization API* n.d.) allows users to set a timeout value for a migration, this mechanism is not enabled in the default settings of Linux distributions.

Impact of The Memory Update Speed

In order to have an idea of the magnitude of the memory update speed in cloud applications, we performed preliminary experiments using representative workloads. This clarifies whether the memory update speed is large enough to be considered as a predominant factor of a live migration model. First, we used a web server workload, and second a database server. We measured how the memory update speed changes in response to the CPU load change of the VM. Each workload runs on the guest OS of a VM.

To measure the memory update speeds, we extended Qemu/KVM to periodically output the current memory update speed of a VM. The VMM has the mechanism to track updated memory pages during a migration, *i.e.*, dirty page tracking. The VMM maintains the bitmap recording updated memory page offsets. With the extension, the VMM enables dirty page tracking: it scans and clears every second the bitmap in order to count up the number of updated pages. It is noteworthy that we did not observe noticeable CPU overhead due to this extension. The recent hardware supports dirty page tracking in the hardware level, and its CPU overhead is substantially small compared to the CPU consumption of a workload.

Web Sever:



(a) *Apache Web Server* (b) *Apache Web Server (cached contents)* (c) *PostgreSQL Database Server*

Figure 26: *The correlation between CPU utilization and memory update speed*

We set up a VM with one VCPU, 1 GB of memory, and one network interface on a PM. The network interface was bridged to the physical network link of GbE. We configured an Apache-2.2 web server on the guest OS of the VM. The Apache server worked in the multi-threads mode, handling each HTTP session with one thread. Another PM was used to launch the *siege* web server benchmark program, [2] which randomly retrieves files available on the web server by performing HTTP get requests. Static web contents had been generated in advance on the guest OS. The size of each file was 100 KB (*i.e.*, a typical size of a web content on the Internet) and the total size of the generated web contents was 2 GB (20K files).

In order to investigate the impact of the page cache mechanism upon the update memory speed, we performed two particular cases:

- For the first case, the benchmark program randomly accessed all 2 GB web contents. Because the RAM size of the VM is 1 GB, accessing more than 1 GB involves I/O accesses since the whole contents cannot be cached by the guest OS. When a requested file is not on the page cache, the guest OS reads the content file from the virtual disk, involving memory updates.

2 http://www.joedog.org/siege-home.

- For the second case, we limited HTTP requests only to 512 MB of the web contents. The corresponding 5000 files had been read on the guest OS, before launching the benchmark program. By caching target files beforehand, we minimized memory updates due to the page cache operation.

For both experiments, we gradually increased the number of concurrent accesses performed by the benchmark program: The number of concurrent sessions was increased by 16 every 60 seconds, up to 512. We measured the memory update speed and the CPU utilization every second. Figures 26a and 26b show the correlation between the CPU utilization level of the VM and its memory update speed. When the number of concurrent sessions increased, the CPU utilization as well as the memory update speed became higher. We expected that the memory update speed of the first case would be significant because of refreshing the page cache, and that of the second case would be small because all file contents were already cached. As shown in Figure 26b, however, in the second case, there exist intensive memory updates (*e.g.*, 30 MB/s at 60% of the CPU utilization), which are not negligible in comparison to the network bandwidth. Considering that the Apache server sends data through zero copy operations (*e.g.*, the use of `sendpage()`, or the combination of `mmap()` and `send()`), this memory update speed results from the pages used by the web server to manage HTTP session (*i.e.*, the heap on the guest OS). The guest OS kernel will also update memory pages for TCP connections, receiving client requests and sending dynamically generated data (*i.e.*, HTTP protocol headers).

Database Server:

The second study focuses on a postgresql-9.2.4 database server. The configuration of the VM was the same as that of the web server experiments. The *pgbench* benchmark [3] was used on the other PM. It emulates the TPC-B benchmark specification (Transaction Processing Performance Council, 1994) that targets database management systems on batch applications, and the back-end database server on market segment. The default setting of the benchmark program aims at measuring the maximum performance of a database server and thus completely saturates the CPU utilization of the VM. To observe the behavior of the VM at various CPU utilization levels, we inserted a 20 ms delay at the end of each sequence of transaction queries.

After starting the experiment, we increased the number of concurrent database sessions by 2 every 60 seconds and up to 70 concurrent sessions. Similarly to the Apache experiments, we can observe on Figure 26c, a clear linear correlation between them. Because every 60 seconds the benchmark program was re-launched with a new concurrency parameter, the sporadic points distant from the majority was caused by establishing new database sessions. As shown in the graph, there exists intensive memory updates of the VM. In this experiment, when the CPU utilization was 60%, the memory update speed reached 40MB/s (30% of the theoretical GbE bandwidth).

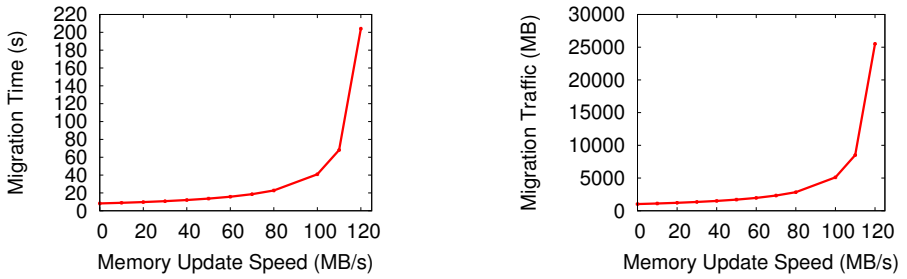
To conclude, although points are scattered in the different graphs, we can observe that a proportional correlation between the CPU usage and the memory update speed exists as a rough trend in the experiments. Such a correlation is important as it will enable to determine the memory update speed of a VM according to its CPU usage.

Summary:

Through the above experiments, we have seen that the memory update speed can be quite significant in comparison with the network bandwidth. Providing a naive model, which simply obtains the cost of a live migration by dividing the VM memory size by the available network bandwidth, is not appropriate as the memory update speed determines the duration of Stage 2 of the precopy algorithm. As an example, the naive model will estimate the migration of the aforementioned database to 8 sec-

3 http://www.postgresql.org/.

onds (*i.e.*, the time required for transferring 1 GB data over the 1 Gbps link) for all situations. This might result in a tremendous gap from the migration performance in the real world once the VM starts to be active: when the database VM is utilizing 60% CPU resource, the live migration time of this VM was approximately 12 seconds, and the transferred data size during the migration reached approximately 1500 MB. This corresponds to 1.5 times the migration cost estimated by the naive model. Figure 27 presents the theoretical estimation of migration cost for a VM (1 GB RAM). The graphs clearly reveal that it is critical to consider the impact of memory updates in order to make an accurate estimation of migration time and generated network traffic.



(a) *Migration Time (s)*

(b) *Migration Traffic (MB)*

Figure 27: *Impact of memory updates on migration cost*

A VM with 1 GB RAM over a 1 Gbps link. The values are theoretically estimated according to the precopy algorithm.

Moreover, the time to perform the first two stages is easily and greatly affected by activities of other VMs and workloads running in the system:

- From the network point of view, the traffic of a live migration can suffer from other network exchanges. For instance, a workload of a VM may create network traffics that competes with migration ones. Similarly, such a competition occurs when multiple live migrations are performed simultaneously. In all these cases, the available network bandwidth for a live migration dynamically changes.
- When several VMs are co-located, they compete with each other for obtaining CPU resources. This contention may lead to performance degradation of workloads. Hence, the memory update speed of the VM dynamically changes due to the activities of other co-located VMs.

A naive model that miscalculates migration times and possibly under- or overestimates the corresponding network traffic will result in erroneous simulation results far from the real world behaviors. To accurately estimate the impact of the memory update speed, the resource sharing among workloads and VMs must be considered. In other words, the simulation framework should consider contention on virtualized and non-virtualized systems: If VMs are co-located on a physical machine, the system should compute a correct CPU time to each VM. If a network link is shared with multiple data transfers, including those of migrations, the system needs to assign a correct bandwidth to each transfer.

Implementation

The live migration model we implemented in SimGrid, performs the same operations of the precopy algorithm described in Section 5.3.2. We denote the memory size of a VM as R. The memory update intensity of the VM is given by a parameter α (bytes/flops). It denotes how much memory pages are marked dirty while one CPU operation (in a simulation world) is executed. We have published a Qemu extension (Qemu-DPT, dirty page tracking) to easily measure the current memory update speed of a VM, which enables SimGrid users to easily determine the memory update intensity of the workloads they want to study. For example, a user launches a VM with Qemu-DPT, starts a workload on the VM, changes a request rate to the workload,

and measures the current CPU utilization level and the memory update speed. Users can easily confirm whether there is a linear correlation and determine the value of correlation coefficient (i.e., the memory update intensity of the workload). If a linear correlation is not appropriate, it is possible to develop another correlation from measured data.

During a live migration, our model repeats data transfer until the end of Stage 3. All transfers are simulated by using the `send`/`recv` operations proposed by the SimGrid MSG API. We define the data size of the ith transfer as X_i bytes. At Stage 1, the precopy algorithm sends all the memory pages, *i.e.*, $X_1 = R$. In Stage 2, the algorithm sends memory pages updated during the previous data transfer. Based on our preliminary experiments, we assume that there will be many situations where the memory update speed is roughly proportional to the CPU utilization level. We use a linear correlation function as the first example of correlation functions. The size of updated memory pages during the data transfer is proportional to the total amount of CPU shares assigned to the VM during this period. Thus, we obtain the data transfer size of the next phase as $X_{i+1} = \min(\alpha S_i, R')$, where S_i (floating operations, flops) is the total amount of CPU shares assigned to the VM during the ith data transfer (as a reminder, S_i is computed by the solver engine as described in Section 5.3.1). R' is the memory size of the working set used by workloads on the VM. The size of updated memory pages never exceeds the working set size.

The simulation framework formulates constraint problems to solve the duration of each data transfer, which is based on the network speed, latency, and other communication tasks sharing the same network link. If the maximum throughput of migration traffic B is given, the model controls the transfer speed of migration data, not to exceed this throughput. This parameter corresponds to `migrate-set-speed` of the libvirt API (*libvirt: The virtualization API* n.d.).

Every time migration data is sent in a simulation world, the model estimates the available bandwidth for the ongoing live migration, in the same manner as the hypervisor does in the real world. The current throughput of migration traffic is estimated by dividing the size of sent data by the time required for sending the data. This estimated value is used to determine the acceptable size of remaining data when migrating from Stage 2 to Stage 3. If the maximum downtime d is 30 ms (*i.e.*, the default value of Qemu), and if the migration bandwidth is estimated at 1 Gbps, the remaining data size must be less than 3,750 KB in Stage 3. The migration mechanism repeats the iteration of Stage 2 until this condition is met.

Finally, at Stage 3, our model creates the communication task of X_n bytes to transfer the rest of memory pages. After this final task ends, the system switches the execution host of the VM to the destination.

Since a linear correlation will explain the memory update speed in typical situations, we implemented it as a default correlation function. However, the proportional function used in the current implementation is just an example of correlation functions to estimate the memory update speed of a VM. If this proportional function is not appropriate for a situation, it is possible to define another correlation function.

The correlation function of the memory update speed can be defined not only with the CPU utilization level but also with any other variables available in the simulation world. For example, a request rate of an application can be used, if it explains the memory update speed of a VM. We consider that in most cases a user will need to modify only one function `get_update_size()` in the VM extension, which calculates the size of updated memory pages.

5.3.3 SimGrid VM API (C and Java)

In our work, we extended the MSG programming API in order to manipulate a VM resource object as shown in Table 6. Each operation in this API corresponds to a real-world VM operation such as create/destroy, start/shutdown, resume/suspend and migrate. We newly defined the `msg_vm_t` structure, which is a VM object in a

`msg_vm_t MSG_VM_create(msg_host_t pm, ...)`	Create a VM object on the given PM with the specified parameters.
`void MSG_VM_destroy(msg_vm_t vm)`	Destroy the VM.
`void MSG_VM_start(msg_vm_t vm)`	Start the VM.
`void MSG_VM_shutdown (msg_vm_t vm)`	Shutdown the VM.
`void MSG_VM_migrate(msg_vm_t vm, msg_host_t dst_pm)`	Migrate the VM to the given destination PM.
`void MSG_VM_set_params(msg_vm_t vm, ws_params_t params)`	Set parameters of the VM.
`vm_state_t MSG_VM_get_state(msg_vm_t vm)`	Return the state of the VM.
`msg_host_t void MSG_VM_get_pm(msg_vm_t vm)`	Return the PM of the VM.
`void MSG_VM_suspend(msg_vm_t vm)`	Suspend the execution of the VM. Keep VM states on memory.
`void MSG_VM_resume(msg_vm_t vm)`	Resume the execution of the VM.
`void MSG_VM_save(msg_vm_t vm)`	Suspend the execution of the VM. Save VM states to storage.
`void MSG_VM_restore(msg_vm_t vm)`	Restore the execution of the VM from storage.
`void MSG_VM_set_bound(msg_vm_t vm, double bound)`	Set the maximum CPU utilization level of the VM.
`void MSG_VM_set_affinity(msg_vm_t vm, unsigned long mask)`	Set the CPU-core affinity of the VM.

Table 6: *The APIs to manipulate a VM resource object in the VM support*

simulation world, supporting these VM APIs. It should be noted that a VM object inherits all features from a PM object `msg_host_t`. A VM resource object supports most existing operations of a PM, such as task creation and execution. From the viewpoint of users, they can treat a VM as an ordinary host, except a VM supports these VM-specific operations.

As discussed in Section 5.3.2, users need to specify the memory size of a VM, the memory update intensity of the VM, and the memory size of the working set of memory used by a workload on the VM. These parameters are specified either at the VM creation or through the `MSG_VM_set_params()` function.

Figure 28 shows an example code using the VM APIs. `example()` starts a VM on the given PM, and launches a worker process on the VM. The way of launching a process is exactly the same as that of a PM; we can use `MSG_process_create()` also for a VM.

Although this example is in `C`, it is noteworthy that the JAVA SimGrid API has been also extended. Hence, end-users can develop their simulators either by interacting with the native C routines or by using the JAVA bindings.

Finally, we highlight that we also extended the multicore support of SimGrid to allow the simulation of virtualized systems running on multicore servers. The `set_affinity` function pins the execution of a VM (or a task) on given CPU cores.

5.4 Validation

In order to confirm the correctness and the accuracy of the VM extensions within SimGrid, we conducted several micro benchmark experiments on Grid'5000 and compared results with the simulated ones. We discuss in this section major ones.

Figure 28: *An Example Code Using the VM APIs*

```
/* the main function of the worker process */
static int worker_main(int argc, char **argv)
{
    /* computation size (floating operations) */
    const double flops = 10000;

    /* Repeat computation. */
    for (;;) {
        msg_task_t task = MSG_task_create(``Task'', flops, 0, NULL
    );
        MSG_task_execute(task);
        MSG_task_destroy(task);
    }
    return 0;
}

void example(msg_host_t pm)
{
    unsigned long ramsize = 2UL * 1024 * 1024; // 2 GB
    double memory_update_intensity = 60; // 60 MB/s at 100 % CPU
    load
    double working_set_size = 0.9; // 90 % of ramsize

    /* 0. Create a VM (named VM0) on the PM. */
    msg_vm_t vm = MSG_vm_create(pm, ``VM0'', ramsize,
    mem_update_intensity, working_set_size);
    MSG_vm_start(vm);

    /* 1. Launch a process on the VM. */
    msg_process_t pr = MSG_process_create(``worker'', worker_main,
     NULL, vm);
    /* 2. Keep the VM running for 10 seconds. */
    MSG_process_sleep(10);
    /* 3. Suspend the VM for 10 seconds. */
    MSG_vm_suspend(vm);
    MSG_process_sleep(10);
    MSG_vm_resume(vm);
    /* 4. Keep the VM running for 10 seconds. */
    MSG_process_sleep(10);
   /* 5. Clean up. */
    MSG_process_kill(pr);
    MSG_vm_shutdown(vm);
    MSG_vm_destroy(vm);
}
```

5.4.1 Experimental Conditions

In this paragraph, we give details that will enable to reproduce the experiments discussed in the next sections.

All experiments in the real world were conducted on the Grid'5000 Graphene cluster. Each PM has one Intel Xeon X3440 (4 CPU cores), 16 GB memory, and a GbE NIC. The hardware virtualization mechanism (*i.e.*, Intel VT) was enabled.

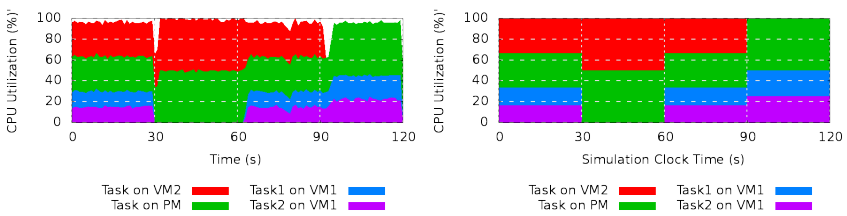
We used Qemu/KVM (Qemu-1.5 and Linux-3.2) for the hypervisor in the experiments. Assuming long-lived active VMs instead of idle VMs that never became active after being booted, we modified a few lines of source code of Qemu to disable the mechanism not to transfer zero-filled pages. This mechanism does not effectively work if the VM is running for a while with active workloads. In such case, non-zero data already exists in most memory pages. Moreover, Qemu-1.5 also supports the XBRLE compression of migration data (Svard et al., 2011). This mechanism, which is disabled in the default settings of major Linux distributions, enables to pick up updated regions of the pages and send them with compression (even though a memory page is marked as dirty, only a few bytes in the page may have been updated, thus selecting only the updated data enables to reduce the migration traffic). Although, it is possible to extend SimGrid to simulate the behaviors of these compression mechanisms, we choose to focus our study on the development of a sound model that can capture common behaviors among hypervisors, and to not focus on implementation-specific details of a particular hypervisor. Hence, we kept this mechanism disabled. The virtual disk of a migrating VM is shared between source and destination physical machines by a NFS server. This is a widely-used storage configuration in cloud computing platforms. To cover more advanced configurations, we are extending our model to support virtual disks and understand the impact of I/O intensive workloads. This effort enables us to simulate the relocation of the associated VM images, which will be reported in our future work.

We used the Execo automatic deployment engine (Imbert et al., 2013) to describe and perform the experiments by using its Python API. According to the scenario of an experiment, the Execo deployment engine automatically reserves, install and configure nodes and network resources that are mandatory before invoking the scripts of the experiment. This mechanism allows us to easily run and reproduce our experiments. All experiments were repeated at least 5 times with no noticeable deviations in obtained results. Then, the same experiments were also conducted on SimGrid.

Although the VM extension of SimGrid supports multicore CPU simulation, in the following micro benchmarks, VMs were pinned to the first CPU core both in real-world and simulation experiments, so as to carefully discuss how resource contention impacts on live migration performance.

Finally, in order to carefully investigate live migration behaviors, we developed a memory update program, `memtouch`, emulating various load conditions by real applications. The `memtouch` program works as a workload that has a linear correlation between CPU utilization levels and memory update speeds. It produces the memory update speed at a given CPU utilization level, by interleaving busy loops, memory updates and micro sleeps in an appropriate ratio as explained in our previous work (Hirofuchi, Nakada, et al., 2012).

The memory update program accepts two kinds of parameters. One is a target CPU utilization level (%). The other is a memory update intensity value that characterizes an application. For example, we observed that the database workload in Section 5.3.2 had a linear correlation between memory update speeds (Y) and CPU utilization levels (X), which was $Y = \alpha X$. This α is the key parameter to model the memory update behavior of this database workload. From Figure 26c, we can roughly estimate the memory update intensity α to be *60MB/s at CPU 100%*. This 60MB/s is passed to the arguments of the memory update program. If a given target CPU utilization level is 50%, the memory update speed of the program becomes 30MB/s. Moreover, if other workloads or co-located VMs compete for CPU resource and the memory



(a) *Grid'5000* (b) *Simulation*

Figure 29: *The CPU load of each task in a CPU resource contention experiment*

In each graph, from the top to the bottom, the CPU usages of Task on VM2, Task on PM, Task1 on VM1, and Task2 on VM1, are illustrated, respectively.

update program only gets 25%, the actual memory update speed becomes 15MB/s. This behavior correctly emulates what happens in consolidated situations.

5.4.2 Evaluation of the VM Workstation Model

The CPU resource allocation of a VM impacts on is memory update speed, and the memory update speed is a dominant parameter that governs live migration time. Before doing the experiments focusing on live migration, we confirmed that our VM support of SimGrid correctly calculates CPU and network resource allocations for VMs.

We launched 2 VMs and 4 computation tasks with the same arrangement as the right side of Figure 25: 2 VMs (VM1 and VM2) are launched on a PM. VM1 has 2 computation tasks, and VM2 has 1 computation task. The PM also has a computation task. All tasks tried to obtain the 100% CPU utilization, competing with each other. We used the memory update intensity of the database benchmark as discussed in the above (*i.e.*, 60MB/s at the 100% CPU utilization).

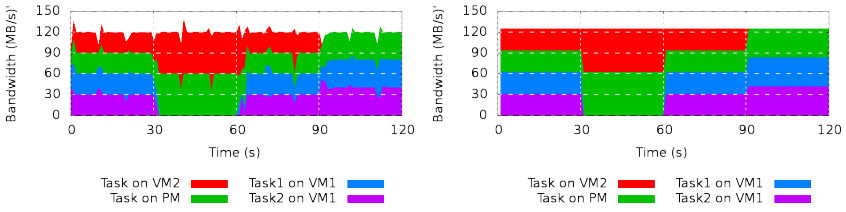
Figure 29a shows the CPU utilization level of each task in a real-world experiment on Grid'5000. First, VM1, VM2, and the task on the PM fairly shared the CPU resource of the PM, consuming approximately 33.3% respectively. On VM1, each task consumed approximately 50% of the CPU share assigned to VM1. At 30 seconds, we suspended VM1 running the 2 tasks. Thus, the task on VM2 and the task on the PM consumed approximately 50%, respectively. At 60 seconds, we resumed VM1. The CPU utilization of each task was recovered as the initial state. At 90 seconds, we shut down VM2. Then, the CPU share of VM1 and the task on the PM increased to approximately 50%, respectively. These results were reasonable, considering the hypervisor fairly assigns CPU resources to each VM and the task on the PM.

The same experiment was performed in simulation. As shown in Figure 29b, the VM support of SimGrid correctly captured the CPU load change of each task in large part. However, there are minor differences between the real-world and simulation experiments, especially just after a VM operation (i.e., suspend/resume and shutdown) was invoked. For example, the shutdown of VM2 took approximately 3 seconds to be completed. When the suspension of a VM is invoked, the guest OS stops all the processes on it, and then commits all pending write operations to virtual disks. In the real world, these operations will sometimes have an impact on other tasks on the same PM. We consider that if a user needs to simulate further detail of VM behaviors, it is possible to add the overhead of VM operations into the VM model. As mentioned earlier, we are working for example on modeling VM boot and snapshotting costs related to I/O accesses to the VM images.

The second experiment we conducted aimed at observing the behavior of VMs under network resource contention. Because live migration time is impacted also by network resource availability, the VM support of SimGrid needs to correctly calculate network resource assignments to each communication task. Although a former study regarding SimGrid (Casanova, Legrand, and Quinson, 2008) already proved the correctness of its network model, the study had been done when our VM support was not available. It is necessary to confirm the network model also correctly works at the VM level.

We launched 2 VMs and 4 communication tasks with the same arrangement as the previous experiment. However, in this experiment, 4 communication tasks were launched instead of computation tasks. Each communication task continued to send data to another PM. The destination PM of each communication task was different, respectively.

Figure 30a shows the result of a real-world experiment. `iperf` was used to send data to destination. The host operating system fairly assigned network resources to each TCP connection. For example, while VM1 was suspended, the 2 communication tasks on it could not send data, and the bandwidth of the 1Gbps link was split into 2 other TCP connections. As shown in Figure 30b, the VM support of SimGrid correctly captured this behavior.



(a) *Grid'5000* (b) *Simulation*

Figure 30: *The network throughput of each task in a network resource contention experiment*

In each graph, from the top to the bottom, the network throughput of Task on VM2, Task on PM, Task1 on VM1, and Task2 on VM1, are illustrated, respectively.

It should be noted that, in the real world experiment, the packet queuing discipline of the physical network interface of the source PM needed to be set to SFQ (Stochastic Fairness Queuing), which enforces bandwidth fairness among TCP connections more strongly than the default one of Linux (i.e., basically, first in first out). Although the TCP algorithm is designed to achieve fairness among connections, we experienced that, without using SFQ, the network traffic made by the task running on the PM occupied more than 90% of the available bandwidth of the network interface. We consider that this problem was caused by the difference of packet flow paths between the task on the PM and the other tasks running on VMs. Although the latest libvirt code on its development repository uses SFQ as we did, libvirt-0.9.12[4] used in the experiments did not.

In summary, through these experiments, we confirmed that the VM support of SimGrid correctly calculates CPU and network resource assignments to VMs and tasks. The prerequisite to obtain sound simulation results of live migrations is met.

5.4.3 Live migrations with various CPU levels

We conducted a series of experiments to confirm the correctness of the live migration model. The settings of VMs were the same as the above experiments. The memory update intensity was set to that of the database benchmark (*i.e.*, 60MB/s at the 100% CPU utilization). The memory size of a VM was 2GB. The size of the working set memory was set to its 90%.

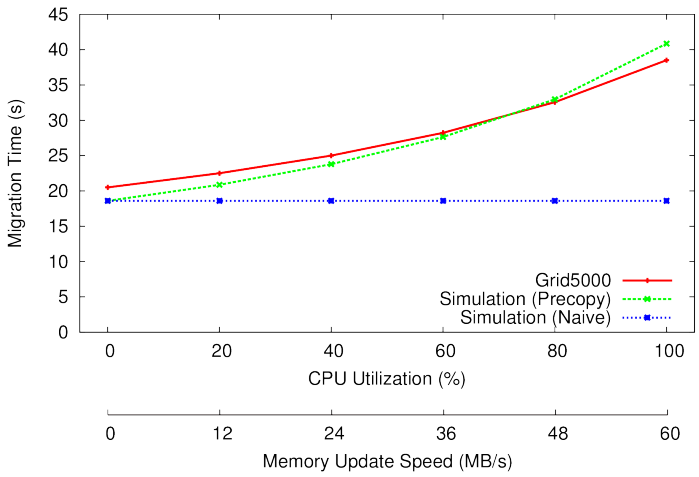
Figure 31 shows live migration times with different CPU utilization levels. This graph compares the real experiments on Grid'5000, the simulation experiments using our migration model, and the simulation experiments with the naive migration model (*i.e.*, without considering memory updates). These results confirm the preliminary investigations performed in Section 5.3.2: As the CPU utilization level of the VM increased, we observed that the live migration time of the VM became longer. Our simulation framework implementing the precopy algorithm successfully simulated this upward curve, within 2 seconds deviations (9% at most).

On the other hand, the naive simulation without the precopy algorithm failed to calculate correct migration times, especially in the higher CPU utilization levels. At the CPU 100% case, the naive simulation underestimated the migration time by 50%.

4 It is included the latest stable release (wheezy) of the Debian/GNU distribution.

Figure 31: *Comparison of live migration time*

The X axis shows CPU utilization levels and corresponding memory update speeds. Update memory speed of the workload is 60 MB/s at 100% of CPU.



Simulation frameworks without the migration model, *e.g.*, CloudSim, will produce these results.

The small differences between the Grid'5000 results and our precopy model, *e.g.*, up to 2 seconds in Figures 31, can be explained by the fact that in addition to the `memtouch` program, other programs and the guest kernel itself are consuming CPU cycles and are slightly updating memory pages of the VM in the reality. Furthermore, because the system clock on the guest OS is less accurate than that of the host OS, the memory update speed by the program involves small deviations from the target speed. We are going to show that this deviation might be problematic in a few corner cases where memory update speed and available migration bandwidth are close to each other.

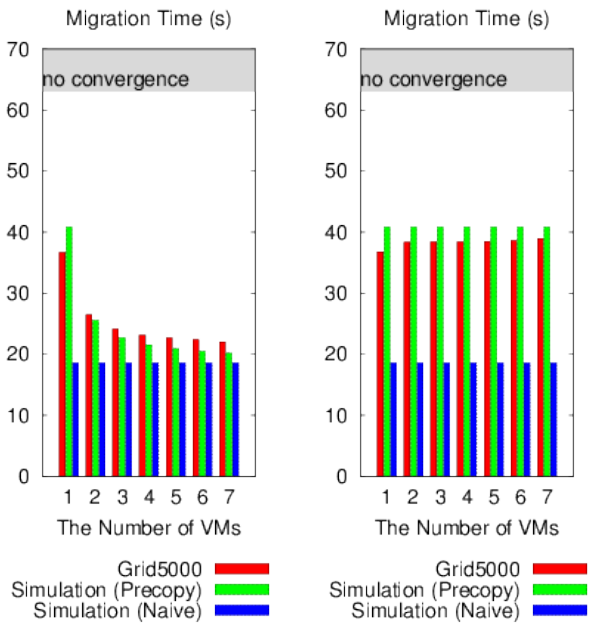
5.4.4 Live Migrations under CPU Contention

A live migration is deeply impacted by CPU resource contention on the source and destination PMs. We performed migration experiments with the different numbers of co-located VMs. In addition to the VM to be migrated, other VMs were launched on the source or destination PM. To discuss serious resource contention, sometimes found dynamic VM packing systems, all the VMs were pinned to the first physical CPU core of the PM. The `cpuset` feature of libvirt (*libvirt: The virtualization API* n.d.) was used in real-world experiments, and `MSG_vm_set_affinity()` was called in simulation programs. All the VMs executed the `memtouch` program on their guest OSes. The target CPU utilization of `memtouch` was set to 100%; although all the VM on the PM tried to obtain 100% CPU resource, they actually obtained partial CPU resource due to co-location. Qemu/KVM will fairly assign CPU resource to each VM. [5]

Figure 32a shows live migration times when other VMs were launched on the source PM. When the number of co-located VMs was higher, the actual live migration times decreased, becoming close to that of no memory update case (*i.e.*, approximately 20 seconds). This serious CPU resource contention reduced the actual memory update speed of the migrating VM, which results in shorter migration times. Our simulation framework correctly calculated assigned CPU resource and captured migration behavior.

Figure 32b is the case where the other VMs were launched on the destination PM. Because the migrating VM, updating memory pages on the source PM, did not compete with the other VMs for CPU resource, the live migration times were not affected

5 It should be noted that even if multiple VMs share one PM, the number of memory pages associated to each VM does not change. The hypervisor assigns 2 GB of memory pages to each VM.



(a) *SRC* (b) *DST*

Figure 32: *Live migration time with resource contention*

In Figures 32a and 32b, the number of co-located VMs is changed, on the source and destination PM, respectively.

by the number of other VMs. This behavior was successfully reproduced by our simulation framework with the precopy model.

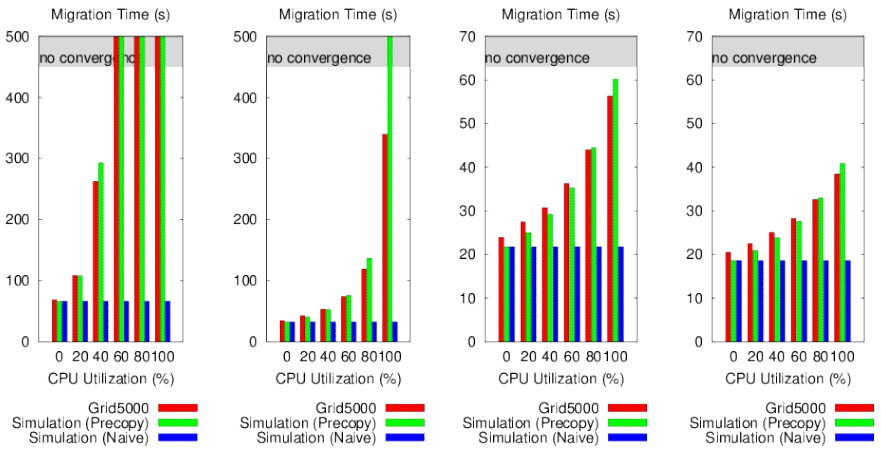
When there were other VMs on the destination PM, the migration times were slightly longer than that of the 1-VM case on Grid'5000. During a live migration, the hypervisor launches a dummy VM process on the destination PM. The dummy VM process is receiving migration data from the source PM. Although this receiving operation requires very little CPU resource, the data receiving speed was slightly reduced due to the CPU resource contention on the destination PM. We consider that CPU overheads of sending/receiving migration data are negligible in most use-cases because they are substantially small as compared to resource usage by VM workloads. However, if a user of our simulation framework needs to carefully simulate such behaviors, it is possible to create micro computation tasks corresponding to data sending/receiving overheads by leveraging the SimGrid MSG API. In addition, as discussed in Section 5.4.3, there were also small deviations between the results of Grid'5000 and the simulation. If these deviations are not negligible, it is also possible to improve the simulation mechanism by taking into account CPU cycle consumption and memory update by the guest kernel and other processes.

5.4.5 Live Migrations under Network Contention

A live migration time is also deeply impacted by the available network bandwidth for the migration. The hypervisor uses a TCP connection to transfer migration data of a VM. It should be noted that from the viewpoint of the host OS this is a normal TCP connection opened by a userland process (i.e., the Qemu process of a VM). On the host OS and the network link, there is no difference between the TCP connection of a live migration and other TCP connections (including NFS sessions used for disk I/O of VMs). We limited the available bandwidth for a migration to 25%, 50%, 75% and 100% of the GbE throughput. The virsh `migrate-set-speed` command was used. These experiments are intended to correspond to real-world situations where concurrent live migrations are performed at once or live migration traffic is affected by other background traffic. Because disk I/O of a VM creates NFS requests, these experiments also cover situations where disk I/O of VMs impact live migration traffic.

Figure 33: *Live migration time with resource contention (cont)*

In Figures 33a-33d, the available bandwidth is limited, i.e., 25%, 50%, 75%, and 100% of GbE, respectively. The results on the gray zone (no convergence) mean the cases where migrations never finished. Note that the scale of the Y axis is different. Update memory speed of the workload is 60 MB/s at 100% of CPU.



(a) *Bw. 25%* (b) *Bw. 50%* (c) *Bw. 75%* (d) *Bw. 100%*

Figures 33a-33d compare real and simulated migration times. When the available bandwidth was smaller, the simulation with the naive model underestimated live migration times more seriously. The real migration times exponentially increased, as the available bandwidth was smaller. The precopy live migration model correctly simulated these trends in most cases. If the available bandwidth is smaller than the actual memory update speed of the VM, the algorithm of the precopy live migration never finished. The iterative memory copy phase of the precopy live migration (i.e., Stage 2 in Section 5.3.2) continues until someone cancels (i.e., gives up) the live migration. This behavior was correctly simulated at the 60%, 80% and 100% CPU utilization levels of the 25% GbE bandwidth case (see Figure 33a).

The only exceptional case where the precopy model did not follow the results of real experiments, was at the 100% CPU utilization level of the 50% GbE bandwidth case (see Figure 33b). The simulation using the precopy model predicted this live migration never finished, although the live migration on Grid'5000 finished in 330 seconds. In this condition, the migration bandwidth was 50% of 1 Gbps (*i.e.*, 59.6MB/s), while the memory update speed was 60MB/s. Because the migration bandwidth is theoretically smaller than the memory update speed, the precopy migration model iterated Stage 2 of the precopy algorithm forever. On the other hand, as discussed in Section 5.4.3, since the actual memory update speeds of the VM would be slightly below the migration bandwidth, the real live migration finished in a finite period of time.

We consider that this problem will not appear in most cases. However, if a user needs to simulate such an exceptional situation where the migration bandwidth and the memory update speed of a VM are very close, it is necessary to give careful consideration to the accuracy of these simulation parameters. A simulation program may need to consider subtle memory updates by the guest kernel and the network bandwidth fluctuation caused by other background traffic.

To conclude, we can affirm that our experiments were accurate enough to validate our extensions in most cases.

5.5 Related Work

CloudSim (Calheiros et al., 2011) is a simulation framework that allows users to develop simulation programs of cloud datacenters. It has been used, for example, for studies regarding dynamic consolidation and resource provisioning. Although it looks that CloudSim shares the same goal with our SimGrid project, the current API of CloudSim is based on a relatively top-down viewpoint of cloud environments. Their API provides a perspective of datacenters composed of application services and

virtual and physical hosts. Users will create pseudo datacenter objects in their simulation programs. In their publication, a migration time is calculated by dividing a VM memory size by a network bandwidth. [6] This model, however, cannot correctly simulate many real environments where workloads perform substantial memory writes. On the other hand, we can say that SimGrid takes a bottom-up approach. Our ongoing project currently pays great attention to carefully simulate the behavior of a VM running in various conditions, which leads to well-fitting simulation results of cloud environments where many VMs are concurrently running with various workloads. As far as we know, our virtualization support of SimGrid is the first simulation framework that implements a live migration model of the precopy algorithm. It will provide sound simulation results for dynamic consolidation studies.

iCanCloud (Núñez et al., 2012) is a simulation framework with the job dispatch model of a virtualized datacenter. Their hypervisor module is composed of a job scheduler, waiting/running/finished job queues, and a set of VMs. For the use of commercial cloud services like Amazon EC2, there is trade-off between financial cost and application performance. This framework was used to simulate provisioning and scheduling algorithms with the cost-per-performance metric. Koala (Mills, Filliben, and Dabrowski, 2011) is a discrete-event simulator emulating the Amazon EC2 interface. It extends the cloud management framework Eucalyptus (Nurmi et al., 2009), modeling its cloud/cluster/node controllers. VM placement algorithms were compared using this simulator. These frameworks are designed to study a higher-level perspective of cloud datacenters, such as resource provisioning, scheduling and energy saving. Contrary, SimGrid, originally designed for the simulation of distributed systems, performs computation and data sending/receiving in the simulation world. It simulates more fine-grained system behavior, which is necessary to thoroughly analyze cloud environments.

GreenCloud (Kliazovich et al., 2010) is a simulator extending a network simulator NS2. It allows simulating energy consumption of a datacenter, considering workload distributions and network topology. This simulator is intended to capture communication details with their packet-level simulation. SimGrid does not simulate distributed system in the packet level, but in the communication flow basis. It is designed to simulate distributed systems, composed of computation and communication, in a scalable manner. The SONGS project is also working on integrate energy models to SimGrid, which enables energy consumption simulations of datacenters.

We consider that it would be possible to implement a precopy live migration model on these simulation toolkits. However, substantial extension of existing code may be necessary. In order to correctly simulate migration behaviors of VMs, a simulation toolkit requires the mechanism modeling a live migration algorithm and the mechanism to correctly calculate resource share of VMs.

SimGrid supports a formal verification mechanism for distributed systems, a simulation mechanism of parallel task scheduling with DAG (direct acyclic graphs) models, and a simulation mechanism of unmodified MPI applications. The VM extension of SimGrid enables researchers to use these mechanisms also for virtualized environments. We have carefully designed the VM extension to be compatible with existing components in SimGrid.

5.6 Summary

We developed a scalable, versatile, and easy-to-use simulation framework supporting virtualized distributed environments, which is based on a widely-used, open-source

6 In the source code of the 3.0.3 release, the power-aware datacenter model of CloudSim (PowerDataceter Class) assumes that the half network bandwidth of a target host is always used for a migration because they assume the other half is used for other VM communications. According to CloudSim FAQ, for those who directly program migrations in simulations, CloudSim provides an API to invoke a migration. However, users need to assign an estimated completion time to each migration. It does not have a mechanism to accurately estimate the completion time.

simulation framework, SimGrid. The extension to SimGrid is seamlessly integrated with existing components of the simulation framework. Users can easily develop simulation programs comprising VMs and PMs through the same SimGrid API. We redesigned the constraint problem solver of SimGrid to support resource share calculation of VMs, and developed a live migration model implementing the precopy migration algorithm of Qemu/KVM. Through micro benchmarks, although we observed that a few corner cases cannot be easily simulated when the memory update speed is closed to the network bandwidth, we confirmed that our precopy live migration model reproduced sound simulations results in most cases. In addition, we showed that a naive migration model, not considering memory updates of the migrating VM nor resource sharing competition, underestimated the live migration time as well as the resulting network traffic. As future work, it would be valuable to complete this model of the precopu algorithm in order to take into account the migration of the persistent states of a VM (*a.k.a.*, the VM image). While VMs generally rely on remote attached volumes within one site, considering the cost of migrating the VM image is mandatory to simulate WANWide migration in an accurate manner.

Regarding the scalability of our extensions, we succeeded to perform sample simulations up to 4K PMs and 25K VMs. However, we discovered that the scalability of our extensions is exponential and not proportional to the number of PMs/VMs. We are working with the SimGrid core developers to investigate how the performances of our extensions can be improved. The first envisioned approach is to use co-routines instead of the pthread library.

The current code of SimGrid, including all the aforementioned VM extensions, is available at: `http://simgrid.gforge.inria.fr`.

To conclude this chapter, I would like to underline a few points:

- While a first use-case is presented in the original paper (Hirofuchi, Pouilloux, and Lebre, 2015) in order to illustrate the advantage of SimGrid VM, I chose to not discuss here but rather to deal with in the next chapter. Indeed, this first use-case enabled us to validate the accuracy of the VMPlaceS proposal that is described in the following.

- Our contribution regarding the live-migration model enabled to make progress on the estimation of the duration of a VMs context switch operation (Kherbache, Madelaine, and Hermenier, 2015) introduced in Chapter 2.

- I have recently started a new activity that aims to extend the presented extensions with a model for the boot operation of a VM (NGuyen and Lebre, 2017). This point is further discussed in Chapter 7.

6 Virtual Machine Placement Simulator

Advanced Virtual Machines placement policies such as the ones presented in Part II have been evaluated either using limited scale in-vivo experiments or ad hoc simulator techniques. These validation methodologies are unsatisfactory. First they do not model precisely enough real production platforms (size, workload representativeness ...). Second, they do not enable the fair comparison of different approaches.

In this chapter, we present VMPlaceS, a dedicated simulation framework to perform in-depth investigations and fair comparisons of VM placement algorithms. VMPlaceS enables researchers (i) to study and compare VM placement algorithms , (ii) to detect possible limitations at large scale and (iii) easily investigate different design choices. Built on top of the SimGrid VM extensions presented in the previous chapter, VMPlaceS provides programming support to ease the implementation of placement algorithms and runtime support dedicated to load injection and execution trace analysis. It supports a large set of parameters enabling researchers to design simulations representative of a large space of real-world scenarios.

To illustrate its relevance, we have implemented and analyzed three well-known approaches: Entropy (Hermenier, Lorca, et al., 2009), Snooze (Feller, Rilling, and Morin, 2012), and DVMS (Quesnel, Lebre, and Südholt, 2013). We have chosen these three systems as they represent three classes of placement algorithms: Entropy is an instance of a centralized model, Snooze of a hierarchical one and DVMS of a fully distributed one. Using VMPlaceS, we compare the scalability and reactivity (i.e., *the time to solve SLA violations) of the strategies — a contribution of its own. We believe that VMPlaceS will allow researchers to validate the significant benefits of new placement algorithms, thus accelerating VM placement research results and favouring the transfer to IaaS production platforms.*

6.1 Challenge Description

Even if more flexible and often more efficient approaches to the Virtual Machine Placement Problem (VMPP) have been developed, most of the popular Cloud Computing management systems (*CloudStack* website; *OpenNebula* website; *OpenStack* website), or IaaS toolkits (Moreno-Vozmediano, Rubén S Montero, and Ignacio M Llorente, 2012), continue to rely on elementary Virtual Machine (VM) placement policies that prevent them from maximizing the usage of CC resources while guaranteeing VM resource requirements as defined by Service Level Agreements (SLAs).

An important impediment to the adoption of more advanced strategies such as dynamic consolidation, load balancing and other SLA-enforcing algorithms developed by the academic community (Hermenier, Lorca, et al., 2009; Feller, Rilling, and Morin, 2012; Quesnel, Lebre, and Südholt, 2013) is related to the experimental processes used for their validation: most VM placement proposals have been evaluated either using ad hoc simulators or small *in-vivo* (*i.e.*, real-world) experiments. These methods are not accurate and not representative enough to (i) ensure their correctness on real platforms and (ii) perform fair comparisons between them.

Implementing each proposal and evaluating it on representative testbeds in terms of scalability, reliability and varying workload changes would definitely be the most rigorous way to observe and propose appropriate solutions for Cloud Computing production infrastructures. However, *in-vivo* experiments, if they can be executed at all, are always expensive and tedious to perform (for recent reference see (Barker et

al., 2014)). They may even be counterproductive if the observed behaviors are clearly different from the expected ones. Consequently, new placement algorithms are continuously proposed without really identifying the significant benefits of each of them.

To address this problem, we proposed VMPlaceS, a dedicated simulation framework to perform in-depth investigations of VM placement algorithms and compare them in a fair way. To cope with real conditions such as the increasing scale of modern data centers, as well as the workload dynamicity and elasticity characteristics that are specific to the Cloud Computing paradigm, VMPlaceS allows users to study large-scale scenarios that involve thousands of VMs, each executing a specific workload that evolves during the simulation.

This chapter discusses the VMPlaceS proposal. The presentation of SimGrid, which is presented in the original paper (Lebre, Pastor, and Südholt, 2015), has been removed for consistency reasons. We invite readers to refer to the previous chapter for further information. In Section 6.2, we describe the VMPlaceS proposal, and its general functioning. A Brief overview of the three approaches we analyzed is given in Section 6.3. This part might look like redundant as the Entropy and the DVMS proposals have been largely discussed in the previous chapter. However, I chose to keep it with the objective of making the read of the whole chapter more straightforward. Section **??** discusses an experiment that proved the satisfying accuracy of VMPlaceS. Results of simulations are discussed in Section 6.4. In particular, they revealed the importance of the duration of the reconfiguration phase (*i.e.*, the step where VMs are relocated throughout the infrastructure) compared to the computation phase (*i.e.*, the step where the scheduler solves the placement problem). Section 6.5 deal with related work. Finally, Section 6.6 present, respectively, related work and a conclusion.

6.2 Our Proposal: VMPlaceS

The aim of VMPlaceS is twofold: (i) to relieve researchers of the burden of dealing with VM creations and workload fluctuations when they evaluate new VM placement algorithms and (ii) to offer the possibility to compare them.

Overview.

VMPlaceS has been implemented in Java by leveraging the SimGrid MSG API. Although Java has an impact on the efficiency of SimGrid, we believe its use is acceptable because Java offers important benefits to researchers for the implementation of advanced scheduling strategies, notably concerning the ease of implementation of new strategies. As examples, we implemented the Snooze proposal in Java and the DVMS proposal using Scala and Java.

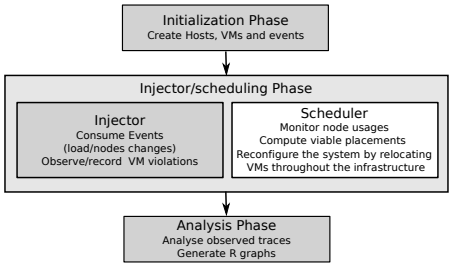


Figure 34: *VMPlaceS's Workflow*

Gray parts correspond to the generic code while the white one must be provided by end-users.

VMPlaceS performs a simulation in three phases, see Figure 34: (i) initialization, (ii) injection and (iii) trace analysis. The initialization phase corresponds to the creation of the environment, the VMs and the generation of an event queue. The simulation is performed by at least two SimGrid processes, one executing the *injector*, the generic part of the framework which is in charge of injecting the events during the ex-

ecution of the simulation, and a second one executing the to-be-simulated *scheduling algorithm*.

The latter analyzes the collected traces in order to gather the results of the simulation, notably by means of the generation of figures representing, *e.g.*, resource usage statistics.

Researchers develop their scheduling algorithm using the SimGrid MSG API and a more abstract interface that is provided by VMPlaceS and consists of the classes `XHost`, `XVM` and `SimulatorManager`. The two former classes respectively extend SimGrid's `Host` and `VM` abstractions while the latter controls the interactions between the different components of the simulator. Through these three classes users can inspect, at any time, the current state of the infrastructure (*i.e.*, the load of a host/VM, the number of VMs hosted on the whole infrastructure or on a particular host, check whether a host is overloaded, etc.).

Initialization Phase.

VMPlaceS first creates n VMs and assigns them in a round-robin manner to the first p hosts defined in the platform file. The default platform file corresponds to a cluster of $h+s$ hosts, where h corresponds to the number of hosting nodes and s to the number of services nodes. The values n, h and s constitute input parameters of the simulations (specified in a Java property file). These hosts are organized in form of topologies, a cluster topology being the most common one. It is possible, however, to define more complex platforms to simulate, for instance, federated data center scenarios.

Each VM is created based on one of the predefined VM classes. A VM class corresponds to a template specifying the VM attributes and its memory footprint. It is defined in terms of five parameters: the number of cores `nb_cpus`, the size of the memory `ramsize`, the network bandwidth `net_bw`, the maximum bandwidth available `mig_speed` and the maximum memory update speed `mem_speed` available when the VM is consuming 100% of its CPU resources. Available classes are defined in a text file that is modifyable by users. As pointed out in Section 5.3.2, the memory update speed is a critical parameter that governs the migration time as well as the amount of transferred data. VM classes provide means to simulate arbitrary kinds of workload (*e.g.*, memory-intensive ones). All VMs start with a CPU consumption of 0 that will evolve during the simulation depending on the injected load as explained below. Once the creation and the assignment of VMs completed, VMPlaceS spawns at least two SimGrid processes, the *injector* and the launcher of the selected scheduler. At its start the *injector* creates an event queue that will be consumed during the second phase of the simulation. Currently, VMPlaceS supports *CPU load change* events (only). The event queue is generated in order to change the load of each VM every t seconds on average. t is a random variable that follows an exponential distribution with rate parameter λ_t while the CPU load of a VM evolves according to a Gaussian distribution defined by a given mean (μ) as well as a given standard deviation (σ). t, μ and σ are provided as input parameters of a simulation. Furthermore, each random process used in VMPlaceS is initialized with a seed that is defined in a configuration file. This way, we can ensure that different simulations are reproducible and may be used to establish fair comparisons.

Finally, we highlight that adding new events can be done by simply defining new event Java classes implementing the `InjectorEvent` interface and by adding the code in charge of generating the corresponding events that are then handled similarly to the *CPU Load* ones.As an example, the next release of VMPlaceS will integrate *node apparition/removal events* that will be used to simulate crashes.

Injector Phase.

Once the VMs and the global event queue are ready, the evaluation of the scheduling mechanism can start. First, the injector process iteratively consumes the different events. Changing the load of a VM corresponds to the creation and the assignment

of a new SimGrid task in the VM. This new task has a direct impact on the time that will be needed to migrate the VM as it increases or decreases the current CPU load and thus its memory update speed.

Based on the scheduler decisions, VMs will be suspended/resumed or relocated on the available hosts. Users must implement the algorithm in charge of solving the VMPP but also the code in charge of applying reconfiguration plans using methods from the `SimulatorManager` class. This step is essential as the reconfiguration cost is a key element of dynamic placement systems.

It is noteworthy that VMPlaceS invokes the execution of each scheduling solver, as a real implementation would do, to get the effective reconfiguration plan. That is, the computation time that is observed is not simulated but corresponds to the effective one, only the workload inside the VMs and the reconfiguration operations (*i.e.*, suspend/resume and migrate) are simulated in SimGrid.

Trace Analysis.

The last step of VMPlaceS consists in analyzing the information that has been collected during the simulation. This analysis is done in two steps. First, VMPlaceS records several metrics related to the platform utilization using an extended version of SimGrid's TRACE module[1]. This way, visualization tools that have been developed by the SimGrid community, such as PajeNG [2], may be used with VMPlaceS. Furthermore, our extension enables the creation of a JSON trace file, which is used to represent resource usage by figures generated using the R statistical environment (Bloomfield, 2014).

By default, VMPlaceS records the load of the VMs and hosts, the start and the duration of each violation of VM requirements in the system, the number of migrations, the number of times the scheduler mechanism has been invoked and the number of times it succeeds or fails to resolve non-viable configurations.

The TRACE API is extensible in that as many variables as necessary can be created by users of our system, thus allowing researchers to instrument their own algorithm with specific variables that record other pieces of information.

6.3 Dynamic VMPP Algorithms

To illustrate the interest of VMPlaceS, we implemented three dynamic VM placement mechanisms: a centralized one based on the Entropy proposal (Hermenier, Lorca, et al., 2009) (see Chapter 2), a hierarchical one based on Snooze (Feller, Rilling, and Morin, 2012), and a fully-distributed one based on DVMS (Quesnel, Lebre, and Südholt, 2013) (see Chapter 3).

These systems search for solutions to violations caused by overloaded nodes. A host is overloaded when its VMs try to consume more than 100% of the CPU capacity of the host. In such a case, a resolution algorithm looks for a reconfiguration plan that can lead to a viable configuration. For the sake of simplicity, we chose to use the latest solver developed as part of the Entropy framework (Hermenier, Demassey, and Lorca, 2011) as this resolution algorithm for all three systems. The Entropy solver evaluates different viable configurations until it reaches a predefined timeout. Once the timeout has been triggered, the algorithm returns the best solution among the ones it finds and applies the associated reconfiguration plan by invoking live migrations in the simulation world. In the remainder of this section, we present an overview of the three systems.

1 http://simgrid.gforge.inria.fr/simgrid/3.12/doc/tracing.html
2 https://github.com/schnorr/pajeng/wiki

Entropy-based Centralized Approach.

The centralized placement mechanism consists in one single SimGrid process deployed on a service node. This process implements a simple loop that iteratively checks the viability of the current configuration by invoking the aforementioned VMPP solver with a predefined frequency. The resource usage is monitored through direct accesses to the states of the hosts and their respective VMs. We also monitor, for each iteration, whether the VMPP solver succeeds or fails. In the case of success, VMPlaceS records the number of migrations

Snooze-based Hierarchical Approach.

Snooze (Feller, Rilling, and Morin, 2012) harnesses a hierarchical architecture in order to support load balancing and fault tolerance, cf. Figure 35. At the top, a *group leader (GL)* centralizes information about the whole cluster using summary data about *group managers (GMs)* that constitute the intermediate layer of the hierarchy. GMs manage a number of *local controllers (LCs)* that, in turn, manage the VMs assigned to nodes.

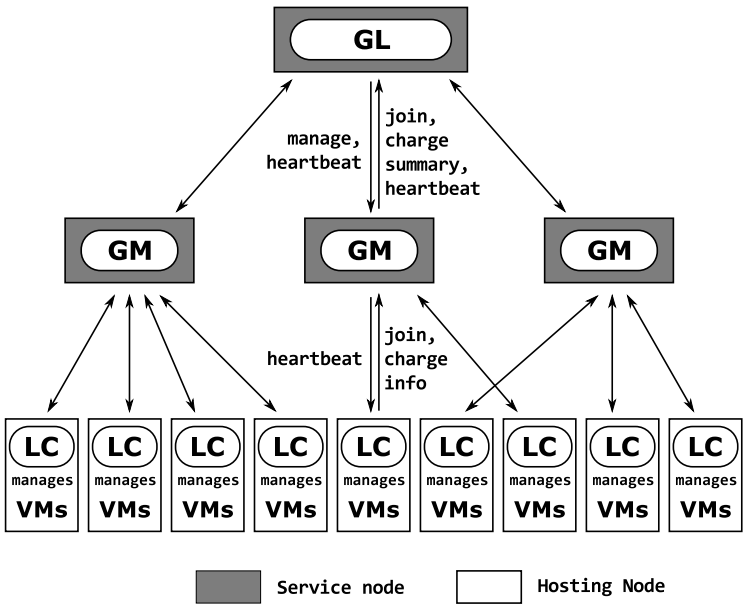


Figure 35: *Snooze Architecture*

During execution, higher-level components periodically send heartbeats to lower-level ones; monitoring information, *e.g.*, about the system load, is also sent periodically in the opposite direction. In order to propagate information, Snooze relies on hardware support for multicast communication.

The implementation in VMPlaceS of the core architectural abstractions of Snooze leverages the `XHOST`, `XVM` and `SimulatorManager` while other mechanisms have been implemented using Simgrid's primitives and standard Java mechanisms. For instance, communication between Snooze actors is implemented based on SimGrid's primitives for, mainly asynchronous, event handling. The multicast capability that is used, *e.g.*, to relay heartbeats, is implemented as a dedicated service that manages a state to relay heartbeat events in a concurrent manner to all receivers. Finally, our Snooze simulation uses, as its original counterpart, a multi-threaded implementation (*i.e.*, based on multiple SimGrid processes) in order to optimize reactivity even for large groups of LCs (or GMs) that have to be managed by one GM (or GL).

DVMS-based Distributed Approach.

DVMS (Distributed Virtual Machine Scheduler) (Quesnel, Lebre, and Südholt, 2013) enables the cooperative and fully-distributed placement of VMs. A DVMS agent is

deployed on each node in order to manage the VMs on the node and collaborate with (the agents of) neighboring nodes. Agents are defined on top of an overlay communication network that defines the node-neighbor relation. We have implemented a simple unstructured overlay that enables the agents to collaborate by providing a link to a neighbor on the latter's request.

When a node detects that it cannot provide enough resources for its hosted VMs, it triggers an *Iterative Scheduling Procedure (ISP)* that enables the node to cooperate with its neighborhood (see Section 3.1.2 for further information).

6.4 Experiments

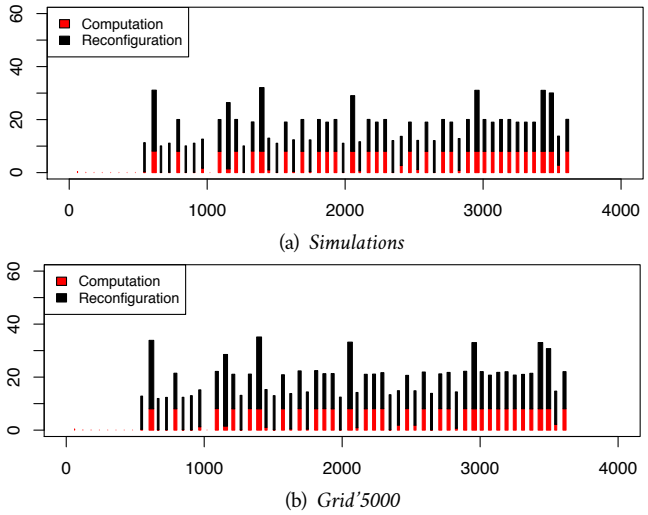
Two kinds of experiments have been performed to validate the relevance of VMPlaceS. The objective of the first one was to evaluate the accuracy of the returned results while the second was a concrete use-case of VMPlaceS, analyzing the three strategies introduced before.

6.4.1 Accuracy Evaluation

To validate the accuracy of VMPlaceS, we have implemented a dedicated version of our framework[3] on top of the Grid'5000 testbed and compared the execution of the Entropy strategy invoked every 60 seconds over a 3600 seconds period in both the simulated and the real world. Regarding the *in-vivo* conditions, experiments have been performed on top of the Graphene cluster (Intel Xeon X3440-4 CPU cores, 16 GB memory, a GbE NIC, Linux 3.2, Qemu 1.5 and SFQ network policy enabled) with 6 VMs per node. Each VM has been created using one of 8 VM predefined classes. The template was 1:1GB:1Gbps:1Gbps:X, where the memory update speed X was a value between 0 and 80% of the migration bandwidth (1Gbps) in steps of 10. Starting from 0%, the load of each VM varied according to the exponential and the Gaussian distributions. The parameters were λ = #VMs/300 and μ= 60, σ = 20. Concretely, the load of each VM varied on average every 5 min in steps of 10 (with a significant part between 40% and 80%). A dedicated `memtouch` program (Hirofuchi, Pouilloux, and Lebre, 2015) has been used to stress both the CPU and the memory accordingly. Regarding the simulated executions, VMPlaceS has been configured to reflect the *in-vivo* conditions. In particular, we configured the network model of SimGrid in order to cope with the network performance of the Graphene servers that were allocated to our experiment (6 MBytes for the TCP gamma parameter and 0.88 for the bandwidth corrective simulation factor).

Figure 36 shows the time to perform the two phases of the Entropy algorithm for each invocation when considering 32 PMs and 192 VMs through simulations (top) and in reality (bottom). Overall, we can see that simulation results successfully followed the in-vivo ones. During the first hundreds seconds, the cluster did not experience VM requirement violations because the loads of VM were still small (*i.e.*, Entropy simply validated that the current placement satisfied all VM requirements). At 540 seconds, Entropy started to detect non viable configurations and performed reconfigurations. Diving into details, the difference between the *simulated* and *in-vivo* reconfiguration time fluctuated between 6% and 18% (median was around 12%). The worst case, *i.e.*, 18%, was reached when multiple migrations were performed simultaneously on the same destination node. In this case and even if the SFQ network policy was enabled, we discovered that in the reality the throughput of migration traffic fluctuated when multiple migration sessions simultaneously shared the same destination node. We confirmed this point by analyzing TCP bandwidth sharing through `iperf` executions. We are currently investigating with the SimGrid core-developers how we

3 The code is available on demand (it used to be at https://github.com/BeyondTheClouds/G5K-VMPlaceS, however, because we do not maintain it anymore, we turned the github repository as a private one).



(a) *Simulations*

(b) *Grid'5000*

Figure 36: *Comparison between simulated and in-vivo Executions*

The Y-axis represents the duration of each Entropy invocation. It is divided into two parts: the time to look for a new configuration (the computation phase in red) and the time to relocate the VMs (the reconfiguration phase in black). Both axis are in seconds.

can integrate this phenomenon into the live-migration model. However, as a migration lasts less than 15 seconds in average, we believe that that the current simulation results are sufficiently accurate to capture performance trends of placement strategies.

6.4.2 Analysis of Entropy, Snooze and DVMS

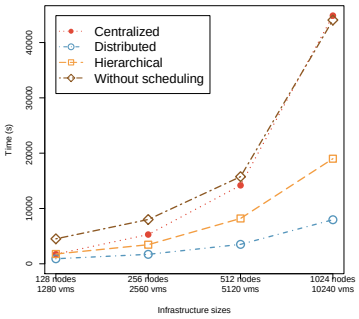
As a validation of our approach (and a contribution by itself), we now provide simulation results comparing the Entropy, Snooze and DVMS strategies. We highlight that most of the DVMS code has been coded in SCALA leveraging the Java primitives of SimGrid for the communications between the different DVMS agents that have been implemented, in turn, using the abstractions of VMPlaceS.

Experimental Conditions.

Each simulation has been executed on a dedicated server, thus avoiding interferences between simulations and ensuring reproducibility between the different invocations. VMPlaceS has been configured to simulate a homogeneous infrastructure of PMs composed of 8 cores, 32 GB of RAM and 1 Gpbs Ethernet NIC. To enable a fair comparison between the three strategies, the scheduling resolver only considered 7 cores, *i.e.*, one was devoted to run the Snooze LC or the DVMS admin processes (a common experimental setup). Ten VMs have been initially launched on each simulated PM. Each VM relied on one of the VM classes described in the accuracy experiment and one set of load-change parameters has been used: λ = #VMs/300, $\mu = 60$ and $\sigma = 20$. The stationary state was reached after 20 min of the simulated time with a global cluster load of 85%. We have performed simulations over a period of 1800 seconds. The consolidation ratio, *i.e.*, the number of VMs per node, has been defined such that a sufficient number of violations is generated. We have discovered that below a global load of 75%, few VM violations occurred under the selected Gaussian distribution we have chosen. This result is rather satisfactory as it can explained why most production DCs target a comparable load level.[4] Finally, infrastructures composed of 128, 256, 512 and 1024 PMs, hosting respectively 1280, 2560, 5120 and 10240 VMs have been investigated. For Entropy and Snooze that rely on service nodes, additional simulated PMs have been provided. For Snooze, one GM has been created per 32 LCs (*i.e.*, PMs). The solver has been invoked every 30s for Entropy and Snooze.

4 http://www.cloudscaling.com/blog/cloud-computing/amazons-ec2-generating-220m-annually/

General Analysis.



Infrastructure size	Duration of **violations** ($\mu \pm \sigma$)		
	Centralized	Hierarchical	Distributed
128 nodes	21.26 ± 13.55	21.07 ± 12.32	9.55 ± 2.57
256 nodes	40.09 ± 24.15	21.45 ± 12.10	9.58 ± 2.51
512 nodes	55.63 ± 42.26	24.54 ± 16.95	9.57 ± 2.67
1024 nodes	81.57 ± 86.59	29.01 ± 38.14	9.61 ± 2.54

Infrastructure size	Duration of **computations** ($\mu \pm \sigma$)		
	Centralized	Hierarchical	Distributed
128 nodes	3.76 ± 7.43	2.52 ± 4.63	0.29 ± 0.03
256 nodes	7.97 ± 15.03	2.65 ± 4.69	0.25 ± 0.02
512 nodes	15.71 ± 29.14	2.83 ± 4.98	0.21 ± 0.01
1024 nodes	26.41 ± 50.35	2.69 ± 4.92	0.14 ± 0.01

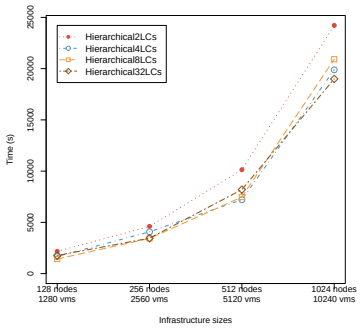
Infrastructure size	Duration of **reconfigurations** ($\mu \pm \sigma$)		
	Centralized	Hierarchical	Distributed
128 nodes	10.34 ± 1.70	10.02 ± 0.14	10.01 ± 0.11
256 nodes	10.26 ± 1.45	10.11 ± 0.83	10.01 ± 0.08
512 nodes	11.11 ± 3.23	10.28 ± 1.50	10.08 ± 0.82
1024 nodes	18.90 ± 7.57	10.30 ± 1.60	10.04 ± 0.63

Figure 37: *Scalability/Reactivity analysis of Entropy, Snooze and DVMS*

Figure 37 presents on the left the cumulated violation time for each placement policy and on the right several tables that give more details by presenting the mean and the standard deviations of the duration of, respectively, the violations and the computation/reconfiguration phases. As anticipated, the centralized approach did not scale and even incurs an overhead in the largest scenario compared to a system that did not perform any dynamic scheduling. The more nodes Entropy has to monitor, the less efficient it is during both the computation and reconfiguration phases. This is to be expected for the computation phase (which tries to tackle an NP-complete problem). As to reconfiguration, the reconfiguration plan becomes more complex for large scenarios, including several migrations coming from and going to the same nodes. Such plans are not optimal as they increase the bottleneck effects at the network level of each involved PM. Such a simulated result is valuable as it confirms that reconfiguration plans should avoid such manipulations as much as possible. The results of the hierarchical approach are clearly better than the Entropy-based ones but worse than those using DVMS-based placement. However, diving into the details, we can see that both the time needed for the computation and reconfiguration are almost independent from the cluster size (around 3s and 10s) and not much worse than those of DVMS, especially for the reconfiguration phase, which is predominant. These results can be easily explained: the centralized policy addresses the VMPP by considering all nodes at each invocation, while the hierarchical and the distributed algorithms divide the VMPP into sub problems, considering smaller numbers of nodes (32 PMs in Snooze and, on average, 4 in the case of DVMS). To clarify the influence of the group size on the performance of Snooze, *i.e.*, the ratio of LCs attached to one GM, we have performed additional simulations for varying group sizes. VMPlaceS has significantly facilitated this study as the corresponding simulations differ just by configuration parameters and do not require modifications to the code base.

Investigating Algorithm Variants.

VMPlaceS facilitates the in-depth analysis of variants of placement algorithms. We have, for example, analyzed, as a first study of its kind, how the Snooze-based placement depends on the no. of LCs assigned to a GM. Figure 38 presents the simulated values obtained for scenarios with 2, 4, 8 and 32 LCs per GM for four infrastructure sizes. The overall performance (*i.e.*, cumulated violation time) shows that 2 LCs per GM result in significantly higher violation times. The relatively bad performance of the smallest group size can be explained in terms of the number of failures of the reconfiguration process, that is, overloading situations that are discovered but cannot



Infra. Size	No. of **failed reconfigurations**			
	2 LCs	4 LCs	8 LCs	32 LCs
128	19	0	0	0
256	29	0	0	0
512	83	1	0	0
1024	173	7	0	0

Infra. Size	Duration of the **computations** ($\mu \pm \sigma$)			
	2 LCs	4 LCs	8 LCs	32 LCs
128	0.16 ± 1.23	0.34 ± 1.81	0.58 ± 2.40	2.53 ± 4.62
256	0.18 ± 1.31	0.42 ± 1.99	0.66 ± 2.50	2.65 ± 4.69
512	0.15 ± 1.20	0.33 ± 1.78	0.67 ± 2.54	2.83 ± 4.98
1024	0.19 ± 1.37	0.42 ± 2.02	0.89 ± 2.90	2.69 ± 4.91

Figure 38: *Hierarchical placement: influence of varying group sizes*

be resolved due to a lack of resources (see tables on the right). Groups of 2 LCs per GM are clearly insufficient at our global load level (85%). Failed reconfigurations are, however, already very rare in the case of 4 LCs per GM and do not occur at all for 8 and 32 LCs per GM. This is understandable because the load profile we evaluated rarely results in many LCs of a GM to be overloaded at once. Violations can therefore be resolved even in the case of a smaller number of LCs available for load distribution. Conversely, we can see that the duration of the computation phases decreases strongly along with the group size. It reaches a value close to the computation times of DVMS for a group size of 4-LCs per GM.We thus cannot minimize computation times and violation times by reducing the number of LCs because larger group sizes are necessary to resolve overload situations if the VM load gets higher. In contrast, DVMS resolves this trade-off by means of its automatic and dynamic choice of the partition size necessary to handle an overload situation. Once again, this information is valuable as it will help researchers to design new algorithms favoring the automatic discovery of the optimal subset of nodes capable to solve violations for given load profiles.

The study performed in this paper has allowed us to analyze several other variants and possible improvements (which we cannot present here for lack of space), such as a reactive approach to hierarchical placement instead of the periodical one used by Snooze, as well as more aggressive partitioning in the case of DVMS. VMPlaceS also provides additional metrics such as the overall count of migrations, the average duration of each migration … These allow important properties, *e.g.*, the migration overhead, to be studied. All these variants can be easily studied and evaluated thanks to VMPlaceS.

Finally, we have succeeded to conduct DVMS simulations up to 8K PMs/80K VMs in a bit less than two days. We did not present these results in this paper because it was not possible to run a sufficient number of Snooze simulations at such a scale (the Snooze protocol being more complex). The time-consuming portions of the code are related to SimGrid internals such as `sleep` and `send/recv` calls. Hence, we are collaborating with SimGrid core developers in order to reduce the simulation time in such cases.

6.5 Related Work

Jobs/tasks scheduling in distributed system is an old challenge and thus several simulators have been proposed to investigate pros and cons of new strategies for several years.As a recent example, Google has released the simulator [5] it used for the Omega framework (Schwarzkopf et al., 2013). However, jobs/tasks scheduling simulators do not consider the notion of VM and its associated capabilities (suspend/resume, migrate) and thus are not appropriate to investigate Cloud Computing production platforms.

5 https://github.com/google/cluster-scheduler-simulator

As discussed in Section 5.5, a few simulator toolkits that have been proposed to address Cloud Computing concerns (Kliazovich et al., 2010; Calheiros et al., 2011; Núñez et al., 2012; Cartlidge and Cliff, 2013) can be classified into two categories. The first corresponds to ad-hoc simulators that have been developed to address one particular concern. For instance, CReST (Cartlidge and Cliff, 2013) is a discrete event simulation toolkit built for Cloud provisioning algorithms. If ad-hoc simulators allow some characteristics of the behaviors of the system to be analyzed, they do not consider the implication of the different layers, which can lead to non-representative results. Moreover, most ad-hoc solutions are developed for one shot analyses. That is, there is no effort to release them as a complete and reusable tool for the scientific community. The second category (Kliazovich et al., 2010; Calheiros et al., 2011; Núñez et al., 2012) corresponds to more generic cloud simulator toolkits (*i.e.*, they have been designed to address multiple Cloud Computing challenges). However, they focus mainly on the API and not on the model of the different mechanisms of Cloud Computing systems. For instance, CloudSim (Calheiros et al., 2011), which has been widely used to validate algorithms and applications in different scientific publications, is based on a top-down viewpoint of cloud environments. That is, there are no articles that properly validate the different models it relies on: a migration time is simply (and often imprecisely) calculated by dividing VM memory sizes by network bandwidth values. In addition to be subject to inaccuracies at the low level, available cloud simulator toolkits often use oversimplified models for virtualization technologies, also leading to non-representative results. As highlighted throughout this chapter, we have chosen to build VMPlaceS on top of SimGrid in order to build a generic tool that benefits from the accuracy of its models related to virtualization abstractions (see Chapter 5).

6.6 Summary

We have presented VMPlaceS, a framework providing generic programming support for the definition of VM placement algorithms, execution support for their simulation at large scales, as well as new means for their trace-based analysis. We have validated its accuracy by comparing simulated and *in-vivo* executions of the Entropy strategy on top of 32 PMs and 192 VMs. We have also illustrated the relevance of VMPlaceS by evaluating and comparing algorithms representative of three different classes of virtualization environments: centralized, hierarchical and fully distributed placement algorithms. Note that in the original implementation of Snooze, a specific heuristic to solve the placement/reconfiguration VM problem is used. As the sake of simplicity, we simply reused the entropy scheduling code for the three systems. The corresponding experiments have provided the first systematic results comparing these algorithms in environments including up to one 1K nodes and 10K VMs. We are in touch with the SimGrid core developers in order to improve our code with the ultimate objective of addressing infrastructures up to 100K PMs and 1 Millions VMs.

The expected extensions that would enable end-users to provision and remove VMs during the execution of a similuation are now available. We should however provide a better API than the one available right now. Regarding start/stop and suspend/resume operations, it is noteworthy to mention that we are working on the integration of a new model for the boot of the VMs (NGuyen and Lebre, 2017). Such a functionality is critical to simulate in accurate way advanced strategies that use those operations. Finally, we are also extending VMPlaceS to benefit from the energy model of physical machines that has been recently integrated into SimGrid. Both features are discussed in the conclusion of this second part.

As future work, we plan to extend VMPlaceS with additional dimensions in order to simulate other workload variations stemming from network and HDD I/O changes. This is important as most of VM placement strategies has only considered CPU and memory dimensions so far. Finally, it would be valuable to play simulations with real traces instead of the statistical models we are using right now. Unfortu-

nately, it is very difficult to find such traces, as most of cloud providers do not want to release them.

A version of VMPlaceS is available on a public git repository: http://beyondtheclouds.github.io/VMPlaceS/.

Conclusion & Open Research Issues

7

Recent and foreseen technical evolutions lead to distributed-infrastuctures of unprecedented dimensions. Evaluating the scalability, robustness and performance of system mechanisms in charge of operating such infrastructures raises severe methodological challenges. Simply executing them is not always possible as it requires to build the complete system beforehand, and it may not even be enough when uncontrolled external load prevents reproducibility. Simulation is an appealing alternative to study such systems. If it may not be sufficient in some cases to capture the whole complexity of the phenomena, simulation allows however to capture some important trends in an easy and convenient way, while ensuring the controllability and reproducibility of experiments.

While simulations have proven its effectiveness in the context of grids and P2P (Buyya and Murshed, 2002; Casanova, Legrand, and Quinson, 2008) for more than a decade, the needs for an accurate cloud simulator toolkit had became critical for the Cloud Computing community. Such a tool should enable researchers to conduct studies on VM-based distributed systems in a productive manner as well as to compare different algorithms with the same simulation platform. Back to 2013 when we started to address this question, only a few toolkits allowed Cloud Computing studies. CloudSim was the most known (Calheiros et al., 2011). It has been built on the same simulation mechanics as GridSim (Buyya and Murshed, 2002) and exposes specific interfaces to conduct cloud studies. The main issue of CloudSim and other available solutions is that they seek a compromise between execution speed and simulation accuracy. To this end, most of them rely on rather simple models of performance that do not allow obtaining realistic results, which is what we were looking for in the context of our placement VM algorithms (see Chapter 3).

In this second part, we have presented the work we have done to deliver a highly-scalable and versatile simulation framework supporting VM environments, keeping in mind our initial goal of delivering to the community a dedicated framework for the evaluation and comparison of VM placement/reconfiguration strategies.

- In Chapter 5, we have described how we leveraged the SimGrid framework to enable the simulation of Infrastructure-as-a-Service Clouds. SimGrid is an almost 15 years-old open source project whose domain of application has kept growing since its creation and whose efforts in term of simulation quality/accuracy are recognized by the community. The extension we have proposed to SimGrid has been designed and implemented in order to be as transparent as possible to other components: SimGrid users can now easily manipulate VMs while continuing to take the advantage of the usual SimGrid MSG API for creating computation and communication tasks either on PMs or on VMs. Such a choice provides a seamless migration path from traditional clusters simulations to IaaSes for hundreds of SimGrid users. This has been made possible by introducing the concept of *abstraction* level (*i.e.*, physical and virtual) into the core of SimGrid. A virtual workstation model, inheriting from the workstation one, has been added into the framework. This model overrides the operations that are VM-specifics only. While implementing the VM abstractions, we have developed a shared-resource calculation mechanism for VMs and a live migration model implementing the precopy migration algorithm of Qemu/KVM. Through experiments, we have confirmed that our simulation framework correctly reproduced live migration behaviors of the real world, un-

der various conditions. To our best knowledge, it was the first class support of live migration in a cloud simulator toolkit.

Although we did not discuss it, we underline that the ANR SONGS French project, which have provided funding for the research activity, has also supported a work focusing on the modeling of datacenters with the same level of the API of Amazon EC2 or OpenStack. [1] This work relies on the set of extensions presented in this manuscript.

Finally, I would like to highlight that our extensions have been integrated into the core of SimGrid since its version 3.11, released in May 2014 and that I am still an active contributor of the corresponding code as discussed later in this chapter.

The current version of SimGrid is available at:
http://simgrid.gforge.inria.fr.

- In Chapter 6, we have described VMPlaceS, a dedicated framework to evaluate and compare VM placement algorithms. Leveraging the aforementioned SimGrid extensions, VMPlaceS provides programming support for the definition of VM placement algorithms, execution support for their simulation at large scale, as well as new means for their trace-based analysis. Globally the framework is composed of two major components: the injector and the VM placement algorithm. The injector constitutes the generic part of the framework (*i.e.*, the one researchers can directly use) while the VM placement algorithm is the component to be analyzed (or compare with other existing algorithms). To illustrate the relevance of VMPlaceS, we have evaluated three algorithms: Entropy (Hermenier, Lorca, et al., 2009), a centralized approach using constraint programming to solve the placement/reconfiguration VM problem, Snooze (Feller, Rilling, and Morin, 2012), a hierarchical approach where each manager of a group invokes Entropy to solve the placement/reconfiguration VM problem, and DVMS (Quesnel, Lebre, and Südholt, 2013), a distributed approach that dynamically partitions the system and invokes Entropy on each partition. One can note that since our initial study, the latest version of Entropy [2] has been integrated into VMPlaceS. This study has shown that VMPlaceS facilitates the implementation and evaluation of variants of placement algorithms. The corresponding experiments have provided the first systematic results comparing these algorithms in environments including up to one thousand nodes and ten thousands VMs in most cases. While such a number is already valuable, we are still working with the SimGrid core developers in order to improve the code of VMPlaceS with the ultimate objective of addressing infrastructures up to 100K physical machines and 1 Million VMs over a period of one day.

  We believe that VMPlaceS will be beneficial to a large number of researchers in the field of Cloud Computing as it enables them to analyze the main characteristics of a new proposal and thus allows *in vivo* experiments to be restricted to placement mechanisms that have the potential to handle Cloud Computing production infrastructures.

  The current version of VMPlaceS is available on a public git repository:
  http://beyondtheclouds.github.io/VMPlaceS/.

Although I have already highlighted a few perspectives for each contribution, I would like to give additional details regarding two of them.

- Following previous works on live-migration, we have conducted an experimental study in order to propose a first-class VM boot time model (NGuyen and Lebre, 2017). Most cloud simulators often ignore the VM boot time or give a

1 http://schiaas.gforge.inria.fr.
2 http://www.btrplace.org

naive model to represent it. However, simple experiments show that the booting time can fluctuate between few seconds up to several minutes according to the system conditions: Keeping in mind that the VM booting process consumes resources, it is obvious that any other co-located workload will impact the booting process of a new VM.

Our work is motivated by the importance of the duration of the boot action for many operations of a cloud system. For instance, if a sporadic VM is mandatory to perform a task, the time to boot it, is a crucial information that may change the decision of provisioning it. More generally, considering the boot time is essential for VM allocation strategies.

In this preliminary study, we have discussed several experiments in order to identify which factors affect the VM boot time and their influence levels. Based on the results, we can confirm that a simple model based on constant values is definitely not accurate enough since it cannot handle the variation of the boot time under workload that may increase up to 50 times under high I/O contentions. We have proposed a first model that succeeds to follow major fluctuation with deviations within 2 seconds in case of CPU contention, and 10 seconds in average for I/O contention (when resource utilization is lower than 90%). Obviously these experiments should be completed with additional ones before the implementation of our model into SimGrid. We plan for instance to analyze the impact of remote-attached volumes that are intensively used in Desktop as a service scenario. However, the results we obtained so far are promising.

- Regarding VMPlaceS, we are finalizing the support for estimating the energy consumption of an infrastructure for different VM placement algorithms. SimGrid has recently integrated an energy consumption model for physical machines. This extension requires only small changes to the initial cluster configuration, *i.e.*, setting one or several p-states with a corresponding power consumption for hosts. SimGrid then estimates the power consumed by a host proportionally to its current CPU load. The energy consumption is fed to SimGrid's TRACE module like any other metric, allowing easy post-mortem visualization and analysis. [3] To illustrate this new feature, we have implemented, by using the VMPlaceS abstractions, a second centralized placement algorithm based on the *First Fit Decreased* proposal (Johnson, 1973) (FFD). FFD favors VM consolidation and enhance the possibility to turn-off hosts when they are not executing VMs. We expect to demonstrate that VMPlaceS eases the comparison of the energy efficiency of placement policies by comparing the two centralized algorithms with respect to this new metric. We are consolidating right now the different metrics we have gathered, and prepare a longer article that will present in details this new feature.

Finally, I would like to highlight that maintaining and moving forward the code of the SimGrid extensions, as well as VMPlaceS, require a considerable effort in terms of engineering. This effort becomes more and more important according to the number of features all the SimGrid community continue to integrate. While on the first side, it is valuable to keep those extensions up-to-date for the scientific community, it is unfortunately, on the other side, not-well recognized by our institution and sometimes a lost battle. Moreover, simulator toolkits such as SimGrid should face the high velocity of technologies in particular in the Cloud Computing domain. For instance, containers technology has also become a fundamental building block of distributed computing environments. It would be valuable for our community to propose abstractions such as the ones we proposed for system virtualization. However, it will require important efforts that should be sustained by recurrent funding programs.

3 http://simgrid.gforge.inria.fr/tutorials/simgrid-energy-101.pdf.

This point that may look out-of-scope of this manuscript, has been decisive in the way I started my recent research activities: while we target the design and development of a fully distributed IaaS toolkit, we have chosen to revise a well-known software stack instead of developing a system from scratch. This way enables us to mitigate engineering efforts as discussed in the third and last part.

Part IV

BEYOND THE CLOUDS: RECENT, ONGOING AND FUTURE WORK

Although this part relies on publications that have been, or in the process of being, published (Bertier et al., 2014; Lebre, Pastor, Simonet, et al., 2017; Cherrueau et al., 2017), the way of presenting differs from the two previous activities. My goal here is to give an overview of my recent, ongoing and future activities. First, I present the DISCOVERY initiative, an open-science action I have been leading since 2014 and that will last at least till July 2019. Note that this chapter does not present a single scientific contribution as such, but rather gives an overview of the DISCOVERY goals leveraging the original Inria Project Lab proposal I submitted two years ago. The objective is to introduce to readers the foundations on which rely my ongoing activities and the ones that will follow. Second, I compiled a short version of two contributions we did during these two first years. Finally, I conclude this part and this manuscrit by presenting a few challenges I plan to address for the next 4 years and probably more.

These results, which are relatively new in comparison to the ones presented before, have also been made possible thanks to several persons that took part to numerous brainstorming sessions [4]

Although, I cannot list of of them here, I would like to recognize the commitment of individuals like:

- Flavien Quesnel, Phd in the Ascola Research Team (0ct 2010- Feb 2013), Nantes, now Research Engineer at System-X Institute ;
- Cédric Tedeschi, Ass. Profesor in the MYRIADS Research Group, University of Rennes 1, Rennes ;
- Marin Bertin, Ass. Professor in the ASAP Research Group, INSA, Rennes ;
- Frédéric Desprez, Senior researcher and Deputy Scientific Director at Inria, Grenoble ;
- Jonathan Pastor, Phd in the Ascola Resarch Team (Oct 2012 - Oct 2016), Nantes, now PostDoc at Chicago University;
- Anthony Simonet, PostDoc Researcher in the Ascola Research Team (Oct 2015-2017), Nantes ;
- Matthieu Simonin, Research Engineer at Inria, Rennes ;
- Ronan-Alexandre Cherrueau, Research Engineer within the framework of DISCOVERY Inria Project Lab (2016, 2019), Nantes;
- Dimitri Pertin, PostDoc Researcher in the ASCOLA Research Team (Oct 2016-Jan 2018), Nantes.

I should also complete this list with first, my academic colleagues who have recently agreed to join me in this initiative: Dr. Christian Perez, Dr. Gilles Fedak, Dr. Anne-Cecile Orgerie, Dr. Mario Südholt Dr. Shadi Ibrahim and Dr. Helene Coullon, and second, all Researchers, Engineers and Phd candidates from Orange Labs. They have all contributed a lot through their fruitful exchanges. Finally, I should thanks, Thierry Carrez, Director of Engineering at OpenStack Foundation for his availability and wise counsels he has been giving us since Dec 2015.

I sincerely hope I did no omit anyone, in which case, I apologize.

Between 2013 and 2014, the activities have been mainly supported by the Ecole des Mines de Nantes (now IMT Atlantique). Since 2015, the Discovery initiative is mainly supported through the Inria Project Labs program and the I/O labs, a joint lab between Inria and Orange Labs. I underline that I am the principal investigator of this initiative. Finally, I have been chairing the Massively Distributed Working Group of the OpenStack community since April 2016. This second action is complementary to the scientific one as it enables us to get contact with strong industrials such as AT&T, Deutch Telekom …in addition to Orange Labs. An important aspects in the current context of the European Research that favor cooperation between academic and industrial sector.

Major Results as well as software code are available at: `http://beyondtheclouds.github.io/` and `https://wiki.openstack.org/wiki/Massively_Distributed_Clouds`

4 Please visit the Discovery website for a complete up-to-date list.

The DISCOVERY Initiative

8

The strong adoption of the Cloud Computing paradigm has greatly favored the involvement of our community to address major challenges faced by Cloud Computing providers. Several scientific contributions led to technical evolutions that allowed the consolidation of services and applications in large-scale data centers of ever-increasing size. Although this is still a main trend for achieving efficiency and reducing total cost of ownerships, experts from academia and industry have started to advocate the evolution of Cloud Computing infrastructures towards a massively distributed federation of smaller data centers placed at the edge of network backbones. This paradigm shift is dictated by requirements for improved Quality of Service and growing user concern of trust/privacy issues, along with technological advances in the capacity and capabilities of servers, mobile networks and end-user devices.

Among the obstacles to the adoption of this model, referred to as Fog/Edge Computing, is the development of a convenient and powerful software stack capable of managing a significant number of remote data-centers deployed at the edge in a unified way.

We published a first study on how such a system could be designed (Lebre, Anedda, et al., 2011) through a collaboration with colleagues from the CRS4 research center in Italy. This work led to the foundations of the DISCOVERY[1] open-science initiative three years later (Bertier et al., 2014) The objective of DISCOVERY is to design, prototype and evaluate the mechanisms and policies required for efficient management of edge cloud infrastructures. This emerging paradigm shows great potential to vastly improve service agility, QoS, and availability, by bringing resources closer to end-users. As described in this chapter, our approach is fundamentally different from today's broker-based solutions as we chose to revise the OpenStack software suite in order to make it inherently cooperative.

8.1 Context & Motivations

Over the last decade, mostly economic concerns have driven consolidation of services and applications in Data Centers (DCs) of ever-increasing size, where the number of physical resources that one DC can host is limited by the capacity of its energy supply and its cooling system (see Figure 39a). To meet these critical needs in terms of energy supply and cooling, the current trend is toward building mega-DCs in regions presenting energy advantages, such as cheap hydroelectricity (see Figure 39b) or location advantages, such as proximity to the polar circle which allows free cooling (Gary Cook, 2013).

(a) *Microsoft DC in Quincy (WA)*

(b) *Quincy DCs*

Figure 39: Quincy - Where Agriculture meets Technology. Cloud providers come to the Quincy area because of its abundant and cheap hydroelectricity.

However, concentrating Mega-DCs in few attractive locations has two main problems: (a) reduced availability and (b) increased response time. A disaster[2] in these areas would be catastrophic for all services hosted in the DCs leading to long service

1 http://beyondtheclouds.github.io.

2 On March 2014, a large crack has been found in the Wanapum dam leading to emergency procedures. This hydro plant supports major data centers in central Washington, including Quincy.

disruption. Similar problems can be caused by network disruptions that can affect all services. Second, hosting resources in a few locations leads to increased overheads, perceived as lower QoS by users. Such overheads can prevent the adoption of Cloud Computing by latency-sensitive applications and lead to significantly lower applications where hierarchical approaches work well, e.g. in the IoT domain. This drawback has been highlighted by industry (Gartner, 2014) as well as scientific studies (B. Zhang et al., 2015).

Localized or micro data centers are a fact of life, but by applying a self-contained, scalable and remotely managed solution and process, CIOs can reduce costs, improve agility, and introduce new levels of compliance and service continuity. Creating micro data centers is something companies have done for years, but often in an ad hoc manner. Gartner, Jan 2015

The concept of micro/nano DCs deployed through the backbone and up to the edge (A. Greenberg et al., 2008; Satyanarayanan et al., 2009; Bonomi et al., 2012; Garcia Lopez et al., 2015) is an alternative to Mega-DCs and a promising solution to the aforementioned concerns. Such a paradigm, referred to as "Fog/Edge Computing", has been discussed in the past but was never realized because operating multiple, small DCs may appear counter-intuitive from the point of view of economies of scale in terms of physical resources and administration effort. However, today, technological advances in the capacity and capabilities of servers, mobile networks and end-user devices, along with the advent of new usages related to Internet of Things applications (IoT) (Atzori, Iera, and Morabito, 2010), Mobile Edge Computing (MEC) (A. Ahmed and E. Ahmed, 2016) and Network Function Virtualization (NFV) (Mijumbi et al., 2015) present new opportunities for placing resources closer to the edge (B. Zhang et al., 2015). The benefits of micro data centers have been also highlighted already by industry and practitioners (David Cappuccio, 2015).

The main barrier for the adoption of such a decentralized model of Cloud Computing is the increased complexity in terms of managing, monitoring, and controlling a complex and diverse network of resources and the extension of current interfaces to turn location into a first-class citizen.

The objective of DISCOVERY initiative is to improve management aspects of edge-clouds and facilitate their deployment. As opposed to existing approaches that rely on brokering systems, we proposed a new model of coordinating a significant number of sites through a common software platform with self-* and P2P mechanisms. Since 2014 (Bertier et al., 2014), we have been advocating the need to redesign a system specifically for edge-clouds that will interact with low level mechanisms available on each physical resource (compute storage and network) and leverage advanced P2P mechanisms to coordinate them in a native manner (see Section **??**). This strategy differs from available solutions that have been built using "glue" components in the form of broker or orchestration services.

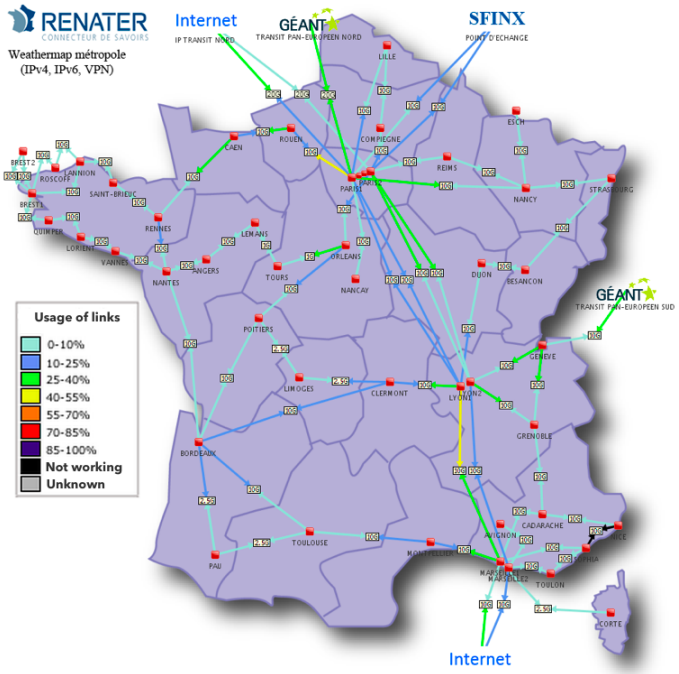
To mitigate the effort required for designing and developing such a distributed cloud management system, we chose, in 2015 in coordination with Orange Labs, to revise the OpenStack core-services. After more than six years of intensive effort, the OpenStack software suite (*OpenStack* website) has become the *de facto* open-source solution to operate, supervise and use a Cloud Computing infrastructure. We made the bet that revising Openstack will allow us to reuse as much as possible from already successful mechanisms and to focus on the key challenges raised by such massively distributed cloud infrastructures.

This chapter presents the foundations on which rely my ongoing and future activities. First, I describe how we proposed to mitigate the cost overheads of deploying and operating multiple DCs that have been enlightened many times by the defenders of large-scale DCs. Second. I discuss some key elements that motivate the choice of designing a system capable of supervising a set of remote DCs in a unified way. I explain why federated proposals (Buyya, Ranjan, and Calheiros, 2010) are not the best approaches to operate Fog/Edge infrastructures and why designing a fully distributed system by revising the OpenStack software suite makes sense. Third, I present several opportunities that can bring an Internet-Scale IaaS manager. Finally, I conclude this chapter by highlighting a few research directions where we expect to contribute.

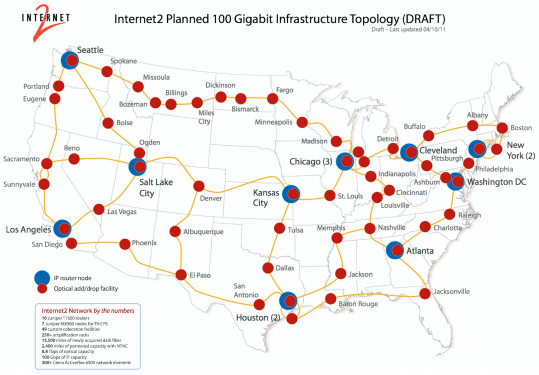
8.2 From Cloud to Fog/Edge Computing Facilities

Micro/Nano DCs relevance has been largely discussed because operating multiple small DCs breaks somehow the idea of resources pooling and thus increase the operational costs. To answer this question once for all, we conducted on own study and confirmed (Simonet, Lebre, and Orgerie, 2016) that distributed Cloud Computing solutions can compete in terms of economical viability with the industry references (Amazon, Microsoft, Google …). The study highlights in particular opportunities for telcos and network operators that can leverage their existing network facilities, starting from the core nodes of the network backbone to the different network access points (*a.k.a.*,. PoPs – Points of Presence) in charge of interconnecting public and private institutions. By hosting micro/nano DCs in PoPs, it becomes possible to mutualize resources that are mandatory to operate network/data centers while delivering widely distributed CC platforms better suited to cope with disasters and to match the geographical dispersal of users.

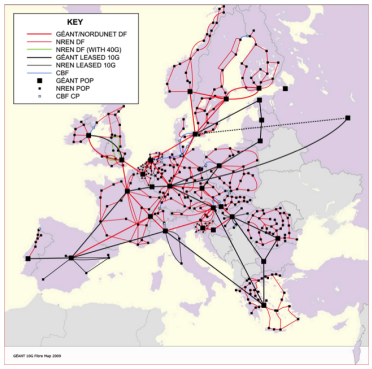
Figure 40: *Examples of NREN backbones*



(a) *Topology/Metrology of the RENATER network (the French NREN) - Each red square corresponds to one network point of presence (72 PoPs overall)*



(b) *Topology of the Internet2 Network (one of the USA NREN)*



(c) *The GEANT federation*

Figures 40a and 40b illustrate the advantages of such a proposal. They present the network topology of the French and USA National Research and Education Network (NREN), i.e. the backbone dedicated to universities and research institutes in France and respectively in USA.

Both figures reveal several important points:

- There are a significant number of network points of presence across each territory;
- Both backbones have been deployed to avoid disconnections (i.e. more than 95% of the PoPs can be reached by at least two distinct routes);
- The backbone was deployed and is renewed to match the demand. In particular for RENATER, we can see that the density/capacity of points of presence as well as the bandwidth of each link are more important on the edge of large cities such as Paris and Marseille.
- The metrology information available on the RENATER map confirms that most resources are underutilized: only two links are used between 45% and 55% (i.e. the links gathering the traffic toward the Internet), a few between 25% and 40%, and the majority below the threshold of 25%;

More generally, all NRENs are interconnected to constitute larger network backbones such as the GÉANT PAN in Europe as depicted on Figure 40c. GÉANT connects over 50 million users at 10,000 institutions across Europe and supporting research in different areas such as energy, environment, space and medicine. Such a European PAN is composed of significant number of PoPs that can be used to deploy and provide a competitive European Cloud for the academics but also all governmental institutions for instance.

Figure 41: *From Containerized micro DCs to massively distributed clouds*

(a) *Deployment of the SF's micro DC*

(b) *Deployment of a new PoP of the Orange French backbone*

(c) *SF's micro DC final installation*

Figure 41 completes the illustration of such an *in-network distributed cloud*. The two pictures on the left show the deployment and the final installation of the Schneider micro DC on the construction site of the Sagrada Familia in Barcelona (Spain) in order to support the construction process as well as ticketing and other SF's processes[3]. The pictures on the right show the deployment of a new network PoP for the Orange French network. From the hardware point of view, such a container looks similar to a containerized DC such as the SF one. We can easily envision that such a PoP can

3 Why a world famous church needs a micro DC? (Nov 2015, Schneider blog)

be equipped with few servers in charge of delivering compute, storage and network capacities.

While from the economical point of view, a few models have been proposed to determine where such micro/nano DCs should be deployed (A. Greenberg et al., 2008; Simonet, Lebre, and Orgerie, 2016), the question of how operating such an infrastructure composed of a significant numbers of of sites still remains. Indeed, at this level of distribution, latency and fault tolerance become primary concerns, and collaboration between components that are hosted on different location must be organized wisely. This is the question we try to solve within the framework of the DISCOVERY initiative.

8.3 Revising OpenStack Through a Bottom/Up Approach

The massively distributed cloud we target is an infrastructure that is composed of up to hundreds of micro DCs, which are themselves composed of up to one hundred servers (up to two racks). In this section, I discuss design considerations that has motivated our implementation choices of revising OpenStack with P2P mechanisms.

8.3.1 Design Considerations

Broker vs Cooperative Systems

Brokering/Orchestration of clouds are the first approaches that are considered when it comes to operate and use distinct clouds. Each micro DC hosts and supervises its own Cloud Computing infrastructure and a brokering service is in charge of provisioning resources by picking them on each cloud. While these top/down approaches with a simple centralized broker can be acceptable for basic use cases, advanced brokering services become mandatory to meet requirements of production environments (monitoring, scheduling, automated provisioning, SLAs enforcements…). In addition to dealing with scalability and single point of failure (SPOF) issues, brokering services become more and more complex to finally integrate most of the mechanisms that are already implemented by IaaS managers (Buyya, Ranjan, and Calheiros, 2010; Houidi et al., 2011). Consequently, the development of a brokering solution is as difficult as the development of management system of CC resources (*a.k.a.*, a Iaas manager or a cloud kit) but with the complexity of relying only on the least common denominator APIs. While several initiatives have been working for several years on Cloud Computing standards such as OCCI (Loutas et al., 2010), they do not allow developers to manipulate low-level capabilities of each system, which is generally mandatory to finely administrate resources. In other words, building mechanisms on top of existing ones, as it is the case of federated systems, prevents them from going beyond the provided APIs (or require intrusive mechanisms that must be adapted to the different cloud computing software stacks).

The second way to operate a distributed cloud infrastructure is to design and build a dedicated system which will define and leverage its own software interface, thus extending capacities of traditional Clouds with its API and a set of dedicated tools. Designing a specific system offers an opportunity to go beyond classical federations by addressing all crosscutting concerns of a software stack as complex as a Cloud Computing resource management system.

From a Hierarchical to a Peer-to-Peer IaaS Manager

Considering the advantage of an Internet-Scale Cloud Computing manager with respect to brokering proposals, the next question is to analyze whether collaborations between mechanisms of the system should be structured either in hierarchical or

in flat way via a P2P scheme. During the last years few hierarchical solutions have been proposed in industry (*Cascading OpenStack* web; *Scaling solutions for OpenStack* web) and academia (Feller, Rilling, and Morin, 2012; Farahnakian et al., 2014). Although they may look easier at first sight than Peer-to-Peer structures, hierarchical approaches require additional maintenance costs and complex operations in case of failure of critical peers. Moreover, mapping and maintaining a relevant tree architecture on top of massively distributed infrastructures is not meaningful. Static partitioning of resources that is usually performed is not appropriated for constantly changing environments such as Fog/Edge Computing platforms. Finally, a hierarchical structure means that there is a global entry point that does not enable to address the latency requirements of modern usages such as IoT, NFV an MEC applications (every cloud request going through the global entry point before being served by one DC). As a consequence, hierarchical approaches do not look to be satisfactory to operate a massively distributed IaaS infrastructure such as the one we target. On the other side, P2P file sharing systems are a good example of software that works well at large scale in a context where Computing/Storage resources are geographically spread. While P2P/decentralized mechanisms have been under-used for building operating system mechanisms, they have showed the potential to handle the intrinsic distribution of massively distributed cloud infrastructures as well as the scalability required to manage them (DeCandia et al., 2007).

To summarize, DISCOVERY advocates the development of a dedicated system that will interact with low level mechanisms available on each physical resource (compute storage and network) and leverage advanced P2P mechanisms to coordinate them.

8.3.2 The Choice of OpenStack

The Internet-scale Cloud manager we target should deliver a set of high level mechanisms whose assembly results in a system capable of operating an IaaS infrastructure.

IaaS Manager Fundamentals

Recent studies have showed that state of the art IaaS managers (Peng et al., 2009) were constructed over the same concepts and that a reference architecture for IaaS managers can be defined (Moreno-Vozmediano, Rubén S Montero, and Ignacio M Llorente, 2012).

This architecture covers primary services that are needed for building the LUC OS:

- The **virtual machines manager** is in charge of managing VMs' cycle of life (configuration, scheduling, deployment, suspend/resume and shut down).
- The **Image manager** is in charge of VM' template files (*a.k.a.*, VM images).
- The **Network manager** provides connectivity to the infrastructure: virtual networks for VMs and external access for users.
- The **Storage manager** provides persistent storage facilities to VMs and the hosted applications.
- The **Administrative tools** provide user interfaces to operate and use the infrastructure.
- Finally, the **Information manager** monitors data of the infrastructure for the auditing/accounting.

Thus the challenge within the DISCOVERY initiative is to guarantee for each of the aforementioned services that it can work in a fully distributed way. However, as designing and developing each of those systems from scratch would be an herculean work, we propose to minimize both design and implementation efforts by reusing as much as possible successful mechanisms, and more concretely by investigating

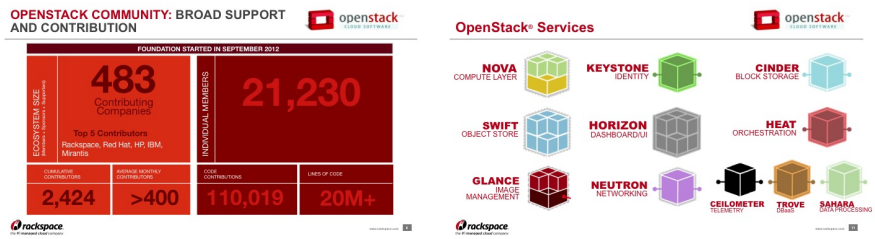
whether a revised version of the OpenStack (*OpenStack* website) could fulfill requirements to operate a Fog/Edge infrastructure. In other words, we propose to determine which parts of OpenStack can be directly used and which ones must be revised with P2P approaches. As already highlighted, OpenStack has become the *de facto* open-source solution to operate, supervise and use a Cloud Computing infrastructure.

This strategy would enable us to focus the effort on key issues such as the distributed functioning and the organization of efficient collaborations between software components composing of OpenStack.

OpenStack Ecosystem Overview

OpenStack (*OpenStack* website) is an open-source project that aims at developing a complete Cloud Computing management system. The description available on the OpenStack website is the following :

OpenStack software controls large pools of compute, storage, and networking resources throughout a data center, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure. Hundreds of the world's largest brands rely on OpenStack to run their businesses every day, reducing costs and helping them move faster. OpenStack has a strong ecosystem, and users seeking commercial support can choose from different OpenStack-powered products and services in the Marketplace. The software is built by a thriving community of developers, in collaboration with users, and is designed in the open at our Summits.



(a) *OpenStack community overview in 2015*

(b) *OpenStack services*

Figure 42: *OpenStack EcoSystem*

Figures 42a and 42b complete the aforementioned description with the objective to illustrate the richness of the OpenStack ecosystem. Started in 2012, the OpenStack foundation gathers 500 organizations with groups such as Google, IBM, Intel …The software platform relies on tens of services with a development cycle based on a 6-month basis. It is composed of more than 2 millions of lines of code, mainly in python, just for the core-services. While these aspects make the whole ecosystem quite veloce, they are also good signs of the maturity of the community.

Within the framework of DISCOVERY, we focus on revising the core services (*i.e.*, the low-level components) that deliver the Compute, Network and Storage as a services capabilities. With the achievements of such an objective, we expect that most of the higher level OpenStack services will be able to benefit from the low level revisions in a rather comfortably and straight forward way.

The Open-Source Cascading OpenStack/ TriCircle Solution

Although there is, as far as we know, no other action that investigates how OpenStack can be revised to cope with Fog/Edge Computing requirements, it is noteworthy to mention the Tricircle action.

Proposed and supported by Huawei, the Tricircle project[4] has been an important attempt to address the massively distributed cloud use-case. At coarse-grained the idea was rather similar to the Cells approach available by default in the Nova OpenStack core services and used by in the CERN infrastructure[5]. Indeed, both approaches are taking a cascading approach to forward the application requests to one particular site in charge of serving it. In Tricircle, a Top minimalistic "OpenStack API gateway" manages multiple Bottom (*a.k.a.*, cascaded) OpenStack instances, which may very well be running in different sites. While the current implementation is based on a central service, the top layer is a logical concept that itself could be distributed and deployed throughout distinct data centers for ensuring reliability as well as scalability of the solution.

While both initiatives, DISCOVERY and Tricircle targeted the same objective, our vision is in the opposite direction precisely. From our point of view Tricircle can be compared to a brokering approach dedicated for OpenStack systems, that is built only on top of the OpenStack API. By consequence, Tricircle suffers of the same drawbacks of brokering approaches that have been highlighted in Section 8.3.1. Moreover, by addressing the challenge of distributing the Tricircle top layer, developers were facing the same challenges we have identified to distributed OpenStack core-services. In other words, the Tricircle developers had to redeveloped most of the mechanisms that were already available at low level to be able to deliver an advanced orchestrating services but then they should also redistributed these mechanisms between distinct sites. As previously stressed, we claim for a bottom/up approach where the distribution of OpenStack core-services should be achieved by default in order to ensure an efficient cooperation of distinct OpenStack instances throughout distinct sites. Nevertheless, as we are facing similar issues such as the sharing of VM images between distinct sites or the extension of the API to reify the location dimension, we are in touch with the Tricircle community. As an example, we will present in the Pike Summit in Boston a shared presentation. [6]

8.4 Innovation Opportunities

While advantages and opportunities of massively distributed clouds have been emphasized several years ago (Church, A. G. Greenberg, and Hamilton, 2008; A. Greenberg et al., 2008), delivering an OpenStack that can natively be extended to distinct sites will create new opportunities.

Micro Datacenters, also known by the adorable name "cloudlets," as a key concept for optimizing the performance and usefulness of mobile and other networked devices via the cloud. Service providers have embraced this vision most strongly from the start, but it won't be long before enterprise IT pros will likely do the same
Victor Bahl, Microsoft Research distinguished scientist Victor, March 2015

From the infrastructure provider point of view, Internet Service Providers (ISPs), academic and private institutions in charge of operating a part of the Internet network will be able to build competitive Fog/Edge Computing platforms in a relatively short time and with a limited cost. Instead of redeploying a complex and expensive infrastructure, they will be able to leverage IT resources and devices such as computer room air conditioning units, inverters or redundant power supplies already present in each network point of presence of their backbone. While we outlined the advantages of such scenario for NRENs in details in Section 8.2, we should complete the description by highlighting that such *in-network* cloud infrastructure could be extended for telecom operators first on each radio base station they operate and second to the extreme edge by leveraging resources available on the home gateways as depicted on Figure 43.

4 https://wiki.openstack.org/wiki/Tricircle_before_splitting and https://wiki.openstack.org/wiki/Tricircle

5 http://superuser.openstack.org/articles/openstack-in-production-cern-s-cloud-in-kilo

6 When one Cloud is not enough: an overview of sites, regions, edges, distributed clouds and more (Boston OpenStack Summit, May 2017).

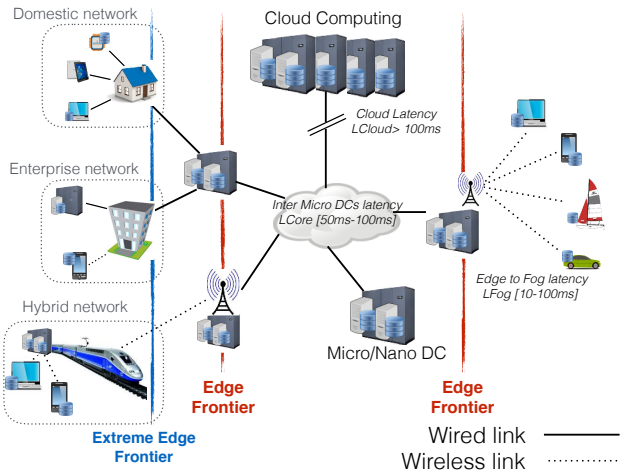


Figure 43: *From the Cloud to the Extreme Edge - Infrastructure Vision*

In addition to deploying computation/storage capabilities in PoPs, the Fog/Edge model can be extended first to each radio base station that enables to interconnect smartphones and IoT devices to the network backbone, and second, to the Extreme Edge by using resources provided by third party tenants (smartboxes, private micro DCs ...)

Similarly, nano/micro DCs solutions proposed by large companies such as Schneider[7] and SMEs such as Celeste[8] in France will benefit from a cooperative OpenStack in order to operate their DCs not on a single basis but globally, delivering a more flexible offer to their end-users. When the capabilities offered by a micro DC will not be sufficient to address the peak of activity it will be possible to temporarily leverage additional resources available from another center of the company that deployed the micro DC solution. More generally, a distributed version of OpenStack will natively allow the federation of a significant number of DCs to constitute a global one. Such a feature is a strong advantage for the private cloud model in comparison to the current hybrid offers that breaks the notion of a single deployment operated by the same tenant. Each time a company will face a peak of activity, it will be possible to provision dedicated servers and attach them to the initial deployment. Those servers can either be provided by dedicated hosting services that have DCs close to the institution (for instance, one can consider to extend the resource available in a company (left side on the figure) with resources provided by the first micro DCs present in the network) or by directly deploying transportable and containerized server rooms close to the private resources. The major advantage is that such an extension is completely transparent for the administrators/users of the IaaS solution because they continue to supervise/use the infrastructure as they are used to without sharing the resources with other tenants.

This notion of WAN-wide elasticity can be generalized, as it will be possible to deploy containerized server rooms whenever and wherever they will be mandatory. As examples, we can envision to temporarily deploy IT resources for sport events such as olympic games or for public safety purposes in case of disasters (the TCS use-case) or any other specific need (see the Sagrada Familia example in Barcelona in Section 8.2) and connect them to the global DC of a company.

The common thread in all scenarios is the possibility of extending an infrastructure wherever needed with additional resources, the only constraint being to be able to plug the different locations with a backbone that offers enough bandwidth and quality of service to satisfy network requirements. The last scenario we can envision is related to current European start-ups that propose to combine computing and heat technologies around specialized micro/nano DCs such as Cloud&Heat[9] in Germany and Qarnot Computing[10] in France. These companies that have a business model relying on the notion of decentralized micro/nano DCs can also take the advantage of a cooperative OpenStack to operate here also their infrastructure globally. I under-

7 http://www.schneider-electric.us/en/solutions/system/s4/data-center-and-network-systems-micro-data-center/
8 https://www.celeste.fr/stardc-offre
9 https://www.cloudandheat.com/en/technology.html
10 http://www.qarnot-computing.com

line that I am involved in a new French ANR Research project coordinated by Qarnot Computing as further discussed in Chapter 11.

From the software provider point of view, DISCOVERY will favor the development of advanced applications dealing with smart-homes/cities, e-governmental services, public safety and more generally all challenges related to the industrial internet [11]. Developers will be able to design new applications but also revise major cloud services in order to deliver more locality aware management of data, computation, and network resources. For instance, it will be possible to deploy on-demand Content Delivery Network solutions according to specific requirements. Cloud storage services such as Dropbox could be revised to mitigate the overheads of transferring data from sources (Confais, Lebre, and Parrein, 2017). Instead of the currently prevalent push of data to remote clouds, DISCOVERY will allow to support pulling data closer to the user location. Nowadays data is mostly uploaded to the remote clouds without considering whether such data movements are effectively solicited or not. Among the DISCOVERY objectives one is to enable data to stay as close as possible to the source that generates them and be transferred on the other side only when it will be solicited. Such strategies will mitigate the cost of transferring data in all social networks for instance but can also be leveraged to address trust/privacy challenges. Similarly, developers will be able to deliver data analytic software stack that enable computations to be launched close to data sources. Such mechanisms will be shortly mandatory to handle the huge amount of data that Internet of Things will generate.

8.5 Summary

Similarly to the key services that have been proposed in the past for Cloud Computing and that led to its success, our community should propose building blocks for Fog/Edge infrastructures. Those building blocks are mandatory (i) to allow operators to supervise these massively distributed hererogenous and dynamic platforms and, (ii) to favor the development of new kinds of *geo-aware* services.

In this chapter, I presented an overview of the DISCOVERY initiative I have been leading since 2014. After describing the physical infrastructure we consider, I discussed some design considerations that justify our choice of (i) not implementing another brokering approach and (ii) revising OpenStack instead of developing a system from scratch.

Although delivering an efficient distributed version of an Internet-Scale IaaS manager is a challenging task, we believe that addressing it is the key to favor the emergence of a new generation of Utility Computing. We expect DISCOVERY to pave the way for exploring in detail, how to operate a widely distributed infrastructure in a unified manner and what are the main challenges to solve. We believe that the DISCOVERY initiative will increase significantly the knowledge base and will provide insights to break new ground in the Fog/Edge Computing research area. In particular, we expect to contribute to several key aspects of large-scale distributed infrastructure management:

- Performance and QoS via taking into account location as a first class citizen.
- Elasticity and scalability via fundamental extensions to scheduling and datapath (network and storage) mechanisms.
- Availability for disaster scenarios by enabling partial infrastructure operation.
- Mobility and disconnected operation by using weak consistency and reconciliation.

11 http://www.iiconsortium.org.

- Reactivity and robustness by treating infrastructure (compute, storage, network) changes due to reconfiguration or failures as common events, rather than the static approaches used today.
- Programming Models and abstractions by extending current cloud APIs to allow end-users and developers to exploit the unique features of Fog/Edge platforms.

Overall, and compared to current state-of-the-art approaches, DISCOVERY will address fundamental infrastructure management concerns that have long been the main barrier for adopting new infrastructure models and improving infrastructure efficiency and application QoS. As the deployment of Fog/Edge computing is expected to increase over the next years, conducting research in this area is timely and key for industry as described in Section 8.4.

Finally, we claim that choosing OpenStack is a key element as it should enable the scientific community to attract key industrial actors. As examples, Orange Labs, British Telecom as well as major European NRENs already expressed their interest in our action. Moreover, we believe that it is time for the scientific community to get involved and contribute to the OpenStack software in the same way it has been done for the Linux ecosystem. We expect that preliminary works such as the ones presented in the next chapters will have a decisive impact in both the scientific and the open-source community to gather experts around the goal that have been presented in this chapter: Delivering an open-source software stack for Fog/Edge Computing infrastructures.

9 Revising OpenStack to Operate Fog/Edge Computing Infrastructures: a First Proof-Of-Concept

Academic and industry experts are now advocating for going from large-centralized Cloud Computing infrastructures to smaller ones massively distributed at the edge of the network. Among the obstacles to the adoption of this model is the development of a convenient and powerful IaaS system capable of managing a significant number of remote data-centers in a unified way.

In this chapter, we introduce the premises of such a system by revising the OpenStack software, a leading IaaS manager in the industry. The novelty of our solution is to operate such an Internet-scale IaaS platform in a fully decentralized manner, using P2P mechanisms to achieve high flexibility and avoid single points of failure. More precisely, we describe how we revised the OpenStack Nova service by leveraging a distributed key/value store instead of the centralized SQL backend. We present experiments that validate the correct behavior and gives performance trends of our prototype through an emulation of several data-centers using Grid'5000 testbed. In addition to paving the way to the first large-scale and Internet-wide IaaS manager, we expect this work will attract a community of specialists from both distributed system and network areas to address the Fog/Edge Computing challenges within the OpenStack ecosystem.

9.1 Challenge Description

As stressed in the previous chapter, concentrating Mega-DCs in only a few attractive places implies different issues. First, a disaster in these areas would be dramatic for IT services the DCs host as the connectivity to CC resources would not be guaranteed. Second, in addition to jurisdiction concerns, hosting computing resources in a few locations leads to useless network overheads to reach each DC. Such overheads prevent the adoption of the Cloud Computing paradigm by several kind of applications such as mobile computing (Satyanarayanan et al., 2009) and IoT (B. Zhang et al., 2015).

The concept of micro/nano DCs deployed at the edge of the backbone has been proposed as a promising solution for the aforementioned concerns (A. Greenberg et al., 2008). Although it has been introduced as soon as 2008, the model has been debated for a couple of years because it was believed that there would be no way to operate multiple small DCs without increasing initial and exploitation expenditures. A recent study demonstrated that a model leveraging existing network facilities (*a.k.a.*, network point of presences) can deliver competitive solutions from the economic viewpoint in comparison to current Amazon offers (Simonet, Lebre, and Orgerie, 2016).

While the question of whether Fog/Edge Computing platforms will be deployed is not being debated anymore, the question of how operating such a widely geo-distributed infrastructure still remains. Indeed, at this level of distribution, latency and fault tolerance become primary concerns, and collaboration between components that are hosted on different locations must be organized wisely. With their *IOx* and *Fog Direc-*

tor software solutions (Bonomi et al., 2012), Cisco allows IoT applications to run on infrastructures composed of NFV-enabled hardware (at the edge) and existing centralized clouds. However, their solution does not allow to run Virtual Machines and does not compete with existing IaaS platforms.

In this chapter, I present how we revised OpenStack (*OpenStack* website) in order to deliver a first-class Internet-scale IaaS manager. Revising OpenStack allow us to propose a system that is as convenient to administrate and as easy to use as existing IaaS managers. Specifically, we describe how we revised the *Nova* service (the OpenStack compute element) with a decentralized key/value store in place of the centralized SQL database. This revision enables us to distribute Nova over several geographical sites. The correct functioning of this proof of concept has been validated via several experiments performed on top of Grid'5000. In addition to tackling both the scalability and distribution issues of the SQL database, our prototype leads to promising performance. More than 80% of the API requests are performed faster than with the SQL backend without doing any modification in the *Nova* code.

With this work, we hope to bring attention to the matter and start building a large community around an OpenStack-based Internet-scale IaaS Manager, like it was once done for Linux and HPC.

The remaining of the chapter is as follows. Section 9.2 describes OpenStack and how we revised it. The validation of our prototype focusing on the Nova service is presented in Section 9.3. Finally, Section 9.4 concludes and discusses future research and development actions. Note that the original paper (Lebre, Pastor, Simonet, et al., 2017) includes a discussion related to the design and development of an Internet-scale IaaS manager. However, those aspects have been presented in the previous chapter. We invite readers to refer to Section 8.3 for futher information, in particular to better understand why we chose to revise OpenStack with P2P mechanisms.

9.2 Revising OpenStack

OpenStack (*OpenStack* website) is an open-source project that aims to develop a complete Cloud Computing software stack. Figures 44 presents the general vision of OpenStack with the three expected capabilities of IaaS platforms: Compute Network and Storage. Applications at the top can request through a high level API compute, network and storage resources. OpenStack bricks in the middle communicate through shared services. From the technical point of view, OpenStack is composed of two kinds of nodes: on the first side, the compute/storage/network nodes are dedicated to deliver the XaaS capabilities such as hosting VMs for the compute; on the other side the controller nodes are in charge of executing the OpenStack services.

Figure 44: *OpenStack Overview*

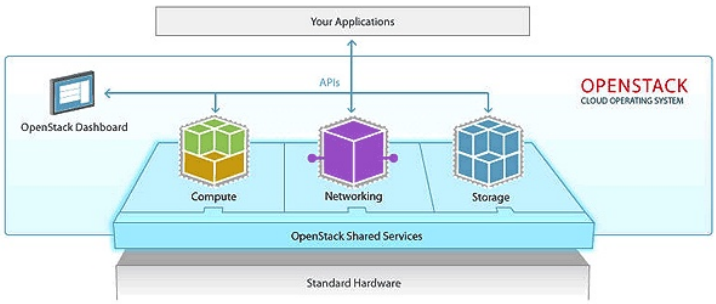


Figure 45 shows the core-services of OpenStack. This architecture is comparable with the reference one described in the previous section.

OpenStack services are organized following the *Shared Nothing* principle. Each instance of a service (*i.e.*, service worker) is exposed through an API accessible through a *Remote Procedure Call* (RPC) system implemented on top of a messaging queue or via web services (REST). This enables a weak coupling between services. During their

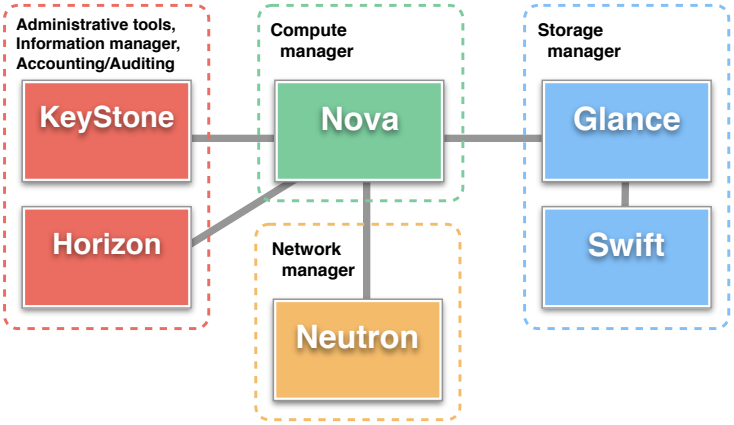


Figure 45: *Core-Services of OpenStack.*

life-cycle, services create and manipulate logical objects that are persisted in shared databases, thus enabling service workers to easily collaborate. However, even if this organisation of services respects the *Shared Nothing principle*, the message bus and the fact that objects are persisted in shared databases limit the scalabilty of the system, as stated in the OpenStack documentation:

> *OpenStack services support massive horizontal scale. Be aware that this is not the case for the entire supporting infrastructure. This is particularly a problem for the database management systems and message queues that OpenStack services use for data storage and remote procedure call communications.*

To conclude, revising OpenStack to make it fully decentralized should be carried out in two ways: distributing the messaging queue as well as the shared relational databases.

9.2.1 Distributing the AMPQ Bus

As indicated, services composing OpenStack collaborate mainly through a RPC system built on top of an AMQP bus. The AMQP implementation used by OpenStack is *RabbitMQ*. While this solution is generally articulated around the concept of a centralized master broker, it also provides a cluster mode that can be configured to work in a highly available manner. Several machines, each hosting a RabbitMQ instance, work together in an Active/Active functioning where each queue is mirrored on all nodes. While it has the advantage of being simple, it has the drawback of being very sensitive to network latency, and thus it is not relevant for multi-site configurations. This limitation is well known from the distributed messaging queue community. Few workarounds have been proposed for systems such as RabbitMQ and more recently, P2P-like solutions such as ActiveMQ (Snyder, Bosnanac, and Davies, 2011) or ZeroMQ (Hintjens, 2013) have been released.

Thoses broker-less approaches satisfy our requirements in terms of scalability. Considering that there is already an action to use ZeroMQ in place of RabbitMQ in OpenStack[1], we chose to focus our efforts on the DB challenge.

9.2.2 Distributing the Databases

As of today, only a few actors such as Rackspace [2] have been dealing with large-scale challenges. In other words, most of the OpenStack deployments only involve a few compute nodes and do not require more than a single database (DB) node. The use of a second DB is generally proposed to meet the high availability constraint that is

1 https://wiki.openstack.org/wiki/ZeroMQ
2 http://rack.ly/6010B2xpQ

mandatory in production infrastructures. In such a context, the OpenStack community recommends the use of at least an *active/passive* replication strategy (a second DB acts as a failover of the master instance).

When the infrastructure becomes larger or includes distinct locations, it becomes mandatory to distribute the existing relational DBs over several servers. Two approaches are proposed by the OpenStack community.

The first one consists in partitioning the infrastructure into groups called *cells* configured as a tree. The top-cell is in charge of redistributing requests to the children cells. Each child cell can be seen as an independent OpenStack deployment with its own DB server and message queue broker. In addition to facing hierarchical approach issues we previously discussed (see Section **??**), we highlight that additional mechanisms are mandatory in order to make collaboration between cells possible (there is no communications nor interactions between children cells). Consequently, this approach is more comparable to a brokering solution than to a native collaboration between IaaS core-services.

The second approach consists in federating several OpenStack deployments throughout an *active/active* replication mechanism of DBs (Kemme and Alonso, 2010) through solutions such as *Galera*: when an instance of a service processes a request and performs some actions on one site, changes in the inner-states stored in the DB are also propagated to all the other DBs of the infrastructure. From a certain point of view, it gives the illusion that there is only one unique DB shared by all OpenStack deployments. Although the described technique has been used in production systems, most of them only involve a limited number of geographical sites. Indeed, active replication mechanisms imply important overheads that limit the size of infrastructures. To sum up neither the hierarchical approach nor the active replication solution are suited to deal with a massively distributed infrastructure as the one we target.

While not yet explored for the main OpenStack components, NoSQL databases built on top of recent P2P key/value stores seem to have more suitable properties for highly distributed contexts, providing better scalability and built-in replication mechanisms. The challenge consists in analyzing how OpenStack internals can be revised to be able to store service states in a key/value store instead of a classical SQL system. In the next section we present how we revised the Nova component in this direction.

9.2.3 The Nova POC: From MySQL to Redis

Nova is the OpenStack core-service in charge of VM management. Its software architecture is organized in a way which ensures that each of its sub-services does not directly manipulate the DB (see Figure 46). Instead it calls API functions proposed by a service called "nova-conductor". This service forwards API calls to the "**db.api**" component that proposes one implementation per database type. Currently, there is only one implementation that works on relational DBs. This implementation relies on the *SQLAlchemy* object-relational-mapping (ORM) that enables the manipulation of a relational database via object oriented code.

Thanks to this software pattern, we developed *ROME*, an ORM library that enables interactions with key/value stores in the same way *SQLAlchemy* interacts with relational DBes. *ROME* has been developed in a non intrusive way by proposing the same API as SQLAlchemy as well as the same internal mechanisms (such as Queries, JOIN operations and Sessions —the SQLAlchemy equivalent of transactions). These implementations have been achieved in order to take into account key/value store specifics: first, we developed a distributed lock system on top of *Redis* as well as a two-phase commit approach to support *sessions*, enabling atomic modifications of the key/value store like traditional SQL transactions; second we included a secondary index in order to speed up complex queries containing join operations.

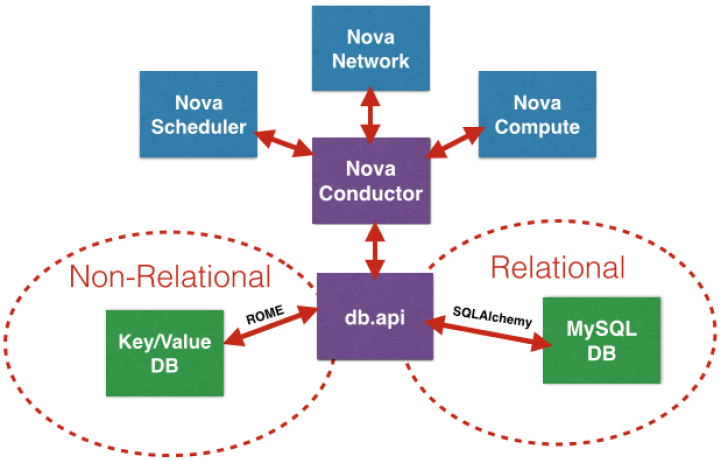


Figure 46: *Nova - Software Architecture and DB dependencies.*

The integration of *ROME* within the OpenStack Nova code was restricted to the "**db.api**" file where every call to *SQLAlchemy* has been replaced with a call to *ROME*. Thanks to this modification, it is now possible to operate and use a multi-site OpenStack infrastructure by relying solely on Redis, a P2P key/value store solution. We chose to use Redis in our prototype because of its deployment/usage simplicity.

Figure 47 depicts such a deployment where a key/value store and a global shared AMQP bus enabled all controllers to interact. The number of controller nodes on each site can vary according to the expected demand created by end-users: sites with a large number of compute nodes can have several controllers whereas a controller node can be be mutualized with a compute node as illustrated for Site 3.

We highlight that any controller node can provision VMs by invoking services on the whole infrastructure and not only on the site where it is deployed. In other words, because the states and the communications are managed globally, any service can satisfy any request.

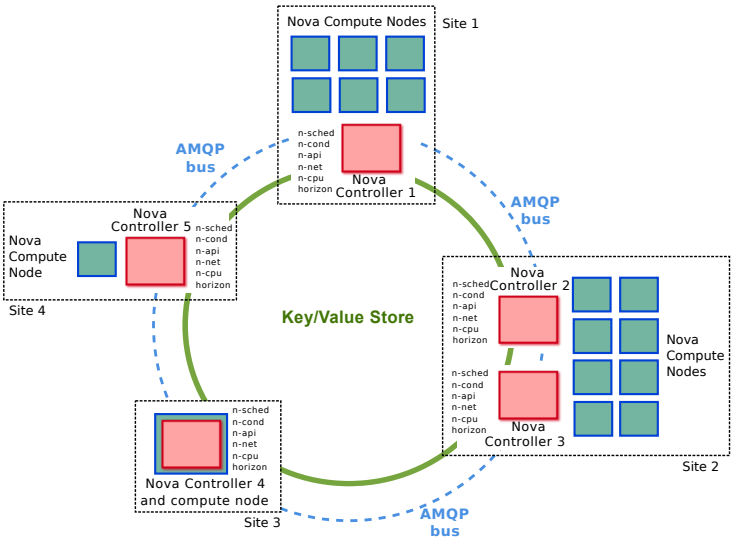


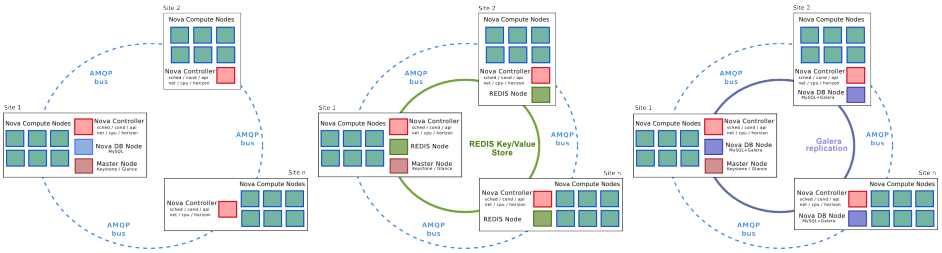
Figure 47: *Distributed Deployment of Nova*

Nova controlers (in light-red) are connected through a shared key/value backend and the AMQP bus. Each controller runs all nova services and can provision VMs on any compute node (in light blue).

9.3 Experimental Validation

The validation of our prototype has been done *via* three sets of experiments targeting the main OpenStack services. The first one aimed at measuring the impact of the use of Rome/Redis instead of SQLAlchemy/MySQL in a single site deployment. The

Figure 48: *Investigated deployments on top of G5K and role of each server node.*



(a) *Single centralized DB* (b) *Redis Key/Value Store* (c) *Galera*

second set focused on multi-site scenarios by comparing the impact of the latency on our distributed Nova service with respect to an active/active Galera deployment. Finally, the last experiment enabled us to evaluate whether our low level revisions impact higher level OpenStack mechanisms such as the *host-agregates* capability.

All Experiments have been performed on Grid'5000, a large-scale and versatile experimental testbed that enables researchers to get access to a large amount of bare-metal computing resources with very fine control of the experimental conditions. Our experiments use the Parasilo and Paravance clusters, that share a similar configuration[3]: two 4-core Intel Xeon E5-2630v3 CPUs @ 2.4 GHz, 128 GB of RAM and two 10 GB ethernet network interface. We deployed and configured each node involved in the experiment with a customized software stack (Ubuntu 14.04, OpenStack Kilo and Redis v3) using Python scripts and the Execo toolbox (Imbert et al., 2013). We underline that we used the legacy network mechanisms integrated in Nova (*i.e.*, without Neutron) and deployed other components that were mandatory to perform our experiment (in particular Keystone and Glance) on a dedicated node on the first site (entitled master node on Figure 48).

9.3.1 Impact of Redis w.r.t MySQL

Time penalties

Changes made over Nova's source code to support Redis is likely to affect its reactivity. The first reason is that Redis does not support operations like joining, and thus the code we developed to provide such operations implies a computation overhead. The second reason is related to networking: unlike a single MySQL node, data is spread over several nodes in a Redis deployment, leading to several network exchanges for one request. Finally, Redis provides a replication strategy to deal with fault tolerant aspects, also leading to possible overheads.

Table 7: *Average response time to API requests for a mono-site deployment (in ms).*

Backend configuration	Redis	MySQL
1 node	83	37
4 nodes	82	-
4 nodes + repl	91	-

Table 7 compares average response times used to satisfy API requests made during the creation of 500 VMs on an infrastructure deployed over one cluster (containing 1 controller node and 6 compute nodes), using either Redis or MySQL under three scenarios: i) a single-node MySQL database, ii) a 4-node Redis database with no replication and iii) a 4-node Redis database with replication. While the distribution of Redis between several nodes and the use of the replication feature do not significantly increase the response time (first column), the difference between the average API response time of Rome/Redis and SQLAlchemy/MySQL may look critical at first sight (124% higher). However, it must be mitigated with Figure 49 and Table 8.

3 https://www.grid5000.fr/mediawiki/index.php/Rennes:Hardware

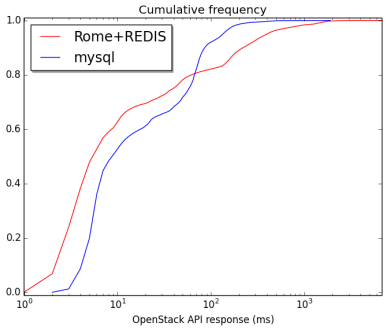


Figure 49: *Statistical distribution of Nova API response time (in ms.).*

Figure 49 depicts the statistical distribution of the response time of each API call that has been made during the creation of the 500 VMs. It is noticeable that for a large part of them (around 80%), Rome/Redis delivers better performance than SQLAlchemy/MySQL. On the other side, the 10% of the slowest API calls are above 222 ms with our proposal while they are are around 86 ms when using MySQL. Such a difference explains the averages recorded in Table 7 and we need to conduct deeper investigations to identify the kind of requests and how they can be handled in a more effective way. Overall, we can see that even with these slow-requests, the completion time for the creation of 500 VMs is competitive with respect to SQLAlchemy/MySQL as illustrated by Table 8. In other words, some API functions have a more significant impact than others on the VM creation time.

Backend configuration	Redis	MySQL
1 node	322	298
4 nodes	327	-
4 nodes + repl	413	-

Table 8: *Time used to create 500 VMs on a single cluster configuration (in sec.)*

Networking penalties

As we target the deployment of an OpenStack infrastructure over a large number of geographical sites linked together through the Internet, the quantity of data exchanged is an important criterion for the evalution of our solution. In particular, as data is stored in the key/value store with an object structure, it requires a serialization/deserialization phase when objects are stored/queried. To enable this serialization, the addition of some metadata is required, which leads to a larger data footprint and thus a larger amount of data exchanged between database and OpenStack nodes.

To determine wether the level of network-overhead is acceptable or not, networking data has been collected during the previous experiments. Table 9 compares the total amount of data exchanged over network depending of the database configuration that has been used. As MySQL does not store serialized objects, *i.e.*, objects are serialized on the client-side by the ORM, only raw data is exchanged over the network. We, thus, consider the single node MySQL as the reference solution, which has been measured at 1794MB. Our solution deployed over a single Redis node exchanges 2190MB, which means that the networking overhead related to the combination of ROME and Redis is estimated around 22%. Doing the same experiment with a 4-node Redis cluster, without data replication, leads to a 33% networking overhead compared to a single MySQL node. Finally, when the data replication is enabled with one replica, the amount of data exchanged over the network is 76% higher than without replication.

Because those values cumulates the network traffic overall, it is not possible to analyze whether the replacement of SQLAlchemy/MySQL by Rome/Redis leads to additional traffic between the controller/compute and the DB nodes. To answer such

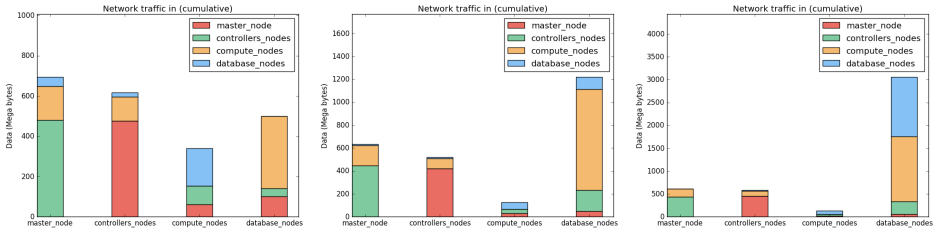
Table 9: *Amount of data exchanged over the network (in MBytes)*

Backend configuration	Redis	MySQL
1 node	2190	1794
4 nodes	2382	-
4 nodes + repl (1 replica)	4186	-

a question, we used the **iftop** [4] tool, and gathered information about the origin and destination of TCP/IP messages exchanged during the creation of VMs. Figure 50 depicts the data exchanges in function of the origin (horizontal axis) and the destination (vertical bars), with varying database configurations. For example, on the first bar, the green area corresponds to the in network of controller nodes where the source is the master node.

Regarding the exchanges between the different kinds of nodes, we observe that there is no significant variation between the three scenarios. The only variation concerns the DB nodes (Figures 50a and 50b). This confirms that the networking overhead depicted in Table 9 comes from the DB nodes. Finally, Figure 50c confirms that most of the overhead observed when enabling data replication is also caused by additional data exchanged between database nodes. This enables us to conclude that using Rome/Redis in place of SQLAlchemy/MySQL is acceptable from the network viewpoint.

Figure 50: *Amount of data exchanged per type of nodes, varying the DB configuration (Y-axis scales differ).*



(a) *Single centralized DB* (b) *4 nodes Redis cluster without replication* (c) *4 nodes Redis cluster with replication (1 replica)*

9.3.2 Multi-site Scenarios

The second experiment we performed consisted in evaluating a single OpenStack deployment spread over several locations. Our goal was to compare the behavior of a single MySQL OpenStack with the advised Galera solution and our Rome/Redis proposal. Figure 48 depicts how the nodes have been configured for each scenario. While the deployment of a single MySQL node is a non sense in a production infrastructure as discussed before, evaluating this scenario enabled us to get an indication of the maximum performance we can expect. Indeed, in this scenario, there is no synchronization mechanism and consequently no overhead related to communications with remote DB nodes. Moreover, conducting such an experiment at large scale enabled us to see the limit of such a centralized approach.

Regarding the experimental methodology, *distinct locations* (*i.e.*, clusters) have been emulated by adding latency between group of servers thanks to the `TC` Unix tool. This enables us to ensure reproducibility between experiments. Each *cluster* contains 1 controller node, 6 compute nodes, and one DB node when needed. Scenarios including 2, 4, 6, and 8 clusters have been evaluated, leading to infrastructures composed of up to 8 controllers and 48 compute nodes in total. The latency between each cluster has been set to 10 ms and then 50 ms.

We evaluate the stress induced on the controllers when provisioning and booting 500 VMs at the same time. VM provisioning queries are fairly distributed amongst the controllers.

4 http://www.ex-parrot.com/pdw/iftop/

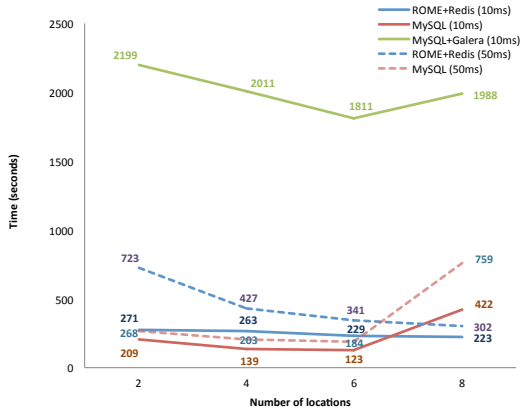


Figure 51: *Time to create 500 VMs with a 10ms and 50ms inter-site latency*

Due to synchronization issues, MySQL+Galera values with 50 ms delay are absent.

Figure 51 presents the observed times. As expected, increasing the number of clusters leads to a decrease of the completion time. This is explained by the fact that a larger number of clusters means a larger number of controllers and compute nodes to handle the workload. The results measured with a 10ms latency show that our approach takes a rather constant time to create 500 VMs, which stabilizes around 220 seconds whatever the number of sites involved in the experiments. Although a single MySQL node scales up to 6 locations, the performance degrades from 8 locations. In this case, the single MySQL performs 89% slower than our approach and the advised Galera solution is 891% slower than our approach. With a 50ms inter-cluster latency, the difference between Redis and MySQL is accentuated in the 8 clusters configuration, as MySQL is 151% slower than our Redis approach.

Regarding Galera, it is noteworthy that important issues related to synchronization issues appear with a 50 ms latency, preventing the collection of trustable results. Such pathological behaviors are due to 1) the high latency between clusters and 2) the burst mode we used to create the 500 VMs.

To summarize, in addition to tackling the distribution issue, the couple Rome/Redis enables OpenStack to be scalable: the more controllers are taking part to the deployment, the better performance.

9.3.3 Compatibility with Advanced Features

The third experiment aimed at validating the correct behavior of existing OpenStack mechanisms while using our Rome/Redis solution. Indeed, in order to minimize the intrusion in the OpenStack source code, modifications have been limited to the `nova.db.api` component. This component can be considered as the part of Nova that has the most direct interaction with the DB. Limiting the modification to the source code of this component should enable us to preserve compatibility with existing mechanisms at higher level of the stack. To empirically validate such an assumption, we conducted experiments involving multi-site and the usage of host-aggregate (*a.k.a.*, availability-zone) mechanism (one advanced mechanism of OpenStack that enables the segregation of the infrastructure). Similarly to the previous experiments, our scenario involved the creation of 500 VMs in parallel on a multi-sites OpenStack infrastructure deployed on top of Rome/Redis. Two sets of experiments were conducted: a set where each node of a same geographical site was member of a same host aggregate (that is the same availability zone) and a second set of experiments involving flat multi-site OpenStack (*i.e.*, without defining any availability zone). Being compatible with the host-aggregate feature is important because as previously mentioned in our flat approach, any controller can be involved in any provisioning request.

Experimental results show that the host-aggregate/availability-zone mechanism behaves correctly on top of our proposal: VMs are correctly balanced according to

the locations where they have been started while the flat multi-site deployment led to a non uniform distribution with respectively 26%, 20%, 22%, 32% of the created VMs for a 4-location experiment.

9.4 Summary

Proposing a software solution to manage "in network" Cloud Computing infrastructures is a key challenge of our community. In this chapter, we presented our view on how such a software stack could be designed by leveraging the OpenStack software suite. As a proof-of-concept, we presented a revised version of the Nova service that uses our ROME library to store states of the OpenStack services in Redis, a P2P key/-value store system. We discussed several experiments validating the correct behavior of our prototype and showed promising performance when operating 8 locations with a single decentralized OpenStack instance.

Our ongoing activities focus on two aspects. First, we are integrating similar changes in other OpenStack core services such as Glance and Neutron.

Second, we study how it can be possible to restrain the visibility of some objects manipulated by the different controllers. Indeed, having low level mechanisms such as a global bus or a key/value system spread WANwide can be source of networking overheads: our POC manipulates objects that might be used by any instance of a service, no matter where it is deployed. We are investigating how some objects could benefit from a restrained visibility: If a user has build an OpenStack project (tenant) that is based on few sites, appart from data-replication, there is no need for storing objects related to this project on external sites. Restraining the storage of such objects according to visibility rules would save network bandwidth and reduce overheads. We are in particular investigating whether network pooling strategies such as the ones proposed in Cassandra (Lakshman and Malik, 2010) may be more appropriate to the geo-distribution of the infrastructure.

As future work, we plan to evaluate the relevance of the newSQL emerging technology (Pavlo and Aslett, 2016). NewSQL systems are appealing since they would allow the gathering of "the best of both worlds" - SQL and NoSQL - by distributing the data while keeping strong properties on the stored datas. Although they are supposed to support natively the MySQL protocol (meaning that switching from a MySQL database to a NewSQL should be very simple), current implementation [5] only support a subset of the MySQL protocol. Our objective is to evaluate to what extent, this technology can be used for OpenStack core-services that require ACID properties.

A version of our Proof-of-concept is available on public git repositories:
Rome code - https://github.com/BeyondTheClouds/rome
Revised nova code (Liberty based) - https://github.com/BeyondTheClouds/nova.

Finally, I underline that this work has been presented at the Spring OpenStack summit in 2016 [6] before being published (Lebre, Pastor, Simonet, et al., 2017). This presentation enabled us to exchange with the OpenStack foundation and to take part to different discussions/working groups. For instance, we have been taking part in the performance team since this date. We perform in particular performance evaluations in order to identify major bottlenecks in the OpenStack Vanilla code. To this aim, we designed the EnOS framework that is introduced in the next chapter. I also highlight that we have set up a new working group dedicated to the massively distributed use-case [7]. Being involved in such actions is important as it allows the community to identify complementary actions and move forward faster.

5 https://github.com/cockroachdb/cockroach or https://github.com/pingcap/tidb
6 A Ring to Rule them All: Revising Openstack Internals to Operate Massively Distributed Clouds (Austin, April 2016).
7 https://wiki.openstack.org/wiki/Massively_Distributed_Clouds

10 *Conducting Scientific Evaluations of OpenStack*

By massively adopting OpenStack for operating small to large private and public clouds, the industry has made it one of the largest running software project, overgrowing the Linux kernel. As discussed in previous chapters, this has been a key element for selecting the OpenStack software suite for the DISCOVERY initiative.

However, with success comes increased complexity; facing technical and scientific challenges, developers are in great difficulty when testing the impact of individual changes on the performance of such a large codebase, which will likely slow down the evolution of OpenStack. Thus, we claim it is now time for the scientific community to join the effort and get involved in the development of OpenStack, like it has been once done for Linux.

In this spirit, we developed EnOS (Cherrueau et al., 2017), an integrated framework that relies on container technologies for deploying and evaluating OpenStack on any testbed. EnOS allows researchers to easily express different configurations, enabling fine-grained investigations of OpenStack services. EnOS collects performance metrics at runtime and stores them for post-mortem analysis and sharing. The relevance of the EnOS approach to reproducible research is illustrated by evaluating different OpenStack scenarios on the Grid'5000 testbed.

While the presentation of EnOS is rather general in the sense that it has been designed to help researchers/engineers to deal with any kind of OpenStack challenges related to performance evaluations, I highlight that EnOS has been designed following the preliminary study. Being able to perform reproducible experiments at large scale is a key element for the Discovery initiative. As discussed at the end of this chapter, EnOS is strongly used in the DISCOVERY consortium to identify in particular how OpenStack core-services behave in a WAN context.

10.1 Challenge Description

Although the adoption of Cloud Computing has been largely favored by public offers (Amazon EC2 and Microsoft Azure, to name a few), numerous private and public institutions have been contributing to the development of open-source projects in charge of delivering Cloud Computing management systems (*CloudStack* website; *OpenNebula* website; *OpenStack* website). In addition to breaking vendor lock-in, these *operating systems* of Cloud Computing platforms enable administrators to deploy and operate private cloud offers, avoiding issues such as data-jurisdiction disputes, latency constraints, etc.

As mentioned in the previous chapters, the OpenStack software suite has become the de facto open-source solution to operate, supervise and use a Cloud Computing infrastructure (*OpenStack* website). Despite its dynamism that makes whole ecosystem incredibly hard to keep up with, OpenStack has been adopted in a large variety of areas such as public administrations, e-commerce and science[1]. Its undeniable success and spread urges the scientific community to get involved and contribute to the OpenStack software in the same way it has been once done for the Linux ecosystem. A major involvement of our community would enable OpenStack to better cope with

1 See http://superuser.openstack.org/ for further information

ongoing changes in Cloud Computing, such as Fog and Edge Computing and IoT applications.

To help developers and researchers identify major weaknesses of a complex system such as OpenStack and to facilitate the evaluation of proposed improvements, we designed EnOS (Experimental eNvironment for OpenStack). [2] Leveraging container technologies and "off-the-shelf" benchmarks, EnOS is the first holistic framework for evaluating OpenStack in a flexible and reproducible way. Its *Experimentation-as-Code* vision allows to automate every step of the experimentation workflow, from the configuration to the results gathering and analysis. In addition, because each service is isolated in a single container, EnOS is able to express and deploy complex scenarios, and to evaluate each service individually.

Moreover, EnOS has been designed around pluggable mechanisms that allow researchers and developers to evaluate OpenStack on various infrastructures (testbed platforms, public and private clouds, *all-in-one* Virtual Machines), and execute any benchmarking suite in addition to Rally (*Rally* web) and Shaker (*Shaker* web) that are supported by default. Finally, EnOS comes with visualization tools that provide multiple synthetic views of the gathered metrics suitable for explanatory and exploratory experiments.

In this chapter, I give a first overview of the EnOS system. Section 10.2 presents the different technologies we used to build the EnOS framework. The framework itself is discussed in Section 10.3. To illustrate the possibility offered by our framework, I discuss a first series of experiments that have been conducted thanks to EnOS in Section 10.4. Section 10.5 discusses related work.

Finally Section 10.6 concludes and discusses future research and development actions.

10.2 Background

To limit the effort in terms of development, EnOS has been built on top of different technologies that are presented in the following. For further information regarding OpenStack, we invite readers to refer to Chapters 8 and 9.

10.2.1 Deploying OpenStack and controlling Experiments

Due to the richness and complexity of the OpenStack ecosystem, making the deployment of OpenStack easy has always been an important topic. Among all the deployment solutions that are available, we chose to use Kolla (*Kolla* web). Kolla provides production ready containers and deployment tools for operating OpenStack infrastructures. In Kolla, each OpenStack service is encapsulated with its dependencies in a dedicated container. Container images can be built on demand, stored and used during the deployment. Kolla features many default behaviors, allowing quick prototyping, but they are fully customizable: vanilla or modified versions of OpenStack can be installed, deployment topologies can be adapted to the user's needs and configuration of all the services can be changed. To perform remote actions such as deploying software components, Kolla uses the Ansible deployment engine (*Ansible* web). Ansible gathers hosts on groups of machines on which specific tasks are applied. This group mechanism in play is very flexible and thus allows alternative deployment topologies to be specified.

10.2.2 Enos Default Benchmarks

Measuring the performance of a cloud infrastructure in a rigorous and comparable way is an important challenge for our community. Although the EnOS abstractions

2 https://github.com/BeyondTheClouds/enos.

allow end-users to plug any custom benchmarks as discussed later on, EnOS comes with two open source benchmarks by default: Rally and Shaker.

Rally

Rally is the official control-plane benchmark suite for OpenStack; it injects API requests to running services. Extensive benchmarks are allowed by two concepts : the *Runner* (*e.g.*, the number of times a request is performed or how many parallel threads are used to perform the requests), the *Context* (*e.g.*, how many users and tenants must be used for the injection)

Shaker

Shaker is a framework for data plane testing of OpenStack. It currently targets synthetic benchmarks execution (for instance iperf3 (*iPerf* web), flent (*Flent* web)) on top of instances. Shaker supports the definition and the deployment of different instances and network topologies. The possible scenarios include extensive evaluation of network capabilities of an OpenStack cloud.

10.2.3 Monitoring and Gathering Metrics

Analysis of OpenStack is mostly based on metrics generated during the experiments and relies on three components: metrics agents, a metrics collector and a metrics visualization service. Those components are loosely coupled, allowing for alternatives to be plugged in when necessary. In the current implementation, metric agents are cAdvisor (*cAdvisor* web) and collectd (*collectd* web). They are responsible for sending metrics from hosts to the collector. Metrics can be enabled or disabled at will through the metrics agents configuration files. The Metrics collector relies on the InfluxDB timeseries optimized database (*InfluxDB* web). Finally, the visualization is delivered by Grafana (*Grafana* web), a dashboard composer.

10.3 EnOS

Evaluating the OpenStack software suite can be divided into four logical phases. The first phase consists in getting raw resources; the second one deploys and initializes the selected version of OpenStack over these resources; the third invokes the benchmarks to be executed; finally, the fourth analyzes results of the evaluation. To help in tackling all these phases, we developed EnOS (Cherrueau et al., 2017; Cherrueau et al., 2016), a holistic approach for the evaluation of OpenStack. After presenting the resource description language that has been designed to configure EnOS, this section describes how each phase has been implemented. In particular, how EnOS can address performance evaluations for any infrastructure by abstracting fundamental principles of each phase.

10.3.1 EnOS Description Language for Flexible Topologies

The description of the resources to acquire as well as the mapping of the different services on top of those resources is made with a YAML resource description language. This language offers a very flexible mechanism that lets EnOS end-users specify and evaluate OpenStack performance over a large set of topologies. However, OpenStack is made of numerous services and writing this description is tedious. For this reason, EnOS reuses Kolla service groups to gather many OpenStack services under the same logical name, which drastically reduces the description size. For instance, the small description in Figure 52a describes a simple deployment topology. This description

Figure 52: *EnOS Resources Description Examples*

```
resources:
  network: 1
  compute: 2
```

(a) *General description*

```
resources:
  paravance:
    network: 1
  econome:
    compute: 2
```

(b) *Extended version for Grid'5000*

says: *"provide one resource for hosting network services and two others for hosting compute services"*.

In the context of EnOS, a resource is anything running a Docker daemon and that EnOS can SSH to. This could be a bare-metal machine, a virtual machine, or a container resource according to the testbed used for conducting the experiments.

Moreover, we emphasize that the language is resource provider dependent in order to handle infrastructure specificities. For instance, on Grid'5000, the language has been extended to specify the name of physical clusters where resources should be acquired, as depicted in Figure 52b. In this description, the paravance cluster (located in Rennes) will provide resources for the network services and the econome cluster (located in Nantes) will provide resources for the compute nodes.

Isolating a service on a dedicated resource is as simple as adding its name to the description. For instance, adding `rabbitmq: 1` at the end of the description on Figure 52a tells EnOS to acquire a dedicated resource for the AMQP bus. Henceforth, the bus will no longer be part of the network resource but deployed on a separate resource at the deployment phase. Obviously, it is possible to do the same for the database, `nova-api`, `glance`, `neutron-server`...

Scaling a service simply requires to increase the number of resources allocated to this service into the description. For instance, increasing the value of `rabbitmq: 1` to `rabbitmq: 3` tells EnOS to acquire three dedicated resources for the AMQP bus. Henceforth, the deployment phase will deploy a cluster composed of three RabbitMQ.

These two characteristics of the language allow a very flexible mechanism to both isolate and scale services.

10.3.2 EnOS Workflow

The following describes the four steps that are achieved by EnOS.

Getting Resources Phase

Calling `enos up` launches the first phase that acquires the resources necessary for the deployment of OpenStack. To get these resources, EnOS relies on the aforementioned description and the notion of *provider*. A provider implements how to get resources on a specific infrastructure and thus makes this job abstract to EnOS. With such mechanism, an operator can easily evaluate OpenStack over any kind of infrastructure by implementing the related provider. A provider can also be given by the support team of an infrastructure, independently of any particular OpenStack evaluation project. In other words for each testbed, an extended version of the EnOS DSL and a *provider* should be available. Currently, EnOS supports two kinds of infrastructure; the first one gets bare-metal resources from the Grid'5000 testbed ; the second one uses a VM based on Vagrant (*Vagrant* web).

We emphasize that additional drivers for any other system can be easily implemented in less than 500 lines of Python code.

The output of the first phase is a list of addresses which reference resources, together with the name of the OpenStack services to deploy over each resource. This way, EnOS will be able to initiate a SSH connection to these resources during the next phase and deploy the requested OpenStack services.

Deploying and Initializing OpenStack Phase

Calling `enos init` deploys and initializes OpenStack with Kolla. Concretely, EnOS uses the list of resources and services provided by the previous phase and writes them into a file called the *inventory file*. Kolla then uses this file to deploy, in a containerized manner, OpenStack services on the correct resources.

The Kolla tool runs each OpenStack service in an isolated container which presents a huge advantage for collecting metrics such as CPU, memory, and network utilization. Indeed, in addition to isolation, container technologies offer fine-grained resource management and monitoring capabilities (Xavier et al., 2013). This means it is possible to collect the current resource usage and performance information, whatever the container runs, through a standard API, and hence offers to EnOS a generic metrics collection mechanism that stands for every OpenStack service. Under the hood, EnOS relies on *cAdvisor* (See Section 10.2.3) to implement this generic collection mechanism.

Running Performance Evaluation Phase

Calling `enos bench` injects workloads to stress the platform. By default, EnOS comes with Rally and Shaker frameworks. However, the EnOS abstractions allow end-users to plug any custom benchmarks like for instance the recent SPEC Cloud Benchmark.[3].

A workload in EnOS is composed of scenarios that will be run in sequence. Each scenario description is specific to the underlying benchmarking tool but EnOS will know how to run it. In the case of Shaker or Rally parameterized scenarios are possible adding more flexibility to the execution.

Analysing the Evaluation Phase

Calling `enos backup` generates all components needed for the analyses of the performance evaluation.

Metrics gathering is twofold. First, EnOS collects general metrics (CPU/memory usage, network utilization, opened sockets…). Second, it is able to store specific statistics offered by the benchmarking suite used. The former relies on a set of agents whose role is to send metrics to a collector. The latter is specific to the benchmarking suite that is executed and occurs during the `backup` phase. Similarly to the previous section, integrating custom benchmarking tools may require extending EnOS to retrieve the relevant reports.

EnOS allows general metrics to be observed in real-time during the experiment. Preconfigured dashboards are indeed accessible through a Web interface. EnOS's `backup` gathers a larger source of information since they include configuration files, logs of OpenStack services, all the collected metrics and all the reports generated by the benchmarking suite used. EnOS can then build a virtual machine image embedding all these data and tools to allow post-mortem exploration.

10.4 Experiments

To illustrate how operators and developers can use EnOS to identify limiting services through the exploration of general metrics, we conducted a first series of evaluations to study how OpenStack behaves when the number of compute nodes scales up to 1,000. Experiments have been executed on the Grid'5000 *paravance* cluster.

We deploy an OpenStack cloud multiple times with EnOS and vary the number of compute nodes from 100 to 1,000 between two deployments. The OpenStack code was based on the Mitaka release. The "fake driver" was chosen as virtualisation driver at the compute level to increase the density of compute nodes on a single physical machine. We achieve to deploy 1,000 *nova-compute* "fake driver" on 20 physical machines.

3 SPEC Cloud™ IaaS 2016

Table 10: *Average CPU usage of OpenStack services while varying the number of compute nodes (in number of cores).*

Nb. of compute nodes	100	200	500	1,000
Nova Conductor	1.22	2.00	3.68	7.00
HAProxy	0.11	0.18	0.33	0.49
RabbitMQ	0.98	1.65	3.11	5.00
MariaDB	0.03	0.06	0.13	0.21

Table 11: *Maximum RAM usage of OpenStack services while varying the number of compute nodes in megabytes.*

Nb. of compute nodes	100	200	500	1,000
Nova Conductor	2.47	2.47	2.45	2.47
HAProxy	6.27	6.32	7.04	8.71
RabbitMQ	1,628	2,580	5202	11,520
MariaDB	502	546	570	594

Two other physical machines were used to host Control and Neutron groups respectively. The fake driver is an in-memory hypervisor that performs mostly the same routine tasks to maintain the state of its local –fake– instances except for *neutron-server*. The latter service is thus out of the scope of the current evaluation.

Metrics are collected for one hour without performing any request on the deployed system. EnOS enables us to inspect these metrics per service. Table 10 and 11 present respectively the CPU and RAM consumption of representative services during this idle period. The observed CPU consumption is very small, except for the Nova Conductor service that interfaces all Nova services with the MariaDB database. This information is valuable for the OpenStack community as it clearly shows that there is room for improvement to reduce the consumption of the `nova-conductor` service (for 1,000 nodes, the current code requires the equivalent of 7 cores while the compute nodes are idle). For the RAM consumption, an important increase is observed for RabbitMQ that is heavily used for communications between services like `nova-conductor` and `nova-compute`. Table 12 presents the maximum number of connections for RabbitMQ and MariaDB. It clearly shows that the increased RAM usage is linked to network usage: the number of open connections on the RabbitMQ container grows indeed at the same rate as memory usage. Moreover, the number of connections opened by RabbitMQ can be explained by the fact that, even in idle state, OpenStack is maintaining several permanent connections with each Nova and Neutron agents. This leads to the conclusion that RabbitMQ will be hard to scale beyond this limit without reviewing the communication patterns in use. To further explain this increase in resource usage, we export from EnOS the number of database queries performed each second by MariaDB.

Table 12: *Maximum number of simultaneous open connections for OpenStack services while varying the number of compute nodes (thousands).*

Nb. of compute nodes	100	200	500	1,000
RabbitMQ	1.5	2.93	6.89	13.5
MariaDB	79	85	120	170

The number of SELECT queries performed each second for the one-hour period is plotted in Figure 53. We observe that the average number of queries increases linearly with the number of nodes. More importantly, from the figure, we observe periodic spikes. These spikes are due to periodic tasks run by Nova services and Neutron agents. They are indeed reporting periodically their states in the database. UPDATE queries follow the same pattern but are not plotted here. Note that the reporting interval is configurable and may be decreased in the configuration file at the cost of decreasing the consistency of the state stored in the database.

This evaluation demonstrates how OpenStack can be studied with EnOS as a black-box and how complex mechanisms involving multiple services can be explored.

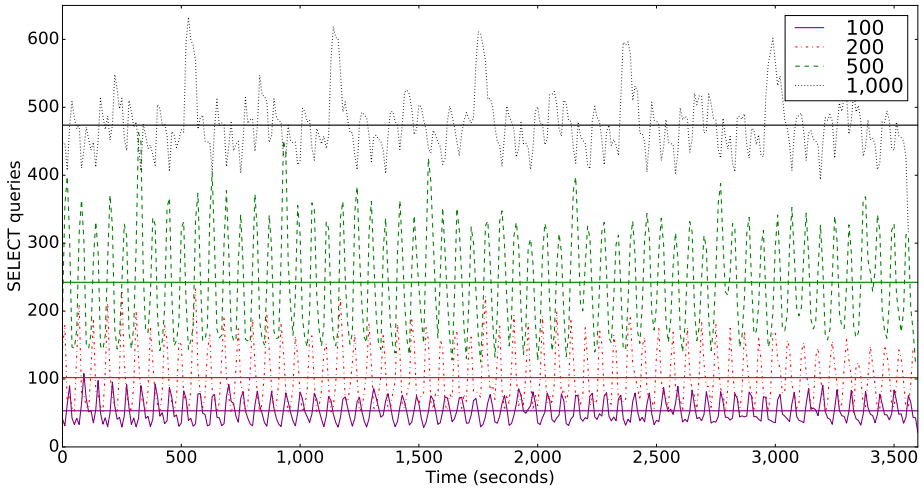


Figure 53: *Number of SQL queries per second executed by MariaDB while varying the number of compute nodes. Horizontal lines show the average for each series.*

10.5 Related Work

The evaluation of OpenStack can be achieved either by the control or the data plane side.

As previously highlighted, several control plane evaluations have been performed (Litvinski and Gherbi, 2013; Schmidt et al., 2016). However, they have been investigated using ad-hoc frameworks that prevent researchers to reproduce them. For instance, the authors of (Litvinski and Gherbi, 2013) reviewed the behavior of the Nova Scheduler using specific and deprecated tools [4] on Nova Computes.

Data plane evaluations have suffered from the same problem. For instance, different OpenStack network components were compared in (Callegati et al., 2014) using deprecated tools named Rude and Crude [5]. Additionally, an ad-hoc framework based on perl and bash scripts has been proposed for network evaluation (Karacali and Tracey, 2016).

Many studies have investigated the challenges of evaluating complex infrastructures such as distributed systems (Buchert et al., 2015) and IaaS clouds (Iosup, Prodan, and Epema, 2014). Four challenges can be extracted from these studies: the ease of experimenting, the replicability (replay an experiment in the same conditions), the reproducibility (experiments can be launched on different infrastructures), the control of the parameter space and the experiment scalability. By embracing the *Experimentation-as-Code* vision and by choosing a pluggable design, EnOS should be able to offer a sustainable method for evaluating OpenStack by tackling these four challenges. Although the current code base only integrate two benchmark suites, namely Rally (*Rally* web) and Shaker (*Shaker* web), attractive tools such as PerfKit (*PerfKit Benchmarker* web) and CloudBench (Silva et al., 2013) can be easily invoked to provide a large panel of synthetic and real-world workloads.

Among the different actions we know, Browbeat (*Browbeat - Performance monitoring and testing of OpenStack* web) is the only OpenStack project whose goals closely match those of EnOS. It provides indeed a set of Ansible playbooks to run workloads on OpenStack and to analyze the result metrics. The workloads are generated by Rally, Shaker or PerfKit Benchmarker (*PerfKit Benchmarker* web) and the metric visualization is done by services such as collectd, grafana or ELK (*Elasticsearch, Logstash, Kibana* web). However, compared to EnOS, Browbeat requires that the operator sets a functional OpenStack with TripleO (*TripleO* web) (*i.e.*, OpenStack On OpenStack). TripleO is an OpenStack project to deploy two clouds. The first one is a deployment cloud (named *undercloud*), and is used to set up tunable workload *overclouds* on which

4 KRASH: Reproducible CPU Load Generation for Linux.
5 RUDE & CRUDE: Real-time UDP Data Emitter & Collector for RUDE

Browbeat runs its benchmarks. This deployment phase adds a significant difficulty for researchers who desire to evaluate OpenStack releases. Moreover, it constraints the researcher to evaluate OpenStack on top of an OpenStack cloud whereas EnOS is testbed agnostic.

10.6 Summary

With a community that gathers more than 5,000 people twice a year at a single location, the OpenStack software suite has become the *de facto* open-source solution to operate, supervise and use Cloud Computing infrastructures. While it has been mainly supported by key companies such as IBM, RedHat and more recently Google, we claim that distributed computing scientists should now join the effort to help the OpenStack consortium address the numerous technical and scientific challenges related to its scalability and reliability. Similarly to what our scientific community has been doing for Linux, the OpenStack software suite should benefit from scientific guidance. However, diving into OpenStack and understanding its intricate internal mechanisms is a tedious and sometimes too expensive task for researchers.

To allow academics, and more generally the OpenStack consortium to identify issues, propose counter-measures, and validate code improvements, we presented in this paper the EnOS framework. Thanks to container technologies and the use of "off-the-shelf" benchmarks, EnOS is the first holistic approach for evaluating OpenStack in a controlled and reproducible way. We illustrated the relevance of EnOS by analyzing how OpenStack behaves at different scales. This experiment helps in identifying services which will become bottlenecks (*e.g.*, RabbitMQ, `nova-conductor`) with a large number of compute nodes. An important aspect for our objective of operating Fog/Edge infrastructures.

Similarly to the previous action (see Chapter 9), this work strengthened our relationships with the OpenStack community. As an example, EnOS has been used to conduct several experiments in the context of the OpenStack performance working group (*OpenStack Performance Documentation* web). Concretely, we evaluated OpenStack workability and performance at large scale by analysing a 1000 nodes cloud emulation on top of Grid'5000. Results have been presented in the Barcelona OpenStack summit in October 2016. **??**Chasing 1000 nodes Scale (OpenStack Summit, Barcelona, Oct 2016)

I should highlight that EnOS has been recently extended in order to allow Researchers/Engineers to enforce network emulation in terms of latency and bandwidth limitations. [6] Thanks to this feature, we can emulate different network topologies (*e.g.*, LAN vs WAN) and evaluate how OpenStack core-services behave. Current experiments are related (i) to the evaluations of different message bus solutions that can replace the current RabbitMQ solution, which does not scale well, and (ii) to identify network requirement in the case of WAN infrastructures. Conducting these experiments is a major concern for the DISCOVERY initiative and more generally for telcos that target the deployment of Fog and Edge Computing infrastructures. We will present major results in the OpenStack Summit in Boston in May 2017. [7]

As mid-term actions, we plan to extend EnOS with the new OpenStack Profiler tool. This will help researchers investigating in details performance issues by analyzing the execution traces of any OpenStack functionality.

The current version of EnOS is available on a public git repository: `https://github.com/BeyondTheClouds/enos`.

Finally, I would like to underline that OpenStack is only one example of such large software projects that can benefit from the involvement of our scientific community. The approach pursued by EnOS can be easily extended to other complex software

6 Enos document - Network Emulation.
7 Toward Fog Edge and NFV Deployments: Evaluating OpenStack WANwide (Boston, May 2017).

stacks. The only requirement is to get containerized versions of said software. This trend is expected to grow, looking for instance at the Docker Hub repository (*Docker Hub* web).

11 *Ongoing and Future Work: A Software Stack for Fog/Edge Infrastructures*

Fog and Edge computing infrastructures can be considered as a new generation of Utility Computing that have been proposed as an alternative to the current Cloud Computing facilities. The main idea is to deploy smaller data-centers at the edge of the backbone in order to bring Cloud Computing resources closer to the end-usages. While a couple of works illustrated the advantages of such infrastructures in particular for Internet of Things, Mobile Edge Computing and Network Function Virtualization applications, the question of how to operate such widely geo-distributed infrastructures still remains opens.

However, designing a well-suited management system is a challenging task because Fog/Edge infrastructures significantly differ from traditional cloud systems regarding heterogeneity, dynamicity and the potential massive distribution of resources and networking environments. While clouds are placed on few sites and rely on high performance servers and networks, Fog/Edge Computing infrastructures are deployed across a significant number of sites. Each site may have its own hardware specifics and network topology with different bandwidth and latencies characteristics. Moreover, a significant part of resources can join and leave the infrastructure according to the service usage, failures, policies and maintenance operations.

Because the deployment of Fog/Edge Computing infrastructures is expected to increase over the next years, conducting research in this area is timely and key for industry and academia worldwide. In particular, it is crucial for our community to quickly propose models, abstractions and algorithms that will enable operators as well as end-users to supervise, and reciprocally take the advantage of, Fog/Edge Computing infrastructures.

In this part, we have presented the vision I have been promoting since 2013 (Bertier et al., 2014) and discussed preliminary results regarding how such a new generation of Utility Computing infrastructure can be deployed and controlled:

- In Chapter 8, we have given an overview of the DISCOVERY project, an open-science initiative I have proposed and that I have been leading for the three last years. The originality of DISCOVERY lies in the way of designing a system in charge of operating a massively geographically distributed cloud. The system we envision can be viewed as a distributed operating system, manipulating Virtual Machines in a geo-distributed infrastructure. The major difference with respect to the state of the art is related to our bottom-up approach: instead of designing a system in charge of orchestrating independent cloud instances (*a.k.a.*, brokering approaches), we target to revise and extend mechanisms of cloud software stacks with P2P and self-* techniques. Using such design principles for establishing the foundation mechanisms of a cooperative cloud management system will be a major breakthrough compared to current management solutions. It is noteworthy that this proposal follows the work we did around DVMS (see Chapter 3). To mitigate the development effort and favor the transfer of our research to the industry, we chose to achieve this by using OpenStack.

The DISCOVERY initiative is being mainly supported since 2015 by Inria and Orange Labs and will last at least until 2019. It gathers about 20 Researchers/Engineers/PhD Candidates that are working on different mechanisms that are mandatory to manipulate virtualized environments WANwide as well as new software abstractions to support the development of new cloud services, which will benefit from Fog/Edge characteristics.

- Chapter 9 presents a proof of concept of a first-class Internet-scale IaaS manager. Specifically, we have described how we revised the Nova service (the OpenStack core-service in charge of managing VMs) with a decentralized key/-value store in place of the centralized SQL database. Although the development guidelines of OpenStack services encourage the shared nothing principle, the message bus and the fact that objects are persisted in shared databases limit the scalabilty of the system. While P2P solutions have been proposed in the literature and a ZeroMQ driver is available in the OpenStack ecosystem, there were no correct solution to distribute the databases. By implementing a dedicated library that enables to replace the SQL backend by a NoSQL one, we succeeded to distribute Nova over several geographical sites. We have discussed several experiments validating the correct behavior of our prototype and have shown promising performance when operating 8 locations with a single decentralized OpenStack instance. AS previously underlined, this work enabled us to exchange with the OpenStack foundation and to take part in different discussions/working groups. Those exchanges conducted us to set up a new working group that deals specifically with the massively distributed use-case.[1]

- Chapter 10 is not directly related to the DISCOVERY target. In fact, it presents an overview of the EnOS framework. A tool we have implemented to make our investigations on OpenStack easier: With a software stack that relies on tens of services, including more than 3 millions lines of code for the core-services alone [2], diving into the OpenStack code is a tedious and often daunting task. EnOS is a free software framework that leverages container technologies and "off-the-shelf" benchmarks for automating reproducible evaluations of OpenStack in a flexible and extensible way. Thanks to EnOS, researchers and developers alike can now evaluate the performance of distinct OpenStack deployment scenarios, and compare the collected results, identify limitations, propose counter-measures, and validate code improvements. We illustrated the relevance of EnOS through a first series of experiments on top of Grid'5000. Those experiments aimed at analyzing how OpenStack behaves at different scales. This has helped us identify services which will become bottlenecks (e.g., RabbitMQ, nova-conductor) with a large number of compute nodes. To the best of our knowledge, EnOS is the first holistic approach for evaluating OpenStack. That is, it has been designed with the Experimentation-as-Code vision: every step of the experimentation workflow, from the configuration to the results gathering and analysis, can be automated. Although EnOS has been designed with OpenStack in mind, I should underline that the concepts that have been proposed are relevant for any kind of complex software stack. The only requirement is to get containerized versions of said software. This trend is expected to grow, looking for instance at the Docker Hub repository.

Each of these activities is ongoing and will last at least until 2019. It is important to understand that revising a complete software stack such as OpenStack is a huge task that goes far beyond a single researcher. It is actually possible to start at least one Phd on each of the core-services of an IaaS manager and to evaluate proposals by implementing them in the OpenStack framework.

1 https://wiki.openstack.org/wiki/Massively_Distributed_Clouds
2 https://www.openhub.net/p/openstack.

In addition to the short-term perspectives that have been highlighted at the end of each section, I plan to address the following challenges during the next three years:

- Monitoring Services – Delivering a distributed version of cloud management systems will not be sufficient for allowing developers to benefit from all opportunities of massively distributed cloud platforms. In addition to locality considerations, a Fog/Edge cloud manager should offer the possibility to monitor contextual information that might be available on the different locations. Concretely, developers should be able to program and deploy their own probes to monitor and collect all the information that might be relevant for them. Such information is valuable for specifying deployment/reconfiguration constraints and to develop advanced adaptation scenarios in order to satisfy for instance the so well-known Service Level agreements. Monitoring large-scale infrastructures is an important challenge of distributed computing and several solutions have been proposed (Fatema et al., 2014). However, none of them has been designed to fit the Fog/Edge Computing requirements. In addition to being static in the sense that you cannot reconfigure them on demand, they rely on centralized persistent storage systems that are not suited to a massively distributed infrastructure such as the ones we target. Through a Phd Grant obtained in the context of Inria/Orange joint Lab, we have been working on this challenge since 2016. The Phd subject consists in proposing a framework in charge of managing the probes life cycle (deployment, configuration, collect of information and removal) for Fog/Edge Computing infrastructures. The objective is twofold because in addition to enriching the capabilities from the developer point of view, we expect to use such a service for the internal mechanisms of the revised version of OpenStack designed in the context of DISCOVERY.

- Installation and Upgrade Process – As discussed, an IaaS manager such as OpenStack is a complex and tedious ecosystem composed of tens of services including millions of lines of code released on 6-month cycles. In a Fog/Edge context where those services will be distributed throughout all sites –like we did with Nova– it will become impractical for administrators to deploy and upgrade the services on-demand and with the current System Configuration Tools (Delaet, Joosen, and Van Brabant, 2010). Hence models and mechanisms for deployment and reconfiguration of services in an autonomous manner will be paramount. Those systems must be capable of automatically installing and upgrading any service without impacting the execution of hosted applications; this process will require some computation and data to be relocated elsewhere. I recently proposed to address this challenge with Dr. Helene Coullon who recently joined the ASCOLA research group and Dr. Dimitri Pertin, currently Post-doctorate researcher in the DISCOVERY initiative.

- Advanced VM Placement Strategies – Although several contributions have been proposed around the efficient provisioning of VMs on cloud federations (placement/scheduling (Tordsson et al., 2012), the contributions have been limited by either using the least common denominator of the functionality available in different clouds or not being able to interact directly with the native cloud resource management system. Instead, by enabling higher-level services (*i.e.*, at the application level) to interact with lower-level mechanisms (*i.e.*, the internal mechanisms in charge of manipulating physical resources in each site), it will be possible to provide new placement and auto-scaling methods with full control over resource management decisions. This tight integration will also allow dynamic reconfiguration at higher layers, each time lower-level mechanisms detect significant events (disconnections, failures, new resources). Therefore, the techniques to be developed should be able to globally optimize scheduling and scaling decisions based on location of users, data, network and all information related to the physical infrastructure. This work will leverage my previous

activities around (i) the DVMS proposal (Quesnel, Lebre, and Südholt, 2013) (described in Chapter 3) and (ii) the synergy between low level and higher level services (Oliveira et al., 2012) (not addressed in this manuscript). Delivering such high-level *IaaS* service has to the best of my knowledge not been addressed yet. I plan to address this aspect with Dr. Frédéric Desprez and the support of a new post-doctorate researcher that should be hired in the context of the DISCOVERY project by the end of 2017.

- Energy & Sustainability Aspects – By not requiring the construction of new facilities in isolated places – which means building huge structures, resulting in wasted amount of time and energy to bring all the needed resources there – and by leveraging resources closer to end-users, the model envisioned by DISCOVERY represents a serious opportunity to develop a more sustainable Utility Computing model. However, there is to the best of my knowledge, no study that really answer the question whether Fog/Edge infrastructures would enable to reduce the energy footprint of the Internet. Moreover being smaller, Fog/Edge sites are better candidates to renewable energies. Similarly to the follow-the-moon/follow-the sun approach, the use of several sites spread across a large territory will offer opportunities to use various energy sources (solar panels, wind turbines, etc.). This opportunity (Berral et al., 2014), will be enabled by the native capability of our cloud management system to federate distinct sites and accommodate entire DCs going periodically offline. Since the whole infrastructure will be supervised by a single system, implementing advanced load-balancing strategies will be simplified compared to a federated approach. I plan to address these challenges, starting in particular with the energy cost and renewable opportunities analysis with Dr. Anne Cecile Orgerie and the support of a new post-doctorate researcher that should be hired within the DISCOVERY context by the end of 2017. This activity should also be strengthened in the next years through the CPER SeDuCe (a French funding program) supervised by Prof. Jean Marc Menaud from the ASCOLA research group. The SeDuCe project (Sustainable Data Centers: Bring Sun, Wind and Cloud Back Together), aims to design an experimental infrastructure dedicated to the study of data centers with low energy footprint within the Grid'5000 framework.

- Cloud storage – Cloud storage should be revised to mitigate the overhead of transferring data from their sources to the different locations where they are needed. Programmers could for example want to favor a pulling mode, as opposed to a pushing one. Nowadays, data is mostly uploaded to remote clouds without considering whether the induced data movements will be of any use in the future. A challenge for storage services like the famous Amazon S3 (Palankar et al., 2008) (or Swift in the case of OpenStack) is to enable data to stay as close to their source as possible, and be transferred on longer distances only when necessary. Similarly, developers will be able to deliver Hadoop-like strategies where computations are launched close to data sources. Such mechanisms will be shortly mandatory to handle the huge amount of data the Internet of Things will generate.

  Although they have not been presented in this manuscript, I should highlight that I have already started to conduct activities in this area with Dr. Benoit Parrein { (Confais, Lebre, and Parrein, 2016; Confais, Lebre, and Parrein, 2017).

  Moreover, I will strengthen this activity with the new ANR GRECO project (2017-2020). The goal of the GRECO project is to develop a reference resource manager for cloud of things. One of the principal challenges will be to handle the execution context of the environment in which the cloud of things operate. Indeed, unlike classical resource managers, connected devices imply to consider new kinds of networks, execution supports, sensors and new constraints like human interactions. The great mobility and variability of these contexts

make the modeling of the quality of service more difficult. To face this challenge, we intend to innovate in designing scheduling and data management systems that will use machine learning techniques to automatically adapt their behavior to the execution context. The GRECO project is built upon a collaboration between an enterprise (Qarnot Computing) and two french research institutes: the Laboratoire d'Informatique de Grenoble (LIG) and Inria for which I am the main representative. Although all results achieved in GRECO will be beneficial for DISCOVERY (and reciprocally), I will mainly focus on the data management aspect in GRECO.

All these research directions are already well defined with concrete ongoing or planned actions. I propose to address them with academic and industry colleagues coming from either the distributed computing community or the network one. I am convinced that the cloud/network convergence is achievable only by gathering a set of people from diverse backgrounds. More generally, I hope that activities such as the ones conducted within the DISCOVERY framework will contribute to fill the gap between both communities. This connection has already started with the different activities around Software-Defined Networking and Network as a Service (*a.k.a.*, Network Function Virtualization) and may result in a new community dealing with Utility Computing challenges where network and computational concerns are fully integrated. Such a new community may leverage the background of both areas to propose new systems that are more suitable to accommodate the needs of our modern societies. I highlight that I have been co-leading the virtualization and cloud action of the French GDR RSD since 2015. This action aims to favor this cloud/network technology convergence across the French Research community.[3] As previously highlighted, I am also deeply involved in the OpenStack community through different international working groups that deal with NFV and Fog/Edge challenges.

In the mid and long term, my expectation is to work with my colleague through a new research group focusing on challenges related to the management and advanced usages of next generations of Utility Computing infrastructures (*i.e.,*. Fog, Edge, Clouds and beyond), notably by delivering appropriate system abstractions at low and high levels in addition to addressing cross cutting dimensions such as energy or security. The strong separation between network and data centers does not exist anymore. Computers, Smartphones, IoT …, all constitute the modern Internet that is at the heart of our society in all the activities we develop. It changes our way of thinking about the world, working, training, and entertaining. This (r)evolution is made possible by the progressive deployment of sophisticated service infrastructures and its articulation with the various players. Indeed, the platforms that constitute this widely distributed digital environment are complex, heterogeneous, dynamic and of variable dimensions, from units with a few "cores" to computing facilities made up of millions of processing units. They also rely on software components and applications consisting of a wide variety of "bricks", which are assembled according to different needs and make extensive use of virtualization for the mobilized resources(compute, network, storage).

Started in 2015, the goal of DISCOVERY has been defined to propose an unified and advanced cloud management systems. This system should allow operators to deploy and supervise a cloud computing platform across several sites while offering to end-users/developers appropriated abstractions to embrace the opportunities offered by the geo-distribution of resources. However, those objectives should be now extended to go beyond the edge and take into account all devices that will compose next generation of ICT infrastructures, from the IoT sensors up to the largest data centers. Taking into account the whole infrastructure is mandatory to propose an efficient and well-suited software stack that will favor the emergence of new kinds of applications related to the Industrial Internet (smart cities, connected cars, cloud manufacturing, e-health services …), sometimes also referred to as Internet of Skills.

3 https://rsd-cloud.lip6.fr/rescom17/.

Finally, I started this manuscript by discussing the importance of being involved in communities, notably in the Grid'5000 consortium. I want to conclude it also by stressing the importance of such facilities for my research and more generally for all Computer and Network scientists that perform experiment-driven research. Next generations of the Internet will result in a system of systems whose availability, reliability and performance will become major challenges that we will have to face. Like Physicians that rely on dedicated facilities such as the Large Hadron Collider to validate hypothesis, the deployment of a Large-Scale Instrument for the digital science community is proven of utmost importance. Withouth such a facility, it will impossible to assess, compare and validate the scientific results. For a few months, we have been working on a new proposal that will benefit from the experience of the Grid'5000 [4] and FIT [5] communities. Entitled SILECS, this project aims at providing a large instrument for research in distributed computing and networks, which will enable us to tackle all the scientific challenges of Internet of Things, data centers, and the networks connecting them. This instrument will offer a multi-platform experimental infrastructure (HPC, Cloud, Big Data, Software Defined Storage, IoT, wireless, Software Defined Network ...) capable of exploring the infrastructures that will be deployed tomorrow and assist researchers and industrial about how to design, build and operate a multi-scale, robust and safe computer system. The findings produced by the community need to be validated in complex environments where the real physical technology is instrumental to give credibility and raise the level of confidence in the research results. Likewise, the data produced by the experiments are valuable and will be processed and archived in order to support benchmarked and reproducible studies. This concern is well-recognized by the research community and is also related to reproducible research concerns raised and discussed as a hot topic lately.

4 http://www.grid5000.fr
5 https://www.iot-lab.info

Cited Publications of the Author

Balouek, Daniel, Alexandra Carpen Amarie, et al. (2012). "Adding virtualization capabilities to the Grid'5000 testbed." In: *International Conference on Cloud Computing and Services Science*. Springer, pp. 3–20 (cit. on pp. 8, 143).

Balouek, Daniel, Adrien Lebre, and Flavien Quesnel (2013). "Flauncher and DVMS–Deploying and Scheduling Thousands of Virtual Machines on Hundreds of Nodes Distributed Geographically." In: *ACM/IEEE International Scalable Computing Challenge (SCALE 2013), held in conjunction with CCGrid'2013* (cit. on pp. 4, 5, 26, 32, 139).

Bertier, Marin et al. (2014). "Beyond the clouds: How should next generation utility computing infrastructures be designed?" In: *Cloud Computing*. Springer, pp. 325–345 (cit. on pp. 4, 8, 86–88, 119, 143).

Cherrueau, Ronan-Alexandre et al. (2016). *ENOS: a Holistic Framework for Conducting Scientific Evaluations of OpenStack*. Technical Report RT-0485. Inria Rennes Bretagne Atlantique ; Nantes (cit. on p. 111).

Cherrueau, Ronan-Alexandre et al. (2017). "Toward a Holistic Framework for Conducting Scientific Evaluations of OpenStack." In: *Proceedings of the 17th IEEE/ACM International Symposium Cluster Grid and Cloud Computing (short papers track, CCGRID 2017)*, To appear (cit. on pp. 7, 86, 109, 111, 142).

Confais, Bastien, Adrien Lebre, and Benoît Parrein (2016). "Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures." In: *Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2016)*, pp. 294–301 (cit. on p. 122).

Confais, Bastien, Adrien Lebre, and Benoît Parrein (2017). "An Object Store Service for a Fog/Edge Computing Infrastructure based on IPFS and Scale-out NAS." In: *Proceedings of the 1st IEEE International Conference on Fog and Edge Computing (ICFEC 2017, collocated with CCGRID)*, To appear (cit. on pp. 4, 96, 122).

Gallard, Jérôme, Adrien Lebre, and Christine Morin (2010). "Saline: Improving best-effort job management in grids." In: *Proceedings of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2010)*. IEEE, pp. 575–579 (cit. on pp. 7, 143).

Gallard, Jérôme, Adrien Lebre, Christine Morin, et al. (2012). "Architecture for the next generation system management tools." In: *Future Generation Computer Systems* 28.1, pp. 136–146 (cit. on pp. 8, 143).

Hermenier, Fabien, Adrien Lebre, and Jean-Marc Menaud (2010). "Cluster-wide context switch of virtualized jobs." In: *Proceedings of the 4th International ACM Workshop on Virtualization Technologies in Distributed Computing (colocated with HPDC 2010)*. ACM, pp. 658–666 (cit. on pp. 4, 5, 11, 13, 15, 43, 139).

Hirofuchi, Takahiro and Adrien Lebre (2013). "Adding virtual machine abstractions into SimGrid: A first step toward the simulation of infrastructure-as-a-service concerns." In: *Proceedings of the 3rd Internationl Conference on Cloud and Green Computing (CGC 2013)*. IEEE, pp. 175–180 (cit. on p. 4).

Hirofuchi, Takahiro, Laurent Pouilloux, and Adrien Lebre (2013). "Adding a live migration model into simgrid: One more step toward the simulation of infrastructure-as-a-service concerns." In: *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom 2013)*. Vol. 1. IEEE, pp. 96–103 (cit. on p. 4).

Hirofuchi, Takahiro, Laurent Pouilloux, and Adrien Lebre (2015). "SimGrid VM: Virtual Machine Support for a Simulation Framework of Distributed Systems." In: *IEEE Transactions on Cloud Computing Journal* (cit. on pp. 4, 6, 47, 68, 74, 140).

Lebre, Adrien (2010). *How Virtualization Changed the Grid Perspective?* Invited talk - From Grid to Cloud workshop, Ecole Normale Superieure de Lyon, France. (Cit. on pp. 8, 143).

Lebre, Adrien, Paolo Anedda, et al. (2011). "Discovery, beyond the clouds." In: *Euro-Par 2011: Parallel Processing Workshops - Lecture Notes in Computer Science*. Springer, pp. 446–456 (cit. on pp. 4, 44, 87).

Lebre, Adrien, Jonathan Pastor, Anthony Simonet, et al. (2017). "Revising OpenStack to Operate Fog/Edge Computing infrastructures." In: *Proceedings of the 5th IEEE International Conference on Cloud Engineering (IC2E 2017), Vancouver, France*, To appear (cit. on pp. 4, 7, 86, 100, 108, 142).

Lebre, Adrien, Jonathan Pastor, and Mario Südholt (2015). "VMPlaceS: A Generic Tool to Investigate and Compare VM Placement Algorithms." In: *European Conference on Parallel Processing - Lecture Note in Computer Science*. Springer, pp. 317–329 (cit. on pp. 6, 47, 70, 141).

NGuyen, Thuy Linh and Adrien Lebre (2017). "Virtual Machine Boot Time Model." In: *Proceedings of 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2017)*. IEEE, to appear (cit. on pp. 4, 6, 47, 68, 78, 82, 141).

Oliveira, Frederico Alvares de et al. (2012). "Self-management of applications and systems to optimize energy in data centers." In: *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice*. IGI Global, pp. 372–394 (cit. on p. 122).

Pastor, Jonathan et al. (2014). "Locality-Aware Cooperation for VM Scheduling in Distributed Clouds." In: *European Conference on Parallel Processing - Lecture Notes in Computer Science*. Springer, pp. 330–341 (cit. on pp. 4, 5, 11, 139).

Quesnel, Flavien and Adrien Lebre (2011). "Cooperative dynamic scheduling of virtual machines in distributed systems." In: *Euro-Par 2011: Parallel Processing Workshops - Lecture Notes in Computer Science*. Vol. 7156. Springer, pp. 457–466 (cit. on pp. 4, 5, 26, 139).

Quesnel, Flavien, Adrien Lebre, Jonathan Pastor, et al. (2013). "Advanced validation of the dvms approach to fully distributed vm scheduling." In: *Prooceedings of the 11th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA-13)*. IEEE, pp. 1249–1256 (cit. on pp. 4, 5, 11, 26, 30, 31, 39, 43, 139).

Quesnel, Flavien, Adrien Lebre, and Mario Südholt (2013). "Cooperative and reactive scheduling in large-scale virtualized platforms with DVMS." In: *Concurrency and Computation: Practice and Experience* 25.12, pp. 1643–1655 (cit. on pp. 4, 5, 11, 26, 31, 34, 43, 69, 72, 73, 82, 122, 139).

Simonet, Anthony, Adrien Lebre, and Anne-Cécile Orgerie (2016). "Deploying Distributed Cloud Infrastructures: Who and at What Cost?" In: *Proceedings of the Intercloud Workshop 2016 (co-located with IEEE International Conference on Cloud Engineering)*, pp. 178–183 (cit. on pp. 89, 91, 99).

Bibliography

Ahmed, Arif and Ejaz Ahmed (2016). "A survey on mobile edge computing." In: *Proceedings of the 10th IEEE International Conference on Intelligent Systems and Control (ISCO 2016)*, pp. 1–8 (cit. on p. 88).

Anedda, Paolo et al. (2010). "Suspending, migrating and resuming HPC virtual clusters." In: *Future Generation Computer Systems* 26.8, pp. 1063–1072 (cit. on p. 14).

Ansible (web). https://www.ansible.com (cit. on p. 110).

Armbrust, Michael et al. (2010). "A view of cloud computing." In: *Communications of the ACM* 53.4, pp. 50–58 (cit. on p. 4).

Atzori, Luigi, Antonio Iera, and Giacomo Morabito (2010). "The internet of things: A survey." In: *Computer networks* 54.15, pp. 2787–2805 (cit. on pp. 7, 88, 142).

Barbagallo, Donato et al. (2010). "A bio-inspired algorithm for energy optimization in a self-organizing data center." In: *Self-Organizing Architectures*. Springer, pp. 127–151 (cit. on pp. 41, 42).

Barham, Paul et al. (2003). "Xen and the art of virtualization." In: *ACM SIGOPS operating systems review*. Vol. 37-5. ACM, pp. 164–177 (cit. on p. 4).

Barker, Adam et al. (2014). "Academic Cloud Computing Research: Five Pitfalls and Five Opportunities." In: *HotCloud* (cit. on p. 69).

Berral, Josep L et al. (2014). "Building green cloud services at low cost." In: *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS 2014)*. IEEE, pp. 449–460 (cit. on p. 122).

Bloomfield, Victor A (2014). *Using R for Numerical Analysis in Science and Engineering*. Vol. 1. CRC Press (cit. on p. 72).

Bobroff, Norman, Andrzej Kochut, and Kirk Beaty (2007). "Dynamic placement of virtual machines for managing sla violations." In: *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*. IEEE, pp. 119–128 (cit. on pp. 14, 23).

Bolze, Raphaël et al. (2006). "Grid'5000: A large scale and highly reconfigurable experimental grid testbed." In: *International Journal of High Performance Computing Applications* 20.4, pp. 481–494 (cit. on pp. 8, 143).

Bonomi, Flavio et al. (2012). "Fog computing and its role in the internet of things." In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, pp. 13–16 (cit. on pp. 7, 88, 100, 142).

Browbeat - Performance monitoring and testing of OpenStack (web). https://github.com/openstack/browbeat (cit. on p. 115).

Buchert, Tomasz et al. (2015). "A survey of general-purpose experiment management tools for distributed systems." In: *Future Generation Computer Systems* 45, pp. 1–12. URL: https://hal.inria.fr/hal-01087519 (cit. on p. 115).

Buyya, Rajkumar and Manzur Murshed (2002). "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing." In: *Concurrency and computation: practice and experience* 14.13-15, pp. 1175–1220 (cit. on p. 81).

Buyya, Rajkumar, Rajiv Ranjan, and Rodrigo N Calheiros (2010). "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services." In: *Proocedings of the International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10)*. Springer, pp. 13–31 (cit. on pp. 88, 91).

cAdvisor (web). https://github.com/google/cadvisor (cit. on p. 111).

Calheiros, Rodrigo N et al. (2011). "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algo-

rithms." In: *Software: Practice and experience* 41.1, pp. 23–50 (cit. on pp. 44, 50, 66, 78, 81).
Callegati, Franco et al. (2014). "Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case." In: *Proceedings of the 3rd IEEE International Conference on Cloud Networking (CloudNet 2014)*. IEEE, pp. 132–137 (cit. on p. 115).
Cartlidge, John and Dave Cliff (2013). "Comparison of Cloud Middleware Protocols and Subscription Network Topologies using CReST, the Cloud Research Simulation Toolkit-The Three Truths of Cloud Computing are: Hardware Fails, Software has Bugs, and People Make Mistakes." In: *CLOSER*, pp. 58–68 (cit. on p. 78).
Casanova, Henri, Arnaud Legrand, and Martin Quinson (2008). "Simgrid: A generic framework for large-scale distributed experiments." In: *Proceedings of the 10th International Conference on Computer Modeling and Simulation (UKSIM 2008)*. IEEE, pp. 126–131 (cit. on pp. 6, 44, 50, 62, 81, 140).
Cascading OpenStack (web). https://wiki.openstack.org/wiki/OpenStack_cascading_solution (cit. on p. 92).
Scaling solutions for OpenStack (web). http://docs.openstack.org/openstack-ops/content/scaling.html (cit. on p. 92).
Church, Kenneth, Albert G Greenberg, and James R Hamilton (2008). "On Delivering Embarrassingly Distributed Cloud Services." In: *HotNets, Usenix*. Citeseer, pp. 55–60 (cit. on p. 94).
Clark, Christopher et al. (2005). "Live Migration of Virtual Machines." In: *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, pp. 273–286 (cit. on pp. 13–15, 54).
CloudStack (website). http://cloudstack.apache.org (cit. on pp. 43, 44, 69, 109).
collectd (web). https://collectd.org/ (cit. on p. 111).
Dabek, Frank et al. (2004). "Vivaldi: A decentralized network coordinate system." In: *ACM SIGCOMM Computer Communication Review*. Vol. 34. 4. ACM, pp. 15–26 (cit. on pp. 5, 36, 40, 140).
David Cappuccio Gartner, Inc. (2015). *Apply a Self-Contained Solution to Micro Data Centers*. REPORT G00268769 (cit. on p. 88).
DeCandia, Giuseppe et al. (2007). "Dynamo: amazon's highly available key-value store." In: *ACM SIGOPS operating systems review* 41.6, pp. 205–220 (cit. on pp. 34, 92).
Delaet, Thomas, Wouter Joosen, and Bart Van Brabant (2010). "A Survey of System Configuration Tools." In: *Proceedings of the 24th Large Installation System Administration Conference (LISA 2010)*. Vol. 10. USENIX, pp. 1–8 (cit. on p. 121).
Docker Hub (web). https://hub.docker.com/explore/ (cit. on p. 117).
Elasticsearch, Logstash, Kibana (web). https://www.elastic.co (cit. on p. 115).
Etsion, Yoav and Dan Tsafrir (2005). "A short survey of commercial cluster batch schedulers." In: *School of Computer Science and Engineering, The Hebrew University of Jerusalem* 44221, pp. 2005–13 (cit. on p. 14).
Fallenbeck, Niels et al. (2006). "Xen and the art of cluster scheduling." In: *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. IEEE Computer Society, p. 4 (cit. on pp. 4, 14, 22).
Farahnakian, Fahimeh et al. (2014). "Hierarchical vm management architecture for cloud data centers." In: *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014)*. IEEE, pp. 306–311 (cit. on p. 92).
Fatema, Kaniz et al. (2014). "A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives." In: *Journal of Parallel and Distributed Computing* 74.10, pp. 2918–2933 (cit. on p. 121).
Feitelson, Dror G. et al. (1997). "Theory and Practice in Parallel Job Scheduling." In: *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, pp. 1–34 (cit. on p. 13).
Feller, Eugen, Christine Morin, and Armel Esnault (2012). "A case for fully decentralized dynamic VM consolidation in clouds." In: *Proceedings of the 4th International*

Conference on Cloud Computing Technology and Science (CloudCom 2012). IEEE. Taiwan, pp. 26–33 (cit. on pp. 34, 41).

Feller, Eugen, Louis Rilling, and Christine Morin (2012). "Snooze: A scalable and autonomic virtual machine management framework for private clouds." In: *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, pp. 482–489 (cit. on pp. 26, 41, 69, 72, 73, 82, 92).

Feller, Eugen, Louis Rilling, Christine Morin, et al. (2010). "Snooze: a scalable, fault-tolerant and distributed consolidation manager for large-scale clusters." In: *Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications (GreenComm 2010)*. IEEE Computer Society, pp. 125–132 (cit. on p. 41).

Fernando, Niroshinie, Seng W Loke, and Wenny Rahayu (2013). "Mobile cloud computing: A survey." In: *Future generation computer systems* 29.1, pp. 84–106 (cit. on pp. 7, 142).

Flent (web). https://flent.org/ (cit. on p. 111).

Foster, Ian and Carl Kesselman (2010). "The History of the grid." In: *Computing* 20.21, p. 22 (cit. on p. 4).

Foster, Ian, Yong Zhao, et al. (2008). "Cloud computing and grid computing 360-degree compared." In: *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, pp. 1–10 (cit. on p. 4).

Frumkin, Michael and Rob F Van der Wijngaart (2002). "Nas grid benchmarks: A tool for grid space exploration." In: *Cluster Computing* 5.3, pp. 247–255 (cit. on p. 20).

Garces-Erice, Luis et al. (2003). "Hierarchical peer-to-peer systems." In: *Parallel Processing Letters* 13, pp. 643–657 (cit. on p. 35).

Garcia Lopez, Pedro et al. (2015). "Edge-centric Computing: Vision and Challenges." In: *ACM SIGCOMM Computer Communication Review* 45.5, pp. 37–42 (cit. on p. 88).

Gartner, Inc. (2014). *The Impact of the Internet of Things on Data Centers*. REPORT (cit. on p. 88).

Gary Cook, Jodie Van Horn (2013). *How Dirty is Your Data ?* Greenpeace International Report (cit. on pp. 7, 87, 142).

Grafana (web). http://grafana.org/ (cit. on p. 111).

Greenberg, Albert et al. (2008). "The Cost of a Cloud: Research Problems in Data Center Networks." In: *ACM SIGCOMM Computer Communication Review* 39.1, pp. 68–73 (cit. on pp. 88, 91, 94, 99).

Grit, Laura, David Irwin, Varun Marupadi, et al. (2007). "Harnessing virtual machine resource control for job management." In: *Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing (VTDC)*. Citeseer (cit. on p. 22).

Grit, Laura, David Irwin, Aydan Yumerefendi, et al. (2006). "Virtual machine hosting for networked clusters: Building the foundations for" autonomic" orchestration." In: *Proceedings of the 1st International workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*. IEEE, pp. 7–7 (cit. on pp. 14, 17, 22).

Hermenier, Fabien, Sophie Demassey, and Xavier Lorca (2011). "Bin repacking scheduling in virtualized datacenters." In: *Proceedings of the 17th international conference on Principles and practice of constraint programming*. CP'11. Perugia, Italy: Springer-Verlag, pp. 27–41 (cit. on pp. 31, 72).

Hermenier, Fabien, Julia Lawall, and Gilles Muller (2013). "Btrplace: A flexible consolidation manager for highly available applications." In: *Transactions on dependable and Secure Computing* 10.5, pp. 273–286 (cit. on pp. 24, 31, 139).

Hermenier, Fabien, Xavier Lorca, et al. (2009). "Entropy: a consolidation manager for clusters." In: *Proceedings of the 2009 ACM SIGPLAN SIGOPS international conference on Virtual execution environments*. ACM, pp. 41–50 (cit. on pp. 5, 6, 13, 14, 16, 17, 20, 25, 43, 69, 72, 82, 139).

Hintjens, Pieter (2013). *ZeroMQ: Messaging for Many Applications*. " O'Reilly Media, Inc." (cit. on p. 101).
Hirofuchi, Takahiro, Hidemoto Nakada, et al. (2012). "Reactive cloud: Consolidating virtual machines with postcopy live migration." In: *Information and Media Technologies* 7, pp. 614–626 (cit. on p. 61).
Houidi, Ines et al. (2011). "Cloud service delivery across multiple cloud platforms." In: *Proceedings of the IEEE International Conference on Services Computing (SCC 2011)*, IEEE, pp. 741–742 (cit. on p. 91).
Huebscher, Markus C and Julie A McCann (2008). "A survey of autonomic computing—degrees, models, and applications." In: *ACM Computing Surveys* 40.3, p. 7 (cit. on pp. 5, 139).
Imbert, Matthieu et al. (2013). "Using the EXECO toolbox to perform automatic and reproducible cloud experiments." In: *Proceedings of the 1st International Workshop on UsiNg and building ClOud Testbeds (UNICO, collocated with IEEE CloudCom 2013*. Bristol (cit. on pp. 32, 61, 104).
InfluxDB (web). https://www.influxdata.com (cit. on p. 111).
Iosup, Alexandru, Radu Prodan, and Dick Epema (2014). "Iaas cloud benchmarking: approaches, challenges, and experience." In: *Cloud Computing for Data-Intensive Applications*. Springer, pp. 83–104 (cit. on p. 115).
iPerf (web). https://iperf.fr/ (cit. on p. 111).
Irwin, David E et al. (2006). "Sharing Networked Resources with Brokered Leases." In: *USENIX Annual Technical Conference, General Track*, pp. 199–212 (cit. on p. 22).
Jelasity, Márk and Ozalp Babaoglu (2005). "T-Man: Gossip-based overlay topology management." In: *International Workshop on Engineering Self-Organising Applications*. Lecture Notes in Articial Intelligence. Springer, pp. 1–15 (cit. on p. 36).
Johnson, David S (1973). "Near-optimal bin packing algorithms." PhD thesis. Massachusetts Institute of Technology (cit. on p. 83).
Jussien, Narendra, Guillaume Rochart, and Xavier Lorca (2008). "Choco: an open source java constraint programming library." In: *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pp. 1–10 (cit. on p. 19).
Karacali, Bengi and John M Tracey (2016). "Experiences evaluating openstack network data plane performance and scalability." In: *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*. IEEE, pp. 901–906 (cit. on p. 115).
Kemme, Bettina and Gustavo Alonso (2010). "Database replication: a tale of research across communities." In: *Proceedings of the VLDB Endowment* 3.1-2, pp. 5–12 (cit. on p. 102).
Khanna, Gunjan et al. (2006). "Application performance management in virtualized server environments." In: *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*. IEEE, pp. 373–381 (cit. on p. 23).
Kherbache, Vincent, Eric Madelaine, and Fabien Hermenier (2015). "Scheduling Live-Migrations for Fast, Adaptable and Energy-Efficient Relocation Operations." In: *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*. IEEE, pp. 205–216 (cit. on pp. 49, 68).
Kivity, Avi et al. (2007). "kvm: the Linux virtual machine monitor." In: *Proceedings of the Linux symposium*. Vol. 1, pp. 225–230 (cit. on p. 54).
Kliazovich, Dzmitry et al. (2010). "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers." In: *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, pp. 1–5 (cit. on pp. 67, 78).
Kotsovinos, Evangelos (2011). "Virtualization: Blessing or curse?" In: *Communications of the ACM* 54.1, pp. 61–65 (cit. on p. 26).
Krauter, Klaus, Rajkumar Buyya, and Muthucumaru Maheswaran (2002). "A taxonomy and survey of grid resource management systems for distributed computing." In: *Software: Practice and Experience* 32.2, pp. 135–164 (cit. on p. 13).

Lakshman, Avinash and Prashant Malik (2010). "Cassandra: A Decentralized Structured Storage System." In: *ACM SIGOPS Operating Systems Review* 44.2, pp. 35–40 (cit. on p. 108).

Lebre, Adrien, Arnaud Legrand, et al. (2015). "Adding storage simulation capacities to the simgrid toolkit: Concepts, models, and api." In: *Proceedings of the IEEE/ACM Symposium on Cluster, Cloud and Grid Computing (CCGrid 2015)*. IEEE, pp. 251–260 (cit. on pp. 6, 141).

libvirt: The virtualization API. http://libvirt.org (cit. on pp. 55, 58, 64).

Litvinski, Oleg and Abdelouahed Gherbi (2013). "Openstack scheduler evaluation using design of experiment approach." In: *Proceedings of the 16th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2013)*. IEEE, pp. 1–7 (cit. on p. 115).

Loutas, Nikolaos et al. (2010). "Towards a reference architecture for semantically interoperable clouds." In: *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010)*. IEEE, pp. 143–150 (cit. on p. 91).

Low, Chinyao, Yahsueh Chen, and Mingchang Wu (2011). "Understanding the determinants of cloud computing adoption." In: *Industrial management & data systems* 111.7, pp. 1006–1023 (cit. on p. 4).

Lowe, Scott (2010). *Mastering VMware vSphere 4*. John Wiley & Sons (cit. on pp. 26, 43).

Marzolla, Moreno, Ozalp Babaoglu, and Fabio Panzieri (2011). "Server consolidation in clouds through gossiping." In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*. IEEE, pp. 1–6 (cit. on pp. 41, 42).

Mastroianni, Carlo, Michela Meo, and Giuseppe Papuzzo. "Self-economy in cloud data centers: Statistical assignment and migration of virtual machines." In: *Proceedings of the 17th European Conference on Parallel and Distributed Computing (Euro-Par'11)* (cit. on p. 41).

Mijumbi, Rashid et al. (2015). "Network function virtualization: State-of-the-art and research challenges." In: *IEEE Communications Surveys & Tutorials* 18.1, pp. 236–262 (cit. on p. 88).

Mijumbi, Rashid et al. (2016). "Network function virtualization: State-of-the-art and research challenges." In: *IEEE Communications Surveys & Tutorials* 18.1, pp. 236–262 (cit. on pp. 7, 142).

Mills, Kevin, James Filliben, and Christopher Dabrowski (2011). "Comparing vm-placement algorithms for on-demand clouds." In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, pp. 91–98 (cit. on p. 67).

Montresor, Alberto and Márk Jelasity (2009). "PeerSim: A scalable P2P simulator." In: *Proceedings of the 9th International Conference on Peer-to-Peer Computing (P2P'09)*. IEEE, pp. 99–100 (cit. on pp. 44, 50).

Moreno-Vozmediano, Rafael, Rubén S Montero, and Ignacio M Llorente (2012). "Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures." In: *Computer* 45.12, pp. 65–72 (cit. on pp. 69, 92).

Núñez, Alberto et al. (2012). "iCanCloud: A flexible and scalable cloud infrastructure simulator." In: *Journal of Grid Computing* 10.1, pp. 185–209 (cit. on pp. 67, 78).

Nurmi, Daniel et al. (2009). "The eucalyptus open-source cloud-computing system." In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, pp. 124–131 (cit. on p. 67).

OpenNebula (website). https://opennebula.org (cit. on pp. 43, 44, 69, 109).

OpenStack (website). http://www.openstack.org (cit. on pp. 43, 44, 69, 88, 93, 100, 109).

Kolla (web). https://wiki.openstack.org/wiki/Kolla (cit. on p. 110).

Rally (web). https://wiki.openstack.org/wiki/Rally (cit. on pp. 110, 115).

Shaker (web). https://github.com/openstack/shaker (cit. on pp. 110, 115).

Palankar, Mayur R et al. (2008). "Amazon S3 for science grids: a viable solution?" In: *Proceedings of the 2008 international workshop on Data-aware distributed computing*. ACM, pp. 55–64 (cit. on p. 122).
Pavlo, Andrew and Matthew Aslett (2016). "What's Really New with NewSQL?" In: *ACM SIGMOD Record* 45.2, pp. 45–55 (cit. on p. 108).
Peng, Junjie et al. (2009). "Comparison of several cloud computing platforms." In: *Proceedings of the 2nd International Symposium on Information Science and Engineering (ISISE 2009)*. IEEE, pp. 23–27 (cit. on p. 92).
PerfKit Benchmarker (web). http://googlecloudplatform.github.io/PerfKitBenchmarker/ (cit. on p. 115).
OpenStack Performance Documentation (web). http://docs.openstack.org/developer/performance-docs/ (cit. on p. 116).
Popek, Gerald J and Robert P Goldberg (1974). "Formal requirements for virtualizable third generation architectures." In: *Communications of the ACM* 17.7, pp. 412–421 (cit. on p. 4).
Ratnasamy, Sylvia et al. (2001). "A scalable content-addressable network." In: *Proceedings of the International Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*. San Diego, California, USA: ACM, pp. 161–172 (cit. on p. 27).
Rossi, Francesca, Peter Van Beek, and Toby Walsh (2006). *Handbook of constraint programming*. Elsevier (cit. on pp. 5, 19, 50, 139).
Rouzaud-Cornabas, Jonathan (2010). "A distributed and collaborative dynamic load balancer for virtual machine." In: *Euro-Par 2010: Parallel Processing Workshops - Lecture Notes in Computer Science*. Springer, pp. 641–648 (cit. on pp. 26, 41).
Rowstron, Antony and Peter Druschel (2001). "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems." In: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Vol. 2218. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 329–350 (cit. on pp. 27, 35).
Ruth, Paul et al. (2006). "Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure." In: *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'06)*. IEEE, pp. 5–14 (cit. on p. 14).
Sarathy, Vijay, Purnendu Narayan, and Rao Mikkilineni (2010). "Next generation cloud computing architecture: Enabling real-time dynamism for shared distributed physical infrastructure." In: *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*. IEEE, pp. 48–53 (cit. on p. 5).
Satyanarayanan, Mahadev et al. (2009). "The case for vm-based cloudlets in mobile computing." In: *IEEE pervasive Computing* 8.4 (cit. on pp. 88, 99).
Schmidt, Anita et al. (2016). "Performance Analysis of an OpenStack Private Cloud." In: *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016)*, pp. 282–289 (cit. on p. 115).
Schwarzkopf, Malte et al. (2013). "Omega: flexible, scalable schedulers for large compute clusters." In: *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, pp. 351–364 (cit. on p. 77).
Silva, Marcio et al. (2013). "Cloudbench: Experiment automation for cloud environments." In: *Proceeding of the IEEE International Conference on Cloud Engineering (IC2E 2013)*. IEEE, pp. 302–311 (cit. on p. 115).
Snyder, Bruce, Dejan Bosnanac, and Rob Davies (2011). *ActiveMQ in Action*. Manning (cit. on p. 101).
Sotomayor, Borja, Kate Keahey, and Ian Foster (2008). "Combining batch execution and leasing using virtual machines." In: *Proceedings of the 17th international symposium on High performance distributed computing*. ACM, pp. 87–96 (cit. on pp. 4, 8, 14, 22, 143).

Sotomayor, Borja, Rubén Santiago Montero, et al. (2008). "Capacity leasing in cloud systems using the opennebula engine." In: *Workshop on Cloud Computing and its Applications*. Vol. 3 (cit. on p. 22).

Sotomayor, Borja, Rubén S Montero, et al. (2009). "Virtual infrastructure management in private and hybrid clouds." In: *IEEE Internet computing* 13.5 (cit. on pp. 4, 26).

Stoica, I. et al. (2003). "Chord: a scalable peer-to-peer lookup protocol for internet applications." In: *IEEE/ACM Transactions on Networking* 11, pp. 17–32 (cit. on pp. 27, 34, 38).

Svard, Petter et al. (2011). "High performance live migration through dynamic page transfer reordering and compression." In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, pp. 542–548 (cit. on p. 61).

Tordsson, Johan et al. (2012). "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers." In: *Future Generation Computer Systems* 28.2, pp. 358–367 (cit. on p. 121).

Transaction Processing Performance Council (1994). *TPC BENCHMARK B, Standard Specification Revision 2.0*. http://www.tpc.org/tpcb/spec/tpcb_current.pdf (cit. on p. 56).

TripleO (web). http://tripleo.org/ (cit. on p. 115).

Vagrant (web). https://www.vagrantup.com/ (cit. on p. 112).

Vaquero, Luis M, Luis Rodero-Merino, and Rajkumar Buyya (2011). "Dynamically scaling applications in the cloud." In: *ACM SIGCOMM Computer Communication Review* 41.1, pp. 45–52 (cit. on p. 26).

Verma, Akshat, Puneet Ahuja, and Anindya Neogi (2008). "Power-aware dynamic placement of hpc applications." In: *Proceedings of the 22nd annual International Conference on Supercomputing (ICS'08)*. ACM, pp. 175–184 (cit. on p. 14).

Wood, Timothy et al. (2009). "Memory buddies: exploiting page sharing for smart colocation in virtualized data centers." In: *Proceedings of the 2009 ACM SIGPLAN SIGOPS international conference on Virtual execution environments*. ACM, pp. 31–40 (cit. on pp. 14, 17, 23).

Xavier, Miguel G et al. (2013). "Performance evaluation of container-based virtualization for high performance computing environments." In: *Proceedings of the 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2013)*. IEEE, pp. 233–240 (cit. on p. 113).

Xu, Jing, Ming Zhao, and José AB Fortes (2009). "Cooperative autonomic management in dynamic distributed systems." In: *Proceedings of Symposium on Self-Stabilizing Systems*. Springer, pp. 756–770 (cit. on p. 41).

Xu, Zhichen, Mallik Mahalingam, and Magnus Karlsson (2003). "Turning heterogeneity into an advantage in overlay routing." In: *Proceedings of the 32nd IEEE Conference on Computer and Communications (INFOCOM 2003)*. Vol. 2. IEEE, pp. 1499–1509 (cit. on p. 35).

Xu, Zhichen and Zheng Zhang (2002). "Building low-maintenance expressways for p2p systems." In: *Hewlett-Packard Labs, Palo Alto, CA, Tech. Rep. HPL-2002-41* (cit. on p. 35).

Yazir, Yagiz Onat et al. (2010). "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis." In: *Proceedings of the 3rd International Conference on Cloud Computing (CLOUD 2010)*. Ieee, pp. 91–98 (cit. on pp. 26, 41).

Zhang, Ben et al. (2015). "The Cloud is Not Enough: Saving IoT from the Cloud." In: *Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. USENIX Association (cit. on pp. 7, 88, 99, 142).

Part V

APPENDIX

Résumé long en francais

A

A.1 Contexte générale

Afin de satisfaire la demande croissante de calcul et de stockage, les systèmes informatiques n'ont cessé d'évoluer. Des super-ordinateurs (« mainframes ») à l'avènement des infrastructures actuelles dites d'informatique en nuage, les évolutions ont été rythmées par les avancées technologiques réalisées depuis les années 1960 dans le domaine de la conception des ordinateurs bien évidemment mais également dans celui des communications. En effet, parce qu'ils permettent de fournir des capacités au delà de celles pouvant être délivrées par une unique machine, les systèmes informatiques ont évolué dès les années 1990 vers un modèle d'infrastructure distribuée reposant sur une interconnexion de plusieurs machines distinctes. Cependant, de part leur complexité d'administration et d'utilisation, il a fallut attendre près de vingt ans d'activités de recherche et de transferts technologiques pour démocratiser ces infrastructures aussi bien dans les domaines scientifiques qu'industriels.

Parmi les technologies qui ont permis cette adoption massive des infrastructures distribuées, les technologies de virtualisation système ont joué un rôle clé. La virtualisation système peut être vue comme une couche d'abstraction entre le matériel et les logiciels applicatifs. Cette abstraction permet une division des ressources physiques proposées par une machine en plusieurs éléments « virtuels ». Chacun de ces éléments peut être affecté à la volée à chacune des applications en fonction de leurs besoins, offrant ainsi une plus grande flexibilité dans la gestion des ressources disponibles. Parmi les abstractions disponibles, la machine virtuelle est probablement la plus connue. Une machine virtuelle fournit une vision abstraite d'une machine physique avec sa propre pile logicielle (*i.e.*, un système d'exploitation, des bibliothèques et les applicatifs à exécuter). Cette vision plus ou moins réduite des ressources disponibles est rendue possible grâce à un hyperviseur déployé sur la machine physique. Un hyperviseur peut être vue comme un système d'exploitation traditionnel qui exécute des machines virtuelles en place des processus usuels. En d'autres termes, il est en charge de superviser et d'exécuter l'ensemble des machines virtuelles qui sont présentes sur la machine physique. L'utilisation des machines virtuelles offre plusieurs avantages du point de vue des utilisateurs. Tout d'abord, elle permet de configurer la totalité de la pile logicielle présente dans la machine virtuelle et de sauvegarder cette pile logicielle de manière pérenne. Ce mécanisme permet de redémarrer cet environnement « virtualisé » à chaque fois que nécessaire sans se préoccuper de l'endroit où la machine virtuelle sera démarrée. De plus, en s'appuyant sur la granularité offerte par la machine virtuelle pour développer les applications (en particulier des applications n tiers), il est possible d'adapter le nombre de machines virtuelles de manière à satisfaire la qualité de service attendue. Le terme d'élasticité est souvent employé pour décrire ce phénomène d'adaptation. Enfin, l'utilisation des machines virtuelles offre une isolation plus forte entre les différents services qui sont exécutés simultanément sur une même machine physique. En effet, l'hyperviseur doit assurer qu'une machine virtuelle ne puisse en aucun cas accéder aux ressources affectées aux autres machines virtuelles co-localisées sur la même machine physique.

Bien que les technologies de virtualisation aient joué un rôle important dans l'adoption quasi-unanime aujourd'hui du modèle d'informatique en nuage, notamment sur

les solutions « Infrastructure-as-a-Service[1] » (IaaS), elles ont également conduit à de nouveau défis scientifiques et techniques, notamment sur les aspects qui ont trait à la gestion des ressources offertes par l'infrastructure (c.-à-d. l'ensemble des serveurs de calcul, de stockage et des liens réseau) ou encore à la gestion du cycle de vie des machines virtuelles (création, mise en pause, redémarrage et migration) au travers cette même infrastructure.

Dans ce manuscrit je présente une synthèse des principaux travaux que j'ai réalisés autour ces problématiques depuis mon arrivée en 2008 dans l'équipe ASCOLA. Plus précisément, je me concentre sur trois axes dans lesquels j'ai réalisés diverses contributions. Le premier axe aborde la problématique du placement dynamique des machines virtuelles. Le second traite de la mise en place d'abstractions et de modèles exactes permettant l'étude via simulation des défis relatifs à la gestion et l'utilisation de machines virtuelles dans une infrastructure distribuée à large-échelle. Enfin, le troisième et dernier axe décrit une nouvelle génération d'infrastructure visant à corriger les problèmes intrinsèques au modèle d'informatique en nuage reposant sur un nombre limité de centres de données à large échelle. Cet autour de cet axe et des perspectives que je souhaite y donner pour les prochaines années que je conclus ce document.

A.2 Contributions Scientifiques

Bien que chacune des contributions ait été traitée de manière indépendante dans ce manuscrit, l'intersection entre ces travaux est non nulle. Après avoir étudier pendant près de 5 ans la problématique du placement dynamique de machines virtuelles dans des infrastructures distribuées, nous nous sommes confrontés à la difficulté de valider nos propositions à large échelle. Ces validations sont nécessaires au vu de la taille en des infrastructures actuelles (des dizaines de milliers de machines physiques pour des centaines de milliers de machines virtuelles). C'est pour répondre à ce défi que j'ai décidé de m'intéresser aux bibliothèques de simulations et de proposer des extensions nous permettant d'évaluer les différentes propositions algorithmiques de placement de machines virtuelles proposées par la communauté. Abordé la problématique de la gestion des ressources physiques ainsi que des machines virtuelles sous les angles du massivement distribué et du large-échelle m'a emmené à considérer tout d'abord puis à argumenter en faveur d'une nouvelle génération d'informatique utilitaire massivement distribuée.

A.2.1 Placement dynamique de machines virtuelles 2008 - 2015

Les operateurs d'infrastructure en nuage ont rapidement été confrontés à une explosion du nombre de machines virtuelles. En effet, la souplesse apportée par les machines virtuelles était telle que les administrateurs et les développeurs se sont mis à concevoir des applications en considérant la machine virtuelle comme granularité élémentaire. Le défaut de cette approche est qu'elle conduit rapidement à des problèmes de sur-utilisation, et réciproquement de famine, pour certains services (une machine virtuelle exécutant un calcul peu couteux peut utiliser une quantité de ressources qu'elle n'a plus besoin alors que les performances d'un autre service s'exécutant dans une autre machine virtuelle peuvent être dégradées faute de ressources disponibles). Pour limiter ce phénomène, la communauté scientifique a proposé plusieurs solutions en charge de maximiser l'utilisation de l'infrastructure tout en garantissant la qualité de services pour chacune des machines virtuelles. Parmi les différentes systèmes disponibles à mon arrivée dans l'équipe ASCOLA, plusieurs de mes collègues travail-

1 Une infrastructure distribuée où les développeurs/utilisateurs peuvent instancier à la demande des machines virtuelles.

laient sur la proposition Entropy (Hermenier, Lorca, et al., 2009). Articulé autour d'une boucle autonomique (Huebscher and McCann, 2008) et d'un moteur de programmation par contraintes (Rossi, Van Beek, and Walsh, 2006), le système Entropy avait pour objectif d'assurer le meilleur taux de consolidation possible dans un centre de données (*i.e.*,. trouver le nombre minimal de serveurs physiques permettant d'exécuter l'ensemble des machines virtuelles présente dans l'infrastructure tout en garantissant que les ressources demandées soient satisfaites). La première contribution a consisté en une généralisation des concepts de gestion dynamique des machines virtuelles proposées par Entropy. Le mécanisme proposé a permis une gestion du cycle des machines virtuelles au travers une infrastructure distribuée identique à celle du cycle des processus dans un système d'exploitation standard (Hermenier, Lebre, and Menaud, 2010). Concrètement, nous avons proposé le concept de changement de contexte de machines virtuelles à l'échelle d'une grappe informatique. En ne manipulant que des machines virtuelles et en utilisant les fonctionnalités telles que la mise en pause, le redémarrage et la migration à chaud, il a été possible de résoudre de manière transparente les problèmes de violation de la qualité de service tout en maximisant l'utilisation des ressources : la boucle autonomique surveille de manière périodique l'utilisation effective des ressources et reconfigure la grappe de calcul à chaque fois que cela est nécessaire en mettant soit en pause certaines machines virtuelles soit en les relocalisant vers d'autres nœuds physiques. Il est alors possible d'implémenter n'importe quelle politique de placements (consolidation, équilibrage de charge …) et laisser le module de changement de contexte faire les opérations nécessaires pour passer d'une configuration (*i.e.*, un placement de machines virtuelles sur les machines physiques) à une autre).

Plusieurs perspectives ont été identifiées à la suite de ces travaux. Alors qu'une partie de mes collègues s'est concentrée sur la spécification et la prise en compte des dépendances entre les machines virtuelles afin de ne pas appliquer des changements de contextes incorrectes (comme par exemple la mise en place de deux machines virtuelles sur une même nœud dans un contexte de haute disponibilité (Hermenier, Lawall, and Muller, 2013)), j'ai choisi de me concentrer sur la problématique du passage à l'échelle du mécanisme initialement proposé. L'objectif était de pouvoir appliquer des changements de contexte de machines virtuelles sur des infrastructures composées de dizaines de milliers de serveurs physiques (et donc de centaines de milliers de machines virtuelles). Pour y parvenir, nous avons proposé de garder l'approche par contraintes pour le mécanisme de résolution du problème combinatoire et de distribuer le processus autonomique en utilisant notamment des mécanismes « pair à pair ». Nous avons tout d'abord proposé la solution DVMS (Distributed Virtual Machine Scheduler). DVMS permet de placer et de maintenir dynamiquement et de manière coopérative un ensemble de machines virtuelles sur une infrastructure à large échelle (Quesnel and Lebre, 2011; Quesnel, Lebre, and Südholt, 2013; Quesnel, Lebre, Pastor, et al., 2013; Balouek, Lebre, and Quesnel, 2013). DVMS repose sur un algorithme construit au dessus d'un anneau qui permet lorsqu'une machine physique est surchargée (ou sous chargée) de déclencher un processus itératif qui consiste à aller de proche en proche afin de trouver un meilleur placement (si le voisin directe ne peut pas proposer un meilleur placement alors le voisin suivant est interrogé à son tour, e processus itératif prend fin lorsque que un placement correcte et satisfaisant est trouvé). Cette approche permet de limiter l'espace de recherche à son minimum, réduisant ainsi le temps nécessaire pour résoudre le problème combinatoire ainsi que celui pour appliquer le changement de contexte (*i.e.*, l'opération de reconfiguration). La seconde extension que nous avons proposée, s'est concentrée sur les notions de localité. L'objectif était de favoriser les changements de contexte entre les nœuds « proches », en particulier nous souhaitions proposer une solution pour limiter les reconfigurations impliquant des machines physiques en provenance de sites géographiques distants (Pastor et al., 2014). Nous avons développé une abstraction générique qui permet de réifier les notions de localité au niveau de l'algorithme de placement de machines virtuelles. Dans ces travaux, la notion de localité a

été estimée au travers une fonction prenant en compte la latence entre les machines physiques composant notre infrastructure. En prenant en compte cette information, l'algorithme peut effectuer le processus itératif précédemment décrit en sélectionnant les nœuds les plus proches. L'abstraction proposée pour réifier la localité repose sur le protocole « pair à pair » Vivaldi qui permet de faire correspondre une structure logique à une topologie physique tout en préservant cette notion de localité (Dabek et al., 2004). Au dessus de Vivaldi, un algorithme de plus court chemin permet à chaque nœud de parcourir l'infrastructure de proche en proche. Nous avons évalué cette proposition en remplaçant le réseau logique de type anneau précédemment utilisé dans la proposition DVMS avec cette nouvelle brique. Les résultats ont montré que les opérations inter-sites étaient utilisées qu'en dernier recours (c.-à-d., quand il n'y avait aucune configuration permettant de satisfaire les besoins de toutes les machines virtuelles d'un même site et que par conséquent l'ajout de machines physiques extérieures devient nécessaire).

Des perspectives spécifiques à chacun de ces travaux sont évoquées de manière plus précise dans le document. Toutefois, de manière plus générale, il est intéressant de noter qu'ils sont également le point de départ des deux axes de recherches que j'ai étudiés par la suite. Le premier concerne la difficulté de valider et comparer les stratégies de placements disponibles dans la littérature. Le second plus large m'a emmené à m'interroger sur la pertinence des technologies « pair-à-pair » sur l'ensemble des sous-services nécessaire à la supervision d'une infrastructure virtualisée (ou de type informatique en nuage).

Chacune des propositions évoquées ci dessus a donné lieu à des développements open-sources opérationnels : `http://beyondtheclouds.github.io/DVMS/`).

A.2.2 Virtualisation et bibliothèques de simulation 2013 -

Lors de la validation des différentes algorithmes réalisés dans le cadre de la proposition DVMS, nous avons été confrontés d'une part à la difficulté de conduire des tests *in-vivo*[2] à large échelle mais également à celle de pouvoir de réaliser des comparaisons rigoureuses avec les principales solutions proposées par l'état de l'art. En effet, bien que la problématique du placement des machines virtuelles est un sujet important depuis déjà de nombreuses années, la plupart des solutions a été validée soit en s'appuyant sur des simulateurs ad-hoc soit par des évaluations *in-vivo* à des petites échelles. Une telle méthodologie n'est pas suffisante pour permettre d'avancer et de consolider ces efforts. Tout d'abord, elle ne permet pas d'évaluer les algorithmes dans des conditions représentatives des systèmes de production composés d'un nombre de machines physiques important. Ensuite, elle ne permet pas une comparaison juste entre les différentes solutions et donc il est impossible de bien comprendre les avantages et défauts de chacune.

Le second axe sur lequel j'ai choisi de concentrer des efforts concerne l'extension de l'outil SimGrid, une bibliothèque dédiée à l'étude d'algorithmes pour les infrastructures distribuées à large échelle (Casanova, Legrand, and Quinson, 2008). Bien que l'outil soit utilisé depuis plusieurs années dans les domaines comme le calcul à haute performance ou encore l'étude des systèmes « pair-à-pair », SimGrid ne fournissait pas les abstractions nécessaires à l'étude d'algorithmes impliquant des machines virtuelles. Nous avons donc proposé un ensemble d'extensions permettant de démarrer des centaines de milliers de machines virtuelles via simulation et de les manipuler comme dans le monde réel (mise en pause, redémarrage et migration) (Hirofuchi, Pouilloux, and Lebre, 2015). Les utilisateurs peuvent exécuter des calculs ainsi que des communications sur les machines physiques ou virtuelles au travers l'API historique de SimGrid. Cette caractéristique est un avantage pour tous les utilisateurs qui souhaiteraient faire une transition vers les infrastructures de type IaaS.

2 En conditions réels – à la différence de *in siclo* qui signifie par simulation.

Par ailleurs, nous avons intégré un modèle pour la migration à chaud de machines virtuelles. Ce modèle, qui met en œuvre le mécanisme de pré-copie, calcul de manière précise le temps nécessaire pour effectuer une opération de migration ainsi que la quantité de données échangés entre les machines physiques et ce tout en tenant compte des charges relatives aux autres tâches de calcul et de communications s'exécutant sur l'infrastructure simulée. C'est, à notre connaissance, le premier modèle permettant de fournir une telle estimation et ce de manière précise.

Il est intéressant de noter que l'ensemble des extensions qui ont trait aux machines virtuelles a été intégré au code de SimGrid dès la version 3.11 en Mai 2014. Bien que nous ayons apporté quelques optimisations, les extensions sont toujours présentes et les interfaces restent inchangées. Il ont notamment permis de construire un outil de plus haut niveau permettant l'étude de système comme Amazon EC2 ou OpenStack. [3] A noter que je n'ai pas participé au développement de cette surcouche, mon activité s'est concentrée sur la mise en œuvre de l'outil VMPlaceS (Lebre, Pastor, and Südholt, 2015), brièvement décrit ci-après.

VMPlaceS est un outil dédié à l'évaluation et la comparaison des algorithmes de placement dynamique de machines virtuelles via simulations. Il fournit plusieurs abstractions facilitant le développement des algorithmes de placement, l'évaluation à large échelle ainsi que plusieurs mécanismes pour permettre une analyse automatique des métriques collectées pendant la simulation. La précision de VMPlaceS a été validée en comparant des résultats obtenus *in-siclo* et *in-vivo* pour l'algorithme centralisée Entropy (Lebre, Pastor, and Südholt, 2015). Nous avons également illustré la pertinence de l'outil en analysant et comparant trois classes d'algorithmes : centralisée, hiérarchique et distribuée. Cette analyse a été, à notre connaissance, la première étude comparant de manière automatique plusieurs stratégies de placement au dessus d'infrastructures composées jusqu'à 1000 machines physiques et 10000 machines virtuelles.

Pour conclure sur cet axe de recherche, je tiens à souligner que bien que le projet ANR INFRA SONGS qui a supporté les activités décrites ci dessus soit terminé, je reste autant que possible impliqué dans la communauté SimGrid (Lebre, Legrand, et al., 2015). De plus, j'ai récemment débuté une nouvelle série d'expérience ayant pour objectif la mise en place d'un modèle permettant d'estimer le temps de démarrage d'une machine virtuelle (NGuyen and Lebre, 2017). Une majeure partie des bibliothèques de simulations ignore simplement le temps nécessaire au démarrage d'une machine virtuelle, considérant que ce dernier est négligeable. Cependant il est facile de comprendre que cette hypothèse est erronée puisque le démarrage d'une machine virtuelle demande d'accéder à des ressources (CPU, entrées/sorties disques et réseaux). Les expériences que nous avons réalisées nous ont permis d'observer que le démarrage qui prend quelques secondes peut durer plusieurs minutes en fonction des applications co-localisées sur la machine physique. Nous avons proposé un premier modèle construit via une régression linéaire de nos observations. Bien que nous devions finaliser la validation de ce modèle en étudiant plusieurs scénarios tels que l'utilisation de solutions de stockages distantes, nous espérons pouvoir rapidement intégrer un premier modèle permettant de prendre en compte le temps de démarrage des machines virtuelles, critère prépondérant également dans les algorithmes de placement.

Comme précédemment les travaux ont donné lieu à des développements open-source disponible pour SimGrid à l"adresse : `http://simgrid.gforge.inria.fr` et pour VMPlaceS à l'adresse : `http://beyondtheclouds.github.ior/DVMS/`).

3 `http://schiaas.gforge.inria.fr`.

A.2.3 Au delà de l'informatique en nuage 2014 -

Pour satisfaire une demande en constante croissance tout en réalisant d'importantes économies d'échelle, les opérateurs d'infrastructures de type informatique en nuage ont privilégié la construction de centres de données toujours plus grands. La limite en taille de ces « méga » centres de données est régie par des aspects liés à la capacité électrique qui peut être délivrée ainsi qu'au système de refroidissement qui sont nécessaires à leur fonctionnement. Pour pallier ces limitations, les opérateurs ont construits ces nouveaux géants dans un nombre limité de lieux stratégiques : c'est à dire des endroits où une importante source d'énergie est disponible, ou bien avec un climat suffisamment froid toute l'année pour bénéficier de techniques dites de refroidissement naturel (Gary Cook, 2013). Bien que cette tendance persiste, les nouveaux usages liés à l'internet des objets (IoT) (Atzori, Iera, and Morabito, 2010), la virtualisation des réseaux (Mijumbi et al., 2016) ou encore l'informatique mobile (MEC) (Fernando, Loke, and Rahayu, 2013) mettent en exergue les limitations de ce modèle. A titre d'exemple, une projection sur les quantités de données qui vont être générées et/ou manipulées par les objets connectés et les applications associées ne pourront être traitées avec les infrastructures actuelles (B. Zhang et al., 2015). Pour traiter les limitations intrinsèques du modèle « centralisé », notamment en terme de latences réseau, les experts académiques et industriels défendent aujourd'hui l'idée d'étendre le modèle d'informatique en nuage à la périphérie du réseau. C'est à dire vers un modèle massivement distribué, composé non plus de quelques « méga » centres de données mais de nombreux « micro » centres placés au travers et en bordure de l'Internet notamment en s'appuyant sur les points de présence réseau déjà existants. Ce nouveau paradigme, appelé "Cloud Massivement Distribué" ou plus récemment Fog/Edge Computing promet de rapprocher les ressources des utilisateurs et donc d'améliorer les performances des applications (Bonomi et al., 2012). Toutefois, l'essor de ce nouveau paradigme passe, d'une part, par la mise en œuvre d'un système de gestion distribuée qui va permettre aux opérateurs d'agréger, de superviser et d'exposer une infrastructure à large échelle massivement répartie, et d'autre part en étendant les interfaces actuelles afin de permettre le développement de nouveaux services qui tireront partie de cette géo-distribution. Je me suis concentré sur ces aspects depuis ces quatre dernières années au travers plusieurs études préliminaires. Ces études m'ont permis de proposer en 2015 avec plusieurs collaborateurs l'Inria Project Lab DISCOVERY. Ce projet que je coordonne rassemble des spécialistes des systèmes de gestion d'infrastructures virtualisées, du réseau et des systèmes pair-à-pair et implique également deux opérateurs réseau majeurs au niveau national, à savoir ORANGE et RENATER. Du point de vue scientifique, la rupture de notre approche par rapport à l'état de l'art réside dans la manière de concevoir la pile logicielle en charge d'opérer ce nouveau type d'infrastructure. Le système que nous visons peut être vue comme un système d'exploitation distribué, permettant une gestion unifiée des machines virtuelles au travers de plusieurs sites. C'est à dire qu'à la différence des solutions qui consiste à proposer un système en charge d'orchestrer un ensemble d'infrastructures indépendantes (approche que nous définissons comme de « haut en bas »), nous défendons une approche de « bas en haut ». Dans cette approche les mécanismes systèmes à plus bas niveaux sont conçues de manière coopérative en s'appuyant le cas échéant sur des algorithmes « pair à pair » comme cela l'a été proposé dans nos propositions de placements de machines virtuelles. Afin d'éviter des efforts de développements conséquents, nous avons choisi d'appliquer cette approche aux principaux services de la solution OpenStack. Bien que ces travaux soient récents, en comparaison avec les deux activités précédentes, j'ai choisi de les présenter dans ce manuscrit pour deux raisons majeures. Tout d'abord, nous avons obtenu des résultats qui méritent d'être présentés au vue de l'intérêt qu'ils ont suscité dans la communauté OpenStack ainsi que la communauté scientifique (Lebre, Pastor, Simonet, et al., 2017; Cherrueau et al., 2017). Ensuite, c'est autour de cet axe de recherche que j'ai

choisi d'articuler mes activités à venir pour les prochaines années. En particulier, une des actions à moyen terme consiste en la création d'une nouvelle équipe de recherche sur Nantes. Cette équipe abordera les problématiques liées aux mécanismes systèmes ainsi qu'aux abstractions applicatives permettant la supervision et l'utilisation des infrastructures d'informatique utilitaire actuelles, en cours de déploiements et à venir.

A.3 Une recherche dirigée par l'expérimentation

A la fin de mon post-doctorat, j'ai réalisé une étude qui avait pour objectif d'analyser dans quelle mesure l'utilisation des machines virtuelles pouvait permettre une gestion plus fine des ressources de l'infrastructure Grid'5000, en particulier des tâches sans garanties (« best effort » en anglais). Grid'5000 est une plateforme d'expérimentation dédiée à la communauté française conduisant des recherches dans les différents domaines rattachés aux systèmes distribués (calcul haute performance, gestion massive des données ou encore informatique en nuage) (Bolze et al., 2006). Les utilisateurs de Grid'5000 accèdent aux ressources en les réservant pour une durée déterminée. Au bout de la période spécifiée, les ressources sont libérées que la tâche soit terminée ou non. Pour permettre de sauvegarder l'état des tâches afin de les redémarrer ultérieurement, nous avons proposé de les encapsuler dans des machines virtuelles (Gallard, Lebre, and Morin, 2010). Bien que l'impact de cette étude est été limité du faite qu'elle n'apportait qu'un incrément technique vis à vis de l'état de l'art (Sotomayor, Keahey, and Foster, 2008), elle a complétement changé ma perspective sur la manière de superviser et d'utiliser les infrastructures distribuées (Gallard, Lebre, Morin, et al., 2012; Lebre, 2010) : le concept de conteneur (ou bac à sable) qu'apportait la virtualisation système allait révolutionner notre domaine et il était critique d'étendre les mécanismes présents dans Grid'5000 pour faciliter les études autour de cette technologie. J'ai proposé dès 2008 de créer un groupe de travail au niveau national autour de cet objectif. Les travaux abordés par ce groupe ont rapidement pu être consolidés par le support de plusieurs actions comme l'action d'envergure Hemerahttps://www.grid5000.fr/mediawiki/index.php/Hemera. (2010-2014) où j'ai pu coordonner les travaux sur la virtualisation. Nous avons notamment mis en œuvre plusieurs mécanismes permettant aux utilisateurs de Grid'5000 d'étudier les problématiques liées à la virtualisation et à l'informatique en nuage (Balouek, Amarie, et al., 2012). En parallèle de ces activités, j'ai organisé plusieurs évènements comme les journées « thèmes émergents » sur les technologies de virtualisation dans les systèmes distribués en 2010 et 2014 sur Nantes.[4] Au niveau international, j'ai eu l'opportunité de co-animer l'atelier ACM «Virtualization Technologies in Distributed Computing » entre 2011 et 2015.[5] Co-localisé avec la conférence ACM HPDC, les ateliers VTDC ont été un lieu d'échanges importants sur les défis associés à cette technologie. En plus de VTDC, j'ai également pris part à plusieurs comités de programmes tels que les conférences ACM SuperComputing, IEEE/ACM CCGrid, Europar, IEEE CloudCom ou encore IEEE IC2E, tous à plusieurs reprises. Etre impliqué significativement dans la communauté a été un élément clé qui m'a permis de renforcer mon expertise et de consolider la vision que j'avais sur la manière de concevoir les prochaines générations d'infrastructures pour l'informatique utilitaire. J'ai été, en particulier, parmi les premiers chercheurs à défendre et à étudier le concept d'une informatique en nuage massivement distribué en périphérie du réseau d'Internet (Bertier et al., 2014). Ce choix s'est révélé être judicieux au vu de l'intérêt que porte la communauté des systèmes distribués mais également celle des réseaux et m'a permis aujourd'hui d'animer plusieurs évènements clés de notre communauté (responsable du comité de programme et co-responsable du domaine Cloud, Fog et Edge Computing, réciproquement pour les conférences IEEE ICFEC 2017, IEEE CloudCom 2017 et 2016 et

4 http://web.emn.fr/x-info/ascola/doku.php?id=internet:jte-virtualization-2010 et http://people.rennes.inria.fr/Adrien.Lebre/PUBLIC/CloudDay/.
5 http://people.rennes.inria.fr/Adrien.Lebre/VTDC/vtdc15.html.

EuroPar 2016). Par ailleurs, j'ai été invité à co-animer l'action transverse « Virtualisation et clouds » du groupe de recherche CNRS « Réseau et Systèmes Distribués » depuis 2015.[6] Cette action a pour objectif de favoriser les collaborations entre les communautés systèmes distribuées et réseaux autour de cette convergence des infrastructures de calculs et de communications. Bien que chaque communauté a proposé de nombreuses avancées scientifiques et technologiques, il est primordial d'aborder dès à présent les problématiques de manière commune afin de proposer un nouvelle génération d'infrastructures capable de satisfaire les besoins des nouvelles applications liées à l' Internet des Objets, l'informatique mobile ou encore la virtualisation des réseaux. Enfin, je souhaite à préciser que je reste fortement impliqué dans la communauté Grid'5000, notamment à travers le comité exécutif et le comité des architectes. Parmi les actions en cours, nous étudions dans quelle mesure Grid'5000 devrait évoluer de manière à pouvoir appréhender les problématiques liées à l'Internet Industriel (également connu sous le nom d'Entreprise du Futur). L'Internet Industriel peut être vue comme une combinaison des centres de données, des réseaux de communications et des équipements rattachés à l'Internet des Objets. Une piste sérieuse est la combinaison des deux infrastructures Grid'5000 et FIT comme abordé plus amplement dans le manuscrit. Organisation générale du document. Le document, rédigé en anglais, porte principalement sur les trois contributions scientifiques évoquées préalablement et des perspectives spécifiques à chacune d'entre elle. Il est important de noter que l'ensemble de ces travaux a déjà été publié dans des conférences ou journaux à portée internationale. Les contenus présentés ici ont été consolidés et mis en forme afin de faire ressortir la cohérence globale de mes activités de recherche. Seul le dernier chapitre évoquent des points réellement nouveau notamment en présentant les actions que j'ai prévues de réaliser sur au moins les 4 prochaines années.

6 https://rsd-cloud.lip6.fr.

DETAILED CURRICULUM VITAE

Adrien Lebre

Inria Researcher
(on leave from Mines Nantes)
PhD in Distributed Systems and HPC File systems

Address: 5 Avenue du moulin de la touche
44240 SUCE SUR ERDRE, FRANCE
Phone :+ 33 (0)6 40 12 95 87
E-mail :adrien.lebre@inria.fr

Website: http://www.emn.fr/x-info/alebre08/

39 years old (born on June 1977, the 28^{th})
French nationality
Married, two children (2003/2009)

Education

2003 - 2006 **PhD in Computer Science** at the "Institut National Polytechnique de Grenoble", France.

Design and development of **an Input/Output scheduling framework for HPC architecture**(C, Linux, kernel space - http://aioli.imag.fr).

Funded by BULL SA company around the *LInux Platform Solution* collaboration with the Grenoble Computer Science Laboratory and supervised by Prof. Brigitte Plateau and co-supervised by Prof. Yves Denneulin (currently vice-head of Grenoble Computer Science Lab) and Pascal Rossé-Laurent (BULL Linux Senior Architect).

2001 - 2002 **MSc in Computer Science** "Communications and Systems", University of Sciences Joseph Fourier, Grenoble, France, **with honours** (get a **merit-based scholarship** from the University).

Evaluations and improvements of **meta-data management** by setting up distributed mechanisms **within the Parallel NFS system** (C, Linux, user space - http://nfsp.imag.fr).

1998 - 2001 **Maîtrise de Mathématiques et d'Informatique**, equivalent to a **Taught Master's Degree** in Computer Science, "Institut Universitaire Professionnalisé" Avignon, France, **with honours**.

Work Experience

Since Oct. 2013 **Contract Researcher** at the **Inria Rennes Bretagne Atlantique center**, France (on leave from a Researcher position at Mines Nantes).

Research activities are done in the context of the Discovery intiative that aims at at overcoming the main limitations of the traditional server-centric cloud solutions by deploying cloud infrastructures directly on top of the network backbone facilities (http://beyondtheclouds.github.io).

2008 - 2013 **Researcher** (" Chargé de Recherche EPA ") at the **Mines Nantes** Engineering School, France.

Research activities are done in the **ASCOLA Research Group**. Major works cover two main areas: the design and the implementation of new **distributed file systems** and the use of **virtualization technologies** in **large scale platforms**.

2006 - 2008 **Post Doc** at **INRIA, the French Research Institute of Computer Science**, Rennes, France.

Design and development of a **symmetric file system for cluster** (C, Linux, kernel space) under the framework of **the EU-funded XtreemOS project** led by Christine Morin (Senior Researcher at INRIA). Further information at http://kerrighed.sourceforge.net/wiki/index.php/KernelDevelKdFS

2002 **3 months**, system developer, Grenoble Computer Science Laboratory, France.

Design and development of a **RAID layer** in the Parallel NFS proposal (C, Linux, user space).

2001 **6 months**, **internship**, CanalNumedia (Vivendi group), **Stockholm, Sweden**.

Design and development of several content management modules for the different Scandinavian Canal+ websites (ASP, XML - SQL Server, IIS).

Teaching Experience

2012 - 2013	In charge of the **Information Systems** module, (JEE, **Play Framework**, ∼35 students).
2008 - 2012	In charge of the **Object-oriented Programming** module, (**JAVA**, ∼100 students) In charge of the **Systems and Networks** module, (∼25 graduate students) including **Introduction to Gnu/Linux and Open-Source Systems** (UBUNTU) and **Operating systems and Networks** tutorials.
2006 - 2012	**Distributed Computing** lectures, MSc in Computer Science, University of Rennes, France.
2006 - 2007	**Computer architecture** lectures, last year of statistics engineering school, ENSAI, Rennes, France. **Operating systems** tutorials, MSc in Computer Science, IFSIC, University of Rennes, France.
2004 - 2006	**Parallel Programming** practical works, MSc and PhD, INPG Doctoral School, Grenoble, France. **System, network and database** tutorials, 2^{nd} year of engineering school, ENSGI, Grenoble, France.

Computer Skills

System	**GNU/Linux** (Debian) use and internal design, **distributed computing** (cluster, grid, cloud). **Distributed** and local **file systems**, **virtualization** technologies. **Simgrid** (co-developper of the VM abstractions), **OpenStack** (Chair of the Massively Distributed WG, Member of the Performance Team).
Programming	**C** (**user/kernel**), **JAVA**, **python**, parallel programming (**thread**, **MPI**), Data Base systems (**SQL/NoSQL**), and multimedia programming (javascript, HTML, XML)

Selected Publications

Journals
Conferences
Workshops

Revising OpenStack to Operate Fog/Edge Computing infrastructure, by A. Lebre, J. Pastor, A. Simonet, and F. Desprez - in Proceedings of IEEE International Conference on Cloud Engineering (IC1E 2017), Canada, Apr. 2017.

Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure,, by B. Confais, A. Lebre and B. Parrein - in **Transactions on Lare-Scale Data and Knowledge-Centered Systems Journal** - Springer, 2017.

SimGrid VM: Virtual Machine Support for a Simulation Framework of Distributed Systems, by T. Hirofuchi, A. Lebre and L. Pouilloux - in IEEE **Transactions on Cloud Computing Journal**, Dec 2015.

Adding Virtualization Capabilities to the Grid'5000 Testbed, by D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lebre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Perez, F. Quesnel, C. Rohr, and L. Sarzyniec - in Cloud Computing and Services Science, volume 367 of Communications in Computer and Information Science, Springer 2013.

Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS, by F. Quesnel, A. Lebre, and M. Sudholt - **Concurrency and Computation: Practice and Experience**, Dec 2012.

Managing Virtual Resources: Fly through the Sky, by J. Gallard and A. Lebre, **ERCIM News**, Oct. 2010

Cluster-wide context switch of virtualized jobs by F. Hermenier, A Lebre, and J.M. Menaud, in Proceedings of the **ACM VTDC** Workshop, co-located with the ACM HPDC Conference, US, June 2010.

I/O Scheduling for Multi-Application Clusters by A Lebre, G. Huard, Y. Denneulin, and P. Sowa, in Proceedings of the **IEEE Cluster** Conference, Spain, **2006**.

Performance Evaluation of a Prototype Distributed NFS Server by R. Avila, P. Navaux, P. Lombard, A. Lebre, and Y. Denneulin, in Proceedings of the IEEE SBAC-PAD Conference, Brasil, Oct. 2004.

Selected Invited Talks

International European

Virtualization and Cloud fundamentals, Initial Training School, MSCA-ETN BigStorage, March 2015, Spain.

How Should Next Generation Utility Computing Infrastructures Be Designed to Solve Sustainability and Efficiency Challenges?, ISC Cloud Conference, Sep 2014, Germany

Challenges and Issues of Next Cloud Computing Platforms at Inria, Post-Consultation Workshop on the H2020 ICT Work Programme, April 2013, Belgium.

Beyond the Cloud: The Discovery Initiative, 6th edition of the VHPC workshop (co-located with EUROPAR 2011), France.

Virtualization Technologies in Distributed Architecture: The Grid5000 Recipe, 4th edition of the ACM VTDC workshop (co-located with HPDC 2010), US.

National

An Overview of Cloud Computing Challenges, Institut Francaise de Bio-Informatique, Cumulo NumBio School, Aussois, May 2015.

Beyond the clouds, NRENs opportunities, RENATER, e-infrastuctures, Paris, March 2015.

The Discovery Initiative, Where we Are?, Joint lab INRIA – Alcatel Lucent Bell Labs, Villarceaux, Jan 2015.

The Discovery Initiative, Telecom Paris Sud, Cloud Computing summer school, Evry, Aug 2013.

How Virtualization Changed the Grid Perspective, From Grid to Cloud workshop, Ecole Normale Superieure de Lyon, Sept 2010.

Research Projects/Activities

On-going

Member of the **EU Marie Sklodowska Curie ETN BigStorage** consortium (∼3.8MEuros, 2015/2019), ASCOLA Representative (∼250KEuros), WP leader. Build on the Marie Curie ITN SCALUS, this network will train future data scientists to deal with the on-going deluge of data by leveraging infrastructures at the convergence of cloud and HPC solutions.

Member of the French **ANR Greco** (∼525KEuros, 2017/2020), ASCOLA Representative (∼100KEuros)).The goal of the GRECO project is to develop a reference resource manager for cloud of things. One of the principal challenges will consist in handling the execution context of the environment in which the cloud of things operates.

Project Investigator of the **DISCOVERY** Initiative - http://beyondtheclouds.github.io

Started 4 years ago, this Open-Science initiative targets the design and the implementation of a **new generation of Utility Computing platforms**. In July 2015, Inria agreed to support this action within the framework of the **Inria Project Lab** (IPL) program until 2019 with a budget of approximately 200/250KEuros per year.
Coordinated by myself, the DISCOVERY IPL involves the ASCOLA, ASAP, AVALON, and MYRIADS Inria Project-Teams, Orange Labs and RENATER.

Member of the **Grid'5000** consortium - http://www.grid5000.org

In charge of the **Grid'5000 Winter School 2012** (Dec 2012, 70 persons).
Virtualization WP leader of the **INRIA Large-scale Initiative Hemera** (until Dec 2014).
Architect committee member of the *Groupement d'Interet Scientifique G5K* (since 2010).
Executive committee member of the *GIS G5K* (since 2014).

Past

Member of the French **ANR Infra SONGS** (∼1.8MEuros, 2012/2016), ASCOLA Representative (∼100KEuros), http://infra-songs.gforge.inria.fr). The SONGS project aims at providing a simulator toolkit for the next generation systems. My involvement consists in extending the SimGrid framework with virtualization abstractions.

Member of the **EU Marie Curie ITN SCALUS** consortium (∼3MEuros, 2009/2013), ASCOLA Representative (∼200KEuros). SCALUS aimed at at elevating education, research, and development inside the storage area with a focus on cluster, grid, and cloud.

Member of the French **CNRS PEPS Virtual Machine Scheduling** Action (∼10Keuros, 2012/2013, ASCOLA Representative). Coordinated by the Tours University, this project aimed at defining VM scheduling fundamentals.

EIT ICT Labs, Cloud Computing Action Line, Act. 10239, MYRIADS/ASCOLA (∼25KEuros, 2011/2012), deployment of major cloudkits on Grid'5000 testbed.

Member of the **EU XtreemOS Project** from 2006-2008. **Project Initiatior** of the kDFS proposal, a symmetric file system for the Kerrighed project (http://www.kerrighed.org, maintainer until 2009)

Coordinator of the **aIOLi** and **NFSp** activities (**Brasil**, **Poland** and **France**) from 2005-2007 (http://{aioli,nfsp}.imag.fr)

Collective Duties

Steering/Program Committees

Organization of the 5th (General Chair), the 6th (Program Chair), and the 7th and 8th (co-general chair) ACM/IEEE **VTDC** Workshop (co-located with HPDC).

Organization of the 1st IEEE **International Conference on Fog and Edge Computing** (Program Chair) (co-located with CCGRID 2017).

PC Member at ACM/IEEE **SC 2017/2016/2013**, ACM/IEEE **HPDC 2017**, ACM/IEEE **CCGRID 2017/2016/2015/2013**, IEEE **CloudCom 2016 (Fog/Edge Computing Co-Track Chair)**/2015/2014, **EuroPar 2016 (Cloud Computing Track Chair)**/2015/2013, IEEE **IC2E 2015/2014**, IEEE BigData 2014/2013, IEEE CGC 2013/2012/2011 Conferences and several workshops related to storage, virtualization and energy concerns in large scale infrastructures (please, see my webpage for an up-to-date list).

Publicity Chair at ACM/IEEE **Cluster 2014**, **Tutorial Chair** at IEEE **IC2E 2014**.

Member of the executive committee of the GDR CNRS RSD "Réseau et Système distribué" and Co-leader of the transversal action **Virtualization and Clouds** of this GDR since 2015.
Co-organizer of the CloudDay events since 2015.
Co-organizer of the 2017 RESCOM Spring School.

Reviewer

International journals such as IEEE **TPDS** and **TCC**, Elsevier **JPDC** and **FGCS** ...

Associate Editor of the IEEE **Transactions on Big Data** journal.

FP7 STREP external reviewers (yearly face-to-face evaluations) 2013/2014/2015

Austrian Science START Program 2013 (equivalent to the French *ANR Jeune Chercheur* Program).

PhD Supervisions

Mohamed Abderrahim, **Large-scale programmable monitoring system**, co-supervised with K. Guillouard (Orange Labs), defense expected by the end of 2018.

Bastien Confais, **Locality in Fog Storage Systems**, co-supervised with B. Parrein, defense expected by the end of 2018.

Linh Nguyen, **Elasticity in Cloud and HPC storage Systems**, co-supervised with M. Südholt, defense expected by the end of 2018.

Jonathan Pastor, **Cloud should be a Distributed Infrastructure**, co-supervised with F. Desprez, achieved in Oct 2016 (now PostDoc at University of Chicago).

Flavien Quesnel, **Dynamic scheduling of virtual machines in large scale systems**, co-supervised with Prof. M. Südholt, achieved in Feb 2013 (now Research Engineer at IRT System X).

PostDoc/Engineer Supervisions

Ronan Alexandre Cherrueau, **Main Engineer of the DISCOVERY Initiative**, Engineer position co-supervised with M. Simonin, Funded by Inria in the context of the MERCURY InriaHub program, Jul 2016/Jul 2019.

Anthony Simonet, **Cost-Benefit Analysis of the DISCOVERY Model**, Postdoc position co-supervised with A-C. Orgerie, Funded by Inria in the context of the Discovery initiative, Oct 2015/June 2017.

Ismael Figueroa, **Staying alive without hearbeats**, Postdoc position, co-supervised with N. Tabareau. Funded by Inria in the context of the Discovery initiative, July 2014/Jan 2015 (now Ass. Prof. University Catholic of Valparaiso, Chile).

Takahiro Hirofuchi, **Adding virtualization abstractions into SimGrid**, PostDoc position, Funded by the ANR SONGS Project, Jan/Dec 2013 (now Full-time Researcher at AIST Japan).

Alexandra Carpen Amarie, **One-Click IaaS Clouds in Grid'5000**, Research Engineer Position. Funded by EIT ICT Labs, Act. 10239, Sept. 2011/April 2012 (now PostDoc at Vienna University of Technology, Austria).

Thesis Committees (external reviewer)

Jérome Gallard, **Flexible management of distributed computing infrastructures**, University of Rennes, May 2011.

Alexandra Carpen Amarie, **BlobSeer as a data storage facility for Clouds: self-adaptation, integration, evaluation**, University of Rennes, Dec 2011.

Sheheryar Malik, **Resource Aware Cloud Computing**, University of Nice-Sophia Antipolis, Dec 2012.

Alexandre Lissy, "**Utilisation de méthodes formelles pour garantir des propriétés de logiciels au sein d'une distribution : exemple du noyau Linux**", University of Tours, March 2014.

Houssem Medhioub, "**Architectures et mécanismes de fédération dans les environnements Cloud Computing et Cloud Networking**", University Pierre et Marie Curie/TelecomSud Paris, April 2015.

Vincent Kherbach, "**Ordonnancement des migrations à chaud de machines virtuelles**", University of Nice-Sophia Antipolis, Dec 2016.

Ismael Cuadrado-Cordero, "**Microclouds: An Approach for a Network-Aware Energy-Efficient Decentralized Cloud**", University of Rennes 1, Feb 2017.

Luis Pineda, "**Efficient Support fo Data-Intensive Scientific Workflows on Geo-Distributed Clouds**", University of Rennes 1, May 2017.

Selection Committee

Telecom Nancy, Associate Professor position, May 2013.

Languages

French	Mother tongue
English	Almost fluent

Referees

Academics

Frédéric Desprez, Senior Researcher at INRIA / **Scientific Director** of the **Grid'5000** platform / **IBM Faculty Award 2008** (**http://graal.ens-lyon.fr/~desprez/**)

Narendra Jussien, Director of IMT Mines Albi Carmaux (head of the C.S. department of the Ecole des Mines de Nantes between 2008 and 2013 / **Google Focused Research Award 2010** (**http://narendra.jussien.free.fr/**)

Christine Morin, Senior Researcher at INRIA / Head of the MYRIADS Project Team (**http://www.irisa.fr/myriads/members/cmorin**)

Additional Information

Social skills	Good public relations (**dynamic**, **communicative** and **open-minded**) . Interested in **working in multicultural teams**.
Organization skills	Experience in **setting-up** projects and external **collaborations**.
Interests	Keen on **skateboarding**, skiing and **snowboarding** for more than **25 years** Running (long-distance), **climbing**, listening music, playing drums
Miscellaneous	Clean **driving licence** **First aid certificate** ("Attestation Française des Premiers Secours" graduated in 1997)

COMPLETE LIST OF PUBLICATIONS

PhD Thesis

[1t06] A. Lebre sous la direction de Brigitte Plateau, *aIOLi : contrôle, ordonnancement et régulation des accès aux données persistantes dans les environnements multi-applicatifs haute performance*, Institut National Polytechnique de Grenoble, spécialité Informatique : systèmes et Logiciels, Sep. 2006.

Book Chapters

[1bc14] M. Bertier, F. Desprez, G. Fedak, A. Lebre, A-C Orgerie, J. Pastor, F. Quesnel, J. Rouzaud-Cornabas, and C. Tedeschi, *Beyond the Clouds: How Should Next Generation Utility Computing Infrastructures Be Designed?*, In Cloud Computing: Challenges, Limitations and R&D Solutions, pp 325-345 - Editors: Z. Mahmood - Computer Communications and Network, Springer, 2014.

[2bc12] F. Alvares, A. Lebre, T. Ledoux, and J.M. Menaud, *Self-management of applications and systems to optimize energy in data centers*,In Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice - Editors: I. Brandic, M. Villari and F. Tuse - IGI Global, Feb 2012.

International Journals

[5j15] B. Confais, A. Lebre and B. Parrein, *Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure*, in Transactions on Large-Scale Data and Knowledge-Centered Systems Journal - Springer, 2017.

[4j15] T. Hirofuchi, A. Lebre and L. Pouilloux, *SimGrid VM: Virtual Machine Support for a Simulation Framework of Distributed Systems*, in IEEE Transactions on Cloud Computing Journal, Dec 2015.

[3ji13] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lebre, D. Margery, N. Niclausse, L. Nusbaum, O. Richard, C. Perez, F. Quesnel, C. Rorh and L. Sarzyniec, *Adding Virtualization Capabilities to the Grid'5000 Testbed*,In Cloud Computing and Services Science, Revised selected papers of CLOSER 2012, Vol 367, pp 3-20 - Editors: I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan - Springer, Feb 2013.

[2ji12] F. Quesnel, A. Lebre, and M. Südholt, *Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS*, Concurrency and Computation: Practice and Experience, Dec. 2012.

[1ji12] J. Gallard, A. Lebre, C. Morin, T. Naughton, S. Scott, and G. Vallée. *Architecture for the Next Generation System Management Tools*, Future Generation Computer Systems, Jan. 2012.

International Conferences

[23ic17] M. Abderrahim, K. Guillouard, M. Ouzzif, J. Francois, and A. Lebre, *An Holistic Monitoring Service for Fog/Edge Infrastructures: a Prospective Analysis*, in *Proceedings* of the IEEE International Conference on Future Internet of Things and Cloud (FiCloud 2017), Czech Republic, August 2017.

[22ic17] R-A. Cherrueau, D. Pertin, A. Simonet, M. Simonin, and A. Lebre, *ENOS: a Holistic Framework for Conducting Scientific Evaluations of OpenStack*, short paper in *Proceedings* of the IEEE/ACM Symposium on Cluster, Cloud and Grid Computing (CCGRID 2017), Spain, May 2017.

[21ic17] B. Confais, A. Lebre, and B. Parrein, *An Object Store Service for a Fog/Edge Computing Infrastructure based on IPFS and Scale-out NAS*, in *Proceedings* of the IEEE Conference on Fog and Edge Computing (ICFEC'17), Spain, May 2016.

[20ic17] A. Lebre, J. Pastor, A. Simonet, and F. Desprez, *Revising OpenStack to Operate Fog/Edge Computing Infrastructures*, in *Proceedings* of the IEEE Conference on Cloud Engineering (IC2E'17), Canada, April 2017.

[19ic17] L. T. N'Guyen, and A. Lebre, *Virtual Machine Boot Time Model*, in *Proceedings* of the Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP'17), Russia, March. 2017.

[18ic16] B. Confais, A. Lebre, and B. Parrein, *Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures*, in *Proceedings* of the IEEE Conference on Cloud Computing Technology and Science (CloudCom'16), Luxembourg, Dec. 2016

[17ic15] A. Lebre, J. Pastor, and M. Südholt, *VMPlaceS: A Generic Tool to Investigate and Compare VM Placement Algorithms*, in *Proceedings* of the Europar Conference, LNCS 9233, Austria, August 2015.

[16ic15] A. Lebre, A. Legrand, F. Suter and P. Veyre, *Adding Storage Simulation Capacities to the SimGrid Toolkit: Concepts, Models, and API*, in *Proceedings* of the 15th IEEE/ACM Symposium on Cluster, Cloud and Grid Computing (CCGRID 2015), China, May 2015.

[15ic14] J. Pastor, M. Bertier, F. Desprez, A. Lebre, F. Quesnel and C. Tedeschi, *Locality-aware Cooperation for VM Scheduling in Distributed Clouds*, in *Proceedings* of the Europar Conference, LNCS 8632, Portugal, August 2014.

[14ic13] T. Hirofuchi, A. Lebre, and L. Pouilloux, *Adding a Live Migration Model Into Simgrid*, in *Proceedings* of the IEEE Conference on Cloud Computing Technology and Science (CloudCom'13), pp 96-103, UK, Dec. 2013.

[13ic13] T. Hirofuchi, A. Lebre, *Adding Virtual Machine Abstractions Into SimGrid, A First Step Toward the Simulation of Infrastructure-as-a-Service Concerns*, in *Proceedings* of the IEEE International Conference on Cloud and Green Computing (CGC'13), Germany, Sept 2013.

[12ic13] F. Quesnel, A. Lebre, J. Pastor, M. Südholt, and D. Balouek, *Advanced Validation of the DVMS Approach to Fully Distributed VM Scheduling*, in *Proceedings* of the IEEE International Symposium on Parallel and Distributed Processing with Application (ISPA'13), Australia, July 2013.

[11ic11] F. Quesnel, and A. Lebre, *Operating Systems and Virtualization Frameworks: From Local to Distributed Similarities*, in proceedings of the Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP'11, session: Virtualization in Distributed Systems), Cyprus, Feb. 2011.

[10ic10] J. Gallard, A. Lebre, and Christine Morin, *Saline: Improving Best-Effort Job Management in Grids*. in *Proceedings* of the Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP'10 session: Virtualization in Distributed Systems), Italy, Feb. 2010.

[9çic10] J. Gallard, C. Morin, G. Vallée, T. Naughton, S. Scott, L., and A. Lebre, *Architecture for the Next Generation System Management Tools*, in *Proceedings* of the International Conference on Utility and Cloud Computing (UCC 2010), India, Dec. 2010.

[8ic10] J. Gallard, A. Lebre, G. Vallée, C. Morin, P. Gallard, and S. L. Scott, *Refinement Proposal of the Goldberg's Theory*, in *Proceedings* of the International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'09), LNCS 5574, Taiwan, 2009.

[7ic09] P. Riteau, A. Lebre, and C. Morin, *Handling Persistent States in Process Checkpoint/Restart Mechanisms for HPC Systems*, in *Proceedings* of the IEEE/ACM International Symposium on Cluster Computing and Grid (CCGRID 2009), China, 2009

[6ic08] A Lebre, R. Lottiaux, E. Focht, and C. Morin, *Reducing Kernel Development Complexity in Distributed Environments*, in *Proceedings* of the EUROPAR Conference, LNCS 5168, Spain, 2008.

[5ic06] A. Lebre, G. Huard, Y. Denneulin, and P. Sowa, *I/O Scheduling Service for Multi-Application Clusters*, in *Proceedings* of the IEEE Cluster Conference, Spain, Sep 2006.

[4ic05] A. Lebre, and Y. Denneulin, *aIOLi: An Input/Output LIbrary for cluster of SMP*, in *Proceedings* of the IEEE/ACM *Cluster Computing and Grid* Conference (CCGRID 2005), United Kingdom, May 2005.

[3ic04] R.B. Avila, P.O.A. Navaux, P. Lombard, A. Lebre, and Y. Denneulin, *Performance Evaluation of a Prototype Distributed NFS Server*, in *Proceedings* of the IEEE *Computer Architecture and High Performance Computing* Conference (SBAC-PAD), Brazil, 2004.

[2c03] O. Valentin, P. Lombard, A. Lebre, C. Guinet, and Y. Denneulin, *Distributed File System for Clusters and Grids*, in *Proceedings* of the *Parallel Processing and Applied Mathematics* Conference, LNCS 3019, Poland, 2003.

[1ic03] P. Lombard, Y. Denneulin, O. Valentin, and A. Lebre, *Improving the Performances of a Distributed NFS Implementation*, in *Proceedings* of the *Parallel Processing and Applied Mathematics* Conference, LNCS 3019, Poland, 2003.

International Workshops

[8iw14] A. Lebre, A. Simonet and A-C Orgerie, *Deploying Distributed Cloud Infrastructures: Who and at What Cost?*, in *Proceedings* of the 5th IEEE International Workshop on Cloud Computing Interclouds, Multiclouds, Federations, and Interoperability (InterCloud'15, co-located with IEEE IC2E'15), Germany, April 2015.

[7iw14] G. B. Brand, and A. Lebre, *GBFS: Efficient Data-Sharing on Hybrid Platforms. Towards adding WAN-Wide elasticity to DFSes*, in *Proceedings* of the Workshop on Parallel and Distributed Computing for Big Data Applications (WPBA'14, co-located with IEEE SBAC-PAD'14), France, Oct 2014.

[6iw13] M. Imbert, L. Pouilloux, J. Rouzaud-Cornabas, A. Lebre and T. Hirofuchi, *Using the EXECO toolbox to perform automatic and reproducible cloud experiments*, in *Proceedings* of the International Workshop on UsiNg and building ClOud Testbeds (UNICO'13, co-located with IEEE CloudCom'13), UK, Dec. 2013.

[5iw13] S. Badia, A. Carpen-Amarie, A. Lebre, and L. Nussbaum, *Enabling Large-Scale Testing of IaaS Cloud Platforms on the Grid'5000 Testbed*, in *Proceedings* of the International Workshop on Testing The Cloud (TTC'13, co-located with ACM ISSTA 2013), Switzerland, July 2013.

[4iw11] A. Lebre, P. Anedda, M. Gaggero, and F. Quesnel, *DISCOVERY, Beyond the Clouds - DIStributed and COoperative framework to manage Virtual EnviRonments autonomicallY: a prospective study*, in *Proceedings* of the Virtualization for High Performance Cloud Computing workshop (colocated with EUROPAR 2011), LNCS 6853 , France, August 2011.

[3iw11] F. Quesnel, and A. Lebre, *Cooperative Dynamic Scheduling of Virtual Machines in Distributed Systems*, in *Proceedings* of the Virtualization for High Performance Cloud Computing workshop (colocated with EUROPAR 2011), LNCS 6853 , France, August 2011.

[2iw11] F. Hermenier, A. Lebre, and J. M. Menaud, *Cluster-Wide Context Switch of Virtualized Jobs*, in *Proceedings* of the International ACM Workshop on Virtualization Technologies in Distributed Computing (colocated with ACM HPDC 2010), US, June 2010.

[1iw08] J. Gallard, G. Vallée, A. Lebre, C. Morin, P. Gallard, and S. L. Scott, *Complementarity between Virtualization and Single System Image Technologies*, in *Proceedings* of the Virtualization in High-Performance Cloud Computing workshop (colocated with EUROPAR 2008), LNCS LNCS 5415 Spain, 2008.

International Communications (with reviewing process)

[9icom16] C. Collicutt, A. Lebre, C. Huang *When One Cloud is Not Enough: An Overview of Sites, Regions, Edges, Distributed Clouds, and More*, *OpenStack Summit*, Boston, USA, May 2017.

[8icom16] R. A. Cherrueau, A. Lebre, P. Riteau *Toward Fog, Edge and NFV deployments – Evaluating OpenStack WAN-wide*, *OpenStack Summit*, Boston, USA, May 2017.

[7icom16] A. Lebre, J. Pastor, M. Simonin and T. Carrez, *A Ring to Rule Them All - Revising OpenStack Internals to Operate Massively Distributed Clouds*, *OpenStack Summit*, Austin, USA, April 2016.

[6icom15] F. Desprez, S. Ibrahim, A. Lebre, A-C. Orgerie, J. Pastor and A. Simonet, *Energy-Aware Massively Distributed Cloud Facilities: the DISCOVERY Initiative*, in *Proceedings* of the 11th IEEE international Conference on Green Computing and Communications (GreenCom 2015), poster session, short paper, Sydney, Australia, Dec 2015.

[5icom13] F. Quesnel, D. Balouek, and A. Lebre, *Deploying and Scheduling Thousands of Virtual Machines on Hundreds of Nodes Distributed Geographically*, in the IEEE International Scalable Computing Challenge (SCALE 2013) (colocated with CCGRID 2013), Netherlands, May 2013.

[4icom13] A. Lebre, and F. Desprez *Challenges and issues of next cloud computing platforms at Inria*, Post-Consultation Workshop on the H2020 ICT Work Programme 2014.

[3icom12] G. Brand, and A. Lebre, *Mitigating Network Impact in Large Scale DFSes*, in *Proceedings* of the USENIX Conference on File and Storage Technologies (Poster Session, short paper), Feb 2012.

[2icom10] J. Gallard, and A. Lebre, *Managing Virtual Resources: Fly through the Sky*, ERCIM News, Oct. 2010

[1icom06] A. Lebre, Y. Denneulin, G. Huard, and P. Sowa, *Adaptive I/O Scheduling for Distributed Multi-applications Environments*, in *Proceedings* of the IEEE International Symposium on High Performance Distributed Computing (HPDC 2006), ooster Session, short paper, Paris, France, June, 2006.

National Journals

[1fj08] A. Lebre, G. Huard, and Yves Denneulin, *Optimisation des E/S disques dans les environnements multi-applicatifs distribués*, Technique et Science Informatiques (TSI), Feb 2008.

National Conferences

[5fc16] B. Confais, A. Lebre, and B. Parrein, *Quel système de stockage pour les architectures Fog ?*, in *Proceedings* of the Compas Conference (Conference sur l'informatique en Parallélisme, Architecture et Système), Jul. 2016.

[4fc09] L. Eyraud-Dubois, A. Lebre, P. Martineau, A. Soukhal, V. T'kindt and D. Trystram, *A Server Consolidation Problem: Definition and Model*, in *Proceedings* of the Roadef Conference, Feb 2013.

[3fc09] F. Hermenier, A. Lebre, and J.M. Menaud, *Changement de contexte pour tâches virtualisées à l'échelle de grappes*, in *Proceedings* of the ACM-ASF CFSE Conference, Sept. 2009.

[2fc06] A. Lebre, G. Huard, and P. Sowa, *Optimisation des E/S avec QoS dans les environnements multi-applicatifs distribués*, in *Proceedings* of the ACM-ASF RenPar Conference, Oct. 2006.

[1fc05] A. Lebre, and Y. Denneulin, *aIOLi : gestion des Entrées/Sorties parallèles dans les grappes SMPs*, in *Proceedings* of the ACM-ASF *RenPar* Conference, May 2005.

Miscellaneous

[13misc16] Adrien L., and F. Desprez *Research Issues for Future Cloud Infrastructures*, European Commission, Belgium, Nov. 2016

[12misc16] R-A. Cherrueau, D. Pertin, A. Simonet, M. Simonin, and Adrien L. *ENOS: a Holistic Framework for Conducting Scientific Evaluations of OpenStack*, INRIA Technical Report RR-485, Nov 2016.

[11misc15] A. Lebre, J. Pastor, F. Desprez,*The DISCOVERY Initiative - Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilities*, Inria Research Report RR 8779, Sep 2015.

[10misc12] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lebre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, *Adding Virtualization Capabilities to Grid'5000*, INRIA Research Report RR-8026, July 2012.

[9misc10] J.M. Menaud, A. Lebre, T. Ledoux, J. Noyé, P. Cointe, R. Douence, and M. Südholt, *Vers une réification de l'énergie dans le domaine du logiciel*. In Journées du GDR Génie de la Programmation et du Logiciel, France, March 2010

[8misc09] , J. Gallard, A. Lebre, and C. Morin, *Saline: Improving Best-Effort Job Management in Grids*, INRIA Research Report RR-7055, 2009.

[7misc08] A. Lebre, R. Lottiaux, E. Focht, and C. Morin, *Reducing Kernel Development Complexity in Distributed Environments*, INRIA Research report RR-6405, Jan 2008.

[6misc07] J. Gallard, A. Lebre, G. Vallée, P. Gallard, S.L. Scott, and C. Morin, *Is Virtualization Killing Single System Image Research*, INRIA Research Report RR-6389, Dec. 2007.

[5misc07] XtreemOS consortium, *Design and Implementation of High Performance Disk Input/Output Operations in a Federation*, Delivrable XtreemOS D2.2.5 (35 pages), Dec. 2007, (http://www.xtreemos.eu).

[4misc06] XtreemOS consortium, *Specification of Federation Resource Management Mechanisms*, Delivrable XtreemOS D2.2.1 (67 pages), Dec. 2006, (http://www.xtreemos.eu).

[3misc05] A. Lebre, Y. Denneulin, and T.T. Van, *Controlling and Scheduling Parallel I/O in a Multi-applications Environment*, INRIA Research Report, RR-5689, Sept. 2005.

[2misc05] A. Lebre, and Y. Denneulin, *aIOLi : gestion des Entrées/Sorties Parallèles dans les grappes SMP*, INRIA Research Report RR-5522, March 2005.

[1misc02] A. Lebre, *Composition de service de données et de méta-données dans un système de fichiers distribué*, Master thesis, June 2002.

INVITED TALKS

International Audience

[6ipres15] *Virtualization and Cloud fundamentals*, Initial Training School, MSCA-ETN BigStorage, Spain March 2016.

[5ipres14] *How Should Next Generation Utility Computing Infrastructures Be Designed?*, ISC Cloud Conference, Germany, Sep 2014.

[4ipres13] *Challenges and Issues of Next Cloud Computing Platforms at Inria*, Post-Consultation Workshop on the H2020 ICT Work Programme, Belgium, April 2013.

[3ipres13] *Beyond the Cloud: The Discovery Initiative*, 6th edition of the VHPC Workshop (co-located with EUROPAR 2011), France Aug 2011.

[2ipres11] *Dynamic Scheduling of Virtual Machines*, EIT ICT Lab/Contrail Cloud Computing Summer School, France, June 2011.

[1ipres10] *Virtualization Technologies in Distributed Architectures: The Grid'5000 Recipe*, 4th Edition of the ACTM VTDC Workshop (co-located with HPDC 2010), USA, June 2010.

National Audience

[9fpres16] *Revising OpenStack to operate the next generation of Cloud Computing platforms*, CargoDay - La virtulisation: état des lieux, Nantes , Oct 2016.

[8fpres15] *The Discovery Initiative, Where We Are?*, Inria Alcatel Lucent Bell Labs, Villarceaux, Jan 2015.

[7fpres15] *The Discovery Initiative, Would OpenStack be the solution?*, Journée SUCCESS - France Grille, Paris, Nov 2015.

[6fpres15] *An Overview of Cloud Computing Challenges*, Institut Francaise de Bio-Informatique, Cumulo NumBio School, Aussoy May 2015.

[5fpres15] *Beyond the clouds, NRENs opportunities*, RENATER e-Infrastructures, Paris, Jan 2015.

[4fpres15] *The Discovery Initiative, Where We Are?*, Inria Alcated Lucent Bell Labs, Villarceaux, Jan 2015.

[3fpres13] *The Discovery Initiative*, Cloud Computing Summer School, Evry, France, Aug 2013.

[2fpres11] *Cloud and Virtualization*, France Grille, Cloud Workshop, Lyon, France, Oct 2011.

[1fpres10] *How Virtualization Changed the Grid Perspective*, From Grid to Cloud Workshop, Lyon, France, Sept 2010.

Version: 2017-10-19 23:46